МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ В.О.СУХОМЛИНСЬКОГО

Кафедра комп'ютерних наук та прикладної математики

**Н. А. Махровська, Г. С. Погромська,
О. С. Булгакова, В. В. Зосімов**

Програмування:
Мова програмування С++

Навчально-методичний посібник

*для студентів спеціальності
122 Комп'ютерні науки та інформаційні технології*

Миколаїв – 2017

ББК 22.18 я 73
УДК 004.43(075)
М 36

РЕЦЕНЗЕНТИ:

К.В. Кошкін	доктор технічних наук, професор, директор Навчально-наукового інституту комп'ютерних наук та управління проектами Національного університету кораблебудування імені адмірала Макарова
І.В. Кулаковська	кандидат фізико-математичних наук, доцент кафедри інтелектуальних інформаційних систем Чорноморського національного університету імені Петра Могили

Рекомендовано до друку рішенням
Вченої ради Миколаївського національного університету
імені В.О.Сухомлинського (протокол № 9 від 15 травня 2017р.)

Махровська Н.А.

М 36 Програмування: мова програмування С++. Навчально-методичний посібник / Н. А. Махровська, Г. С. Погромська, О. С. Булгакова, В. В. Зосімов – Миколаїв: ., 2017. – 273 с.

ISBN

У навчально-методичному посібнику розглянуто основи алгоритмізації та програмування на прикладі мови програмування С++. Структура містить теоретичні відомості та лабораторні роботи для вивчення базових алгоритмічних структур та основних структур даних на підтримку вивчення дисципліни «Програмування».

Посібник буде корисним студентам усіх напрямків підготовки, що пов'язані з інформатикою, програмуванням та автоматизацією виробництва.

ББК 22.18 я 73
УДК 004.43(075)
М 36

ISBN

З М І С Т

П Е Р Е Д М О В А

Програмування – досить молодий розділ інформатики, який включає в себе процес проектування, написання, налагодження та тестування комп'ютерних програм. Даний навчально-методичний посібник присвячений розвитку саме таких навичок і містить теоретичні відомості, лабораторні роботи, індивідуальні завдання, матеріал для самостійної роботи та контрольні питання для студентів механіко-математичних факультетів. Може бути використаний при вивченні дисциплін «Програмування», «Інформатика», «Алгоритми і структури даних», «Алгоритмізація і програмування», тощо. Посібник також може бути корисним для тих, хто бажає вивчати мову програмування С++, для викладачів, студентів педагогічних та класичних спеціальностей, школярів.

Зміст посібника містить теми для вивчення основ програмування мовою С++: початкові поняття програмування, основні алгоритмічні структури, принципи роботи з пам'яттю та елементарні структури даних у мові С++, стандартні функції та функції користувача, структури та об'єднання, роботу з класичними рядками та основи бібліотеки STL.

Програмування займає важливу роль в житті людства. Стабільний ріст витрачених ресурсів та людино-годин на створення програмних продуктів підтверджують важливу роль даної справи в повсякденні. Заводи, різні види бізнесу, надання послуг, виробництво товарів та багато інших сфер економіки використовують продукти програмного забезпечення. Основною причиною популярності даного товару є те, що за допомогою нього можна заощадити масу часу та зробити виконання будь-яких завдань більш оптимізованими та точними. Будь-які процеси, що повторюються періодично, або ж виконуються по чітко спланованій схемі можливих варіантів розвитку подій, можна автоматизувати за допомогою програмного забезпечення. Однією з найбільших переваг при автоматизації та комп'ю-

теризації виконання певних дій, на виробництві чи при наданні послуг є те, що людський фактор зменшується до мінімуму, а він, як відомо, є основною причиною помилок, збоїв та нещасних випадків.

В залежності від роду завдань, які необхідно вирішувати, було створено багато різних видів і підвидів мов програмування, кожна з яких більш ефективно здатна виконувати ті чи інші завдання, та керувати різними приладами чи пристроями. Кожна з мов виникла внаслідок чіткої її необхідності. Наприклад керування різними приладами та пристроями здійснюється за допомогою мікропроцесорів, які «прошиваються» програмою на мові C# або ж асемблер, веб-сайт можна створити за допомогою таких засобів як HTML, PHP та MySQL, а мова С була розроблена для системного програмування. А в подальшому її доповнили великою кількістю класів і назвали С++. Мова програмування С++ довгий час була найпопулярнішою серед програмістів.

Набір запропонованих тем охоплює всі аспекти вивчення основ програмування мовою С++. Кожна тема супроводжується прикладами, завданнями та питаннями для самоконтролю.

Умовні позначення:

Лексемою називається… – так у посібнику виділяються важливі моменти (означення, правила, тощо).

Частина 1
Основи програмування на С++

Тема 1. Основні поняття програмування.

§ 1.1. Історія розвитку програмування

Перші мови програмування з'явилися задовго до появи перших комп'ютерів. А перші програми взагалі призначалися для керування ткацькими станками. У 1804 році Жозеф Марі Жаккар розробив ткацький верстат, на якому вишивали виготовляли крупно узорчасті тканини. Узор визначався перфокартами. Зміна візерунка не вимагала змін в механіці верстата, а лише замінювалася серія карт. Це було важливою віхою в історії програмування.

Історія програмування починається з 20-х років XIX століття, коли англійський дослідник Чарльз Беббідж висунув ідею про попередній запис дій обчислювальної машини. У 1833 р. Беббідж розробив проект універсальної цифрової обчислювальної машини – прообразу сучасного комп'ютера. Аналітична машина Беббіджа складалася з трьох частин: пристрою введення вихідних даних, системи обробки даних (тобто обчислювального пристрою) і пристрою виведення кінцевих результатів. Сучасний комп'ютер складається з таких же трьох головних блоків. Це був проект пристрою загального призначення, із застосуванням перфокарт в якості носія вхідних даних і програми.

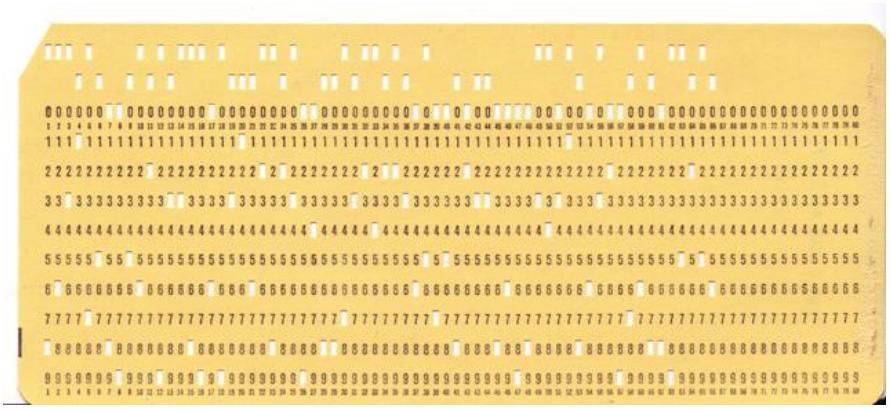


Рис.1. Приклад готової перфокарти

Перфокарта (англ. punched card) – носій інформації, призначений для використання в ранніх системах автоматизованої обробки даних. Виготовлялась з цупкого паперу, мала товщину біля 0.18мм, ширину 82.5мм, довжину 187.3мм.

Інформація кодувалась просічками (перфорацією) в певних позиціях карти. Ранні версії перфокарт мали 12х45, пізніші – 12х80 позицій (перше число – кількість рядків, друге – кількість колонок). Стандартна перфокарта, що використовувалась в ЕОМ та табуляторах — прямокутник із одним скошеним кутом. Позиції позначалися цифрами від 0 до 9 і слугували для кодування відповідних десяткових цифр. Дві верхні позиції залишалися непронумерованими і використовувались як додаткові до цифрових для кодування інших символів з кодового набору ЕОМ. Інформаційна ємність однієї карти становила відповідно 45 або 80 символів з кодами від 0 до 255. Перфокарти стали широко застосовуватися в кінці XIX ст. і служили майже до кінця XX ст. Навіть ЕОМ остаточно позбулися перфокарт лише в 1980-х рр..

Разом з Ч. Беббіджем працювала його сучасниця Ада Лавлейс (дочка великого англійського поета Джорджа Байрона), яку називають першим у світі програмістом. Вона ввела в науку програмування терміни і поняття, що застосовуються і донині. На честь першої програмістки була названа мова програмування Ада, розроблена у 1980 році в Міністерстві оборони США.

Адою Лавлейс були створені три перші в світі обчислювальні програми, складені нею для машини Беббіджа. Найпростіша з них і найбільш докладно описана – програма розв’язування системи двох лінійних алгебраїчних рівнянь з двома невідомими. При розборі цієї програми було вперше введено поняття *робочих комірок* (робочих змінних) і використана ідея послідовного зміни їх змісту. Від цієї ідеї залишається один крок до оператора присвоєння – однієї з основних операцій всіх мов програмування. Друга програма була складена для обчислення значень тригонометричної функції з багаторазовим повторенням заданої послідовності обчислювальних операцій; для цієї процедури Лавлейс ввела поняття циклу – однієї з

фундаментальних конструкцій програмування. У третій програмі, що призначалась для обчислення чисел Бернуллі, були використані рекурентні вкладені цикли. Також Лавлейс висловила припущення про те, що обчислювальні операції можуть виконуватися не тільки з числами, але і з іншими об'єктами, без чого обчислювальні машини так би і залишилися всього лише потужними швидкодіючими калькуляторами.

Людству потрібно було прожити ще понад століття, щоб зрозуміти великий сенс і значення ідеї Бебіджа. Він значно випередив свій час. Комітет Британської наукової асоціації після смерті Бебіджа зробив висновок, що "... Можливості аналітичної машини простираються так далеко, що їх можна порівняти тільки з межами людських можливостей ... Успішна реалізація машини може означати епоху в історії обчислень ...".

§ 1.2. Програми і мови програмування.

Що таке програма і коли з'явилося це поняття?

У XX столітті з'явилися цифрові комп'ютери. Перші програми, що для них призначалися, записувалися в машинних кодах. Програміст, щоб написати правильну програму, повинен був у деталях уявляти собі роботу обчислювальної машини та досконало володіти мистецтвом програмування і набором машинних кодів, оскільки такий процес був досить важким.

Машинні коди – система команд (набір двійкових кодів) для роботи конкретної обчислювальної машини, яка інтерпретується безпосередньо центральним процесором або мікропрограмами лише цієї обчислювальної машини [1]

*Машинний код іноді називають *нативним кодом* (також *власним* або *рідним кодом* – від англ. *native code*), коли мова йде про платформо-залежні частини мови або бібліотеки [2].

Таке написання програм сильно затрудняло спілкування людини з комп'ютером, тому програмісти стали серйозно задумуватися над більш ефективним кодуванням програм. Вони прийшли до висновку, що програму краще було б складати на мові, більш доступній людині, ніж мова

машинних команд. Таким чином, виникла необхідність у створенні штучної мови програмування, яка була б інтуїтивно зрозуміла людині-програмісту.

Мова програмування – це штучна мова написання команд, що призначені для виконання на комп'ютері.

Мова програмування складається з лексичних, синтаксичних та семантичних правил:

- фіксований словник (лексика);
- правила написання команд (синтаксис);
- конструкції мови (семантика).

Ці правила визначають зовнішній вигляд програми і дії, які виконає обчислювальна машина під її керуванням

Оскільки мова програмування обчислювальній машині не зрозуміла, то потрібна спеціальна програма, яка б перекодовувала символи такої мови в двійкові символи машинних команд. Така програма для перекладу символів, або, простіше кажучи, *транслятор* (від англ. *translation* − переклад), була створена на початку 50-х років 20 століття американською програмісткою та одночасно першою і єдиною жінкою-контр-адміралом військово-морських сил США Грейс Хоппер (він називався A Compiler, а перша його версія A-0).

Транслятор – це програма, що перекладає команди мови програмування в машинні коди.

З винаходом трансляторів роль машинних команд у програмуванні стала різко зменшуватися.

У 50-60-і роки почали створюватися мови програмування високого рівня.

Зазначимо, що мови, близькі до числового коду процесора, називають *мовами низького рівня*, а мови, зручні для людини, – *мовами високого рівня*.

Мова найнижчого рівня – мова машинного кодування. Трохи вище лежить мова Асемблера, у якій машинні команди заміняються мнемонічними скороченнями, це дозволяло за текстом програми швидше зрозуміти

на змістовному рівні, яку саме послідовність дій описує складена програма. Мова Асемблера, як і машинні коди, є машинно-залежною мовою. Всі інші мови програмування є мовами більш високого рівня, ніж мова Асемблера, так як вони близькі до розмовних мов, оскільки дають змогу записувати команди у вигляді речень. Приклади таких мов: FORTRAN, BASIC, Pascal, C (мови процедурного типу); Пролог (мова логічного програмування); Visual Basic, Delphi, C++ Builder (середовища візуального програмування), C++, Java (об'єктно-орієнтовані мови).

Перші транслятори призначалися для мов Асемблера (мов низького рівня).

Програма мовою програмування записується в текстовому редакторі і називається початковим кодом (текстом). Цей код складається зі спеціальних команд – операторів мови програмування. За допомогою перекладача-транслятора вихідний код програми перетворюється в машинний код. Тепер уточнимо поняття програми.

Програма – це упорядкований список команд, написаних на мові програмування за її правилами і призначених для виконання на комп'ютері.

Сам процес складання програм називається *програмуванням*. Програмування в даний час виросло в самостійну наукову дисципліну, його в тій або іншій мірі вивчають у школах, коледжах, вузах.

§ 1.3. Компілятори й інтерпретатори.

Отже, як було зазначено вище, для перекладу тексту програми на мову, зрозумілу комп'ютеру, повинна існувати окрема програмна оболонка – *транслятор*.

Трансляція – перетворення програми, представленої на якій-небудь машиннонезалежній мові програмування в еквівалентну форму на машинній мові конкретної ЕОМ.

Трансляцію програм виконують за допомогою програм – трансляторів. Транслятори бувають двох видів: *компілятори* й *інтерпретатори*.

З точки зору виконання роботи компілятор і інтерпретатор істотно відрізняються. Якщо мета трансляції – перетворення всього вихідного тексту на внутрішній мову комп'ютера (тобто одержання деякого нового коду) і тільки, то така трансляція називається також *компіляцією*. Оригінальний текст називається також вихідною програмою або вихідним модулем, а результат компіляції – об'єктним кодом або об'єктним модулем. Якщо ж трансляції піддаються окремі оператори вихідних текстів і при цьому отримані коди відразу виконуються, така трансляція називається *інтерпретацією*. Оскільки трансляція виконується спеціальними програмними засобами (трансляторами), останні носять назву компілятора або інтерпретатора, відповідно.

Компілятор перетворює вихідний код програми в машинну мову конкретного процесора, тобто мову нулів і одиниць. До отриманого коду підключаються стандартні процедури, що використовувалися програмістом. У результаті виходить працююча програма – її називають робочим кодом. Файли таких програм, як правило, мають розширення .exe і представляють собою програму в двійковому коді. Компілятор – тип транслятора, що виконує перетворення всієї програми цілком з якої-небудь мови програмування на мову машинних кодів (абсолютний модуль) або близький до нього (об'єктний модуль).

Інтерпретатори обробляють текст не весь одразу, а безпосередньо під час виконання програми. Інтерпретатори є програмами-посередниками, які читають команди з файлу і переводять їх на мову процесора під час

виконання програми. Інтерпретатор – тип транслятора, що здійснює покомандний (пооператорний) переклад і одночасне виконання програми.

Щоб краще зрозуміти відмінність між компілятором і інтерпретатором, наведемо порівняння. Робота компілятора схожа на роботу літературного перекладача, який спочатку читає текст, аналізує його, а потім виконує переклад. Роботу інтерпретатора можна порівняти з роботою синхронного перекладача, який виконує переклад одразу в міру озвучування тексту. Зрозуміло, що літературний переклад буде більш якісним, ніж переклад синхронний. Аналогічно код, отриманий при компіляції, буде більш компактний і ефективним, ніж код інтерпретатора. У результаті відкомпільовані програми працюють у 20-50 разів швидше, ніж виконувані під керуванням інтерпретатора. Крім того, компілятори під час своєї роботи використовують менше ресурсів комп'ютера.

Іноді, коли час виконання програми не занадто критичний, зручно використовувати мову, що інтерпретується. Найпростіша з них – це мова програмування Basic. Іншими прикладами інтерпретуємих мов є JavaScript і VBScript, які широко використовуються при створенні Web-сторінок, доступних через Інтернет.

В 1982 р. Філіп Кан (засновник фірми Borland) – розробляє транслятор мови Pascal для ПК Apple. Починаючи з 1983 р. Фірма Borland International випустила більше 10 версій компілятора Turbo Pascal. Найпоширеніші з них – версії 3.0, 5.5, 6.0, 7.0. В 1992 фірма випустила новий продукт – компілятор Borland Pascal with Objects 7.0. (для візуального програмування)

Мова програмування С імовірно, є найпопулярнішою у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, котрі її знають. Версії компіляторів для мови С існують для багатьох операційних систем та апаратних архітектур.

Одним з найвідоміших наборів розробника є MinGW – Minimalist GNU for Windows, це колекція безкоштовних С і С ++ компіляторів GCC, призначених для створення програмного забезпечення для системи

Windows. Він надає повний набір інструментів для компілювання і лінковки програм без використання будь-яких платних бібліотек або інструментів. MinGW включає в себе GCC-компілятори, асемблер, лінковщик, архіватор, комплект бібліотек і заголовків файлів та інше.

GCC – GNU Compiler Collection, перший С-компілятор з відкритим вихідним кодом, він був випущений в 1987 р Можливість створювати С++ програми була додана в 1992 р.

§ 1.4. Модель компілятора.

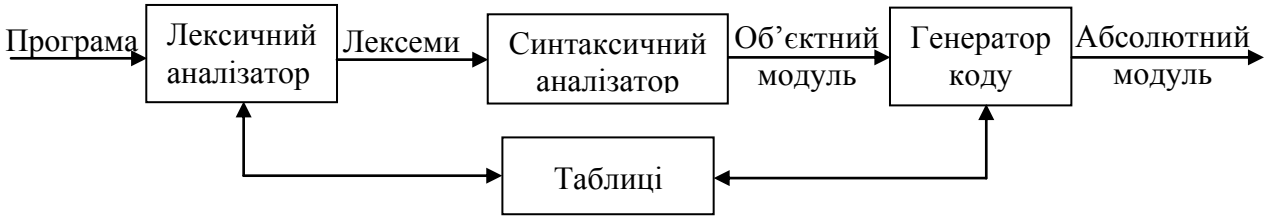


Рис. 2 Графічна модель компілятора

Лексичний аналізатор – програма мовою високого рівня. На цьому етапі послідовність символів вихідного файлу перетворюється на послідовність лексем. Зауважимо, що кожний символ має свій код. Таблиця кодів для IBM-сумісних ПК називається ASCII (American Standard code International Interchange). До набору символів Turbo і Borland Pascal відноситься майже вся таблиця ASCII (крім кирилиці). Серед символів програми на мові високого рівня лексичний аналізатор розрізняє символи-роздільники (наприклад, пробіл), і з їхньою допомогою виділяє лексеми.

Лексема – мінімально значима одиниця тексту програми, що має самостійний зміст.

Приклад. for(int i=1; i<=n; i++) cout<<i;

В даному фрагменті міститься 19 лексем: for, (, int, i, =, 1, ;, i, <, =, n, ;, i, ++,), cout, <<, i, ;.

Синтаксичний аналізатор – на основі правил граматики мови перевіряє коректність запису лексем і речень програми Також виконує і семантичний аналіз з метою встановлення семантики (сенсу) – наприклад, прив'язка ідентифікаторів до їх декларацій, типів, перевірка сумісності, ви-

значення типів виразів і т. д. В результаті перетворює послідовність лексем у послідовність внутрішніх кодів компілятора (об'єктний модуль).

Генератор коду (редактор зв'язків) – здійснює переклад внутрішнього коду компілятора в машинний код комп'ютера.

У таблицях міститься як постійна для трансляції програм інформація (наприклад, таблиця зарезервованих слів), так і індивідуальна інформація для якої-небудь програми (наприклад, таблиця ідентифікаторів).

У конкретних реалізаціях компіляторів ці етапи можуть бути розділені або, навпаки, об'єднані в тому чи іншому вигляді.

§ 1.5. Інтегроване середовище програмування.

Сучасна мова програмування – це не тільки мова програмування з компілятором. Мови програмування як програмний продукт поставляються на ринок програмного забезпечення разом з повним комплектом інструментів для створення програм. Такий комплект називається *інтегрованим середовищем програмування*. Він призначений не тільки для полегшення процесу складання програм, але і для професійної розробки програмних додатків.

Середовище програмування – це програма, що має засоби автоматизації процесів створення, підготовки та виконання програм.

До складу інтегрованого середовища програмування входять, як правило, наступні інструменти:

- текстовий редактор для набору і редагування програми;
- мова програмування з компілятором чи інтерпретатором;
- компонувальник;
- система усунення синтаксичних помилок;
- бібліотека готових до використання програмних модулів (процедур та функцій);
- довідкова система з питань розробки програм у даному середовищі.

Прикладами інтегрованих середовищ програмування є CodeBlocks, Visual Studio, Turbo Pascal, Delphi, C++ Builder та інші. Використання цих середовищ при розробці програмних додатків дозволяє користувачам при-

кладати набагато менше зусиль, ніж при написанні програм за допомогою мов більш низького рівня.

Створення програми починається зі складання алгоритму, орієнтованого на деяке інтегроване середовище програмування. Алгоритм перекладається на мову програми і вводиться з клавіатури у вікні текстового редактора. Після того, набрано текст з клавіатури і виправлені в ньому помилки, потрібно відправити програму на автоматичну перевірку. При цьому автоматично буде запущений ще один засіб зі складу середовища програмування – це налагоджувач (debugger). Налагоджувач перевірить текст із погляду синтаксису та семантики, запропонує вам виправити знайдені помилки. Програміст за допомогою налагоджувача може також переглянути і змінити вміст комірок пам’яті комп’ютера (покроковий налагоджувач).

Налагоджену програму можна запускати на виконання. Перекладом тексту програми на машинну мову займеться відомий уже вам засіб – транслятор. Йому буде допомагати компонувальник (linker), задачею якого є пошук і компонування розрізнених модулів і бібліотек, що необхідні для виконання програми.

Інтегроване середовище програмування забезпечує діалогову взаємодію з користувачем на всіх етапах складання і виконання програми. Середовище програмування сконструйоване таким чином, щоб користувачу були доступні всі можливі інструменти програмування, і він почував себе комфортно.

§ 1.6. Процес програмування.

Процес програмування не вичерпується лише самим написанням програми. Кожну програму потрібно:

- відлагодити (виявити та виправити в ній синтаксичні та логічні помилки);
- відтранслювати (перевести в машинні коди);
- протестувати (виконати програму декілька разів, вводячи при цьому такі набори вхідних даних, які дозволяють отримати різні варіанти відповідей);
- зберегти програму у вигляді файлу для її подальшого використання.

Для автоматизації усіх цих етапів розробки програми використовують *системи програмування*.

У склад системи програмування входить:

- екранний (текстовий) редактор;
- засоби налагодження та трансляції програми в машинні коди;
- засоби запуску програм на виконання;
- засоби запису програм на диски;
- ряд інших засобів для реалізації різних сервісних функцій.

§ 1.7. Історія виникнення та особливості мови С++

Мова програмування С була розроблена у 1972 році Денісом Рітчі у Bell Telephone Laboratories з метою написання нею операційної системи UNIX. Але, незважаючи на те, що мова С була розроблена для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення. На сьогодні вона є, безперечно, однією з найпопулярніших у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, які її знають. Версії компіляторів для мови С створено для багатьох операційних систем та апаратних архітектур. Мова С здійснила великий вплив на інші мови програмування, особливо на об’єктно-орієнтовану мову про-

грамування С++, яка спочатку проектувалася, як розширення для С, а також на Java та C#, які запозичили у мови С синтаксис.

Мова програмування С – це мінімалістична мова програмування. Серед її головних цілей: можливість прямолінійної реалізації компіляції, використовуючи відносно простий компілятор, забезпечення низькорівневого доступу до оперативної пам’яті і досить невелика динамічна підтримка. У результаті, код С придатний для більшості системного програмного забезпечення, яке традиційно писалося на асемблері. Що стосується граматики і синтаксису, то С є структурною мовою програмування.

С++ – універсальна мова програмування високого рівня з підтримкою декількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. При її створенні розробники прагнули зберегти сумісність з мовою С. Більшість програм на С працюватимуть і з компілятором С++. С++ має синтаксис, заснований на синтаксисі С. Мова програмування С++ була створена на початку 1980-х років, її творець – співробітник тієї ж фірми Bell Laboratories – Бьйорн Страуструп. Вона розроблялась як надбудова над С, використовуючи всі можливості останньої і додавши можливість роботи з класами та об’єктами. У 1983 році відбулося перейменування мови з «Сі з класами» в «мову програмування С++».

З 90-х років С++ стає однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для створення різного роду розважальних програм. Вона займала лідируючі позиції протягом більш ніж десятиліття.

У наступні два десятиліття мову С++ було збагачено й стандартизовано. Нащадками мови С++ є мови Java (Джава) та C# (Сі-шарп), спеціалізовані для програмування в сучасних комп'ютерних мережах. Ці мови за структурою схожі на С++, тому, володіючи С++, неважко перейти на Java або C#.

Тема 2. Основні поняття мови програмування С++.

§ 2.1. Склад мов програмування

У тексті на будь-якій природній мові можна виділити чотири основні елементи:

- символи,
- слова,
- словосполучення
- речення.

Алгоритмічна мова також містить такі елементи, тільки слова називають лексемами (елементарними конструкціями), словосполучення – виразами, речення – операторами.

Лексеми утворюються з символів, вирази з лексем і символів, оператори з символів виразів і лексем.

Таким чином, елементами алгоритмічної мови є:

- алфавіт мови,
- лексеми,
- вирази,
- оператори.

Програми складаються з синтаксичних конструкцій, які називаються *командами* (інші назви – оператори, вказівники, речення). Команди будуються з *лексем* – неподільних елементів мови: слів, чисел, символів, операцій.

§ 2.2. Алфавіт мови програмування С++

Алфавіт мови – це той набір символів (знаків), що є допустимим у даній мові.

Алфавіт мови С ++ включає

- великі та малі літери латинського алфавіту: 'A', ..., 'Z', 'a', ..., 'z';
- цифри 0, 1, ..., 9;

- спеціальні символи: " ( ) ' [ ] { } < > . , ; : ? ! * + − = / \ | # % $ & ~ @ та символ підкреслення _;
- пробільні символи (пробіл, символ табуляції, символи переходу на новий рядок);
- інші символи можна використовувати лише в коментарях до тексту програми.

§ 2.3. Лексеми: класифікація.

Текст програми представляє собою послідовність рядків, які складаються з символів, що входять до алфавіту мови. Рядки програми завершуються спеціальними символами. Максимальна довжина рядка становить приблизно 2048 байтів. Символи з алфавіту мови використовуються для побудови базових елементів програм – лексем. Нагадаємо, що лексема це мінімальна неподільна одиниця мови, що має самостійний зміст і при компіляції сприймається як єдине ціле.

Розглядають наступні класи лексем:

- Спеціальні символи
- Зарезервовані (ключові, службові) слова (зарезервовані ідентифікатори)
- Ідентифікатори (імена)
  - Стандартні (визначені) ідентифікатори
    - Ідентифікатори директив
    - Ідентифікатори стандартних функцій
  - Ідентифікатори користувача
- Знаки операцій
- Константи (літерали)
  - Числа
    - Десяткові
    - Вісімкові
    - Шістнадцяткові
  - Символи

- Рядки

- Символи-роздільники (дужки, крапка, кома, пробільні символи)

Межі лексем визначаються іншими лексемами, такими, як роздільники або знаки операцій.

Далі детально охарактеризуємо кожну групу лексем.

До *спеціальних символів* відноситься 22 наступних: + - * / < = > ( ) { } [ ] . , : ; '^ @ # $. Також використовуються комбінації спецсимволів: <= >= := .. (* *) (. .).

§ 2.4. Ключові слова.

Зарезервовані (ключові, службові) слова – це зарезервовані ідентифікатори, які складають фіксований словник мови програмування. Зміст і спосіб їх використання строго визначений в описі мови. Такі слова не можна використовувати в якості імен програмних об'єктів (змінних, процедур, функцій та ін.).

Зверніть увагу, що при написанні ключових слів великі та маленькі літери розрізняються і це треба враховувати при написанні програм, оскільки від регістру символів повністю змінюється їх зміст.

До зарезервованих слів відносяться наступні:

alignas*	alignof*	and	and_eq	asm
auto(1)	bitand	bitor	bool	break
case	catch	char	char16_t*	char32_t*
class	compl	const	constexpr*	const_cast
continue	decltype*	default(1)	delete(1)	do
double	dynamic_cast	else	enum	explicit
export	extern	false	float	for
friend	goto	if	inline	int
long	mutable	namespace	new	noexcept*
not	not_eq	nullptr*	operator	or
or_eq	private	protected	public	register

reinterpret_cast	return	short	signed	sizeof
static	static_assert*	static_cast	struct	switch
template	this	thread_local*	throw	true
try	typedef	typeid	typename	union
unsigned	using(1)	virtual	void	volatile
wchar_t	while	xor	xor_eq	

* - починаючи з С++11

(1) – значення в С++11 змінилося відносно початкового

Ключові слова поділяють на наступні групи:

- специфікатори типів: char, double, enum, float, int, long, short struct, signed, union, unsigned, void, typedef;
- кваліфікатори типів: const, volatile;
- кваліфікатори класів пам'яті: auto, extern, register, static;
- оператори мови та ідентифікатори спеціального призначення: break, continue, do, for, goto, if, return, switch, while; default, cas else,sizeof;
- модифікатори і псевдозмінні: privat, protected.

§ 2.5. Ідентифікатори та імена користувача.

***Ідентифікатор (ім'я)** – послідовність латинських букв та цифр, що починається з букви.*

Нагадаємо, що знак підкреслювання також вважається буквою.

Складаючи програму, користувач описує різні об'єкти і надає їм імена на свій розсуд. Тут простежується аналогія з математикою та фізикою, де різні величини позначають різними буквами, наприклад: *a*, *b*, *c* – довжини сторін трикутника, *h* – висота, *s* – шлях чи площа. В алгоритмічних мовах дане, яке міститиме в собі інформацію про висоту, можна назвати різними способами: *h*, *high*, *vysota* чи ще інакше. Придумуючи імена, треба дотримуватися певних правил.

Правила утворення імен користувача (*користувацьких ідентифікаторів*):

- ім’я може складатися лише з латинських літер, цифр та символу підкреслення – «_» (риска знизу);
- цифра не може бути першим символом в імені;
- літери можуть бути малими або великими;
- може бути будь-якої довжини (обмежується лише доступною пам’яттю, але насправді такі імена сенсу не мають);
- пропуски в іменах не допускаються;
- два різні об’єкта не можна позначати одним і тим же іменем в одній програмі.

В іменах великі і малі букви розрізняються: імена *A* та *a* (або *MyName* та *my_name*) – це різні імена з точки зору мови програмування С++.

Довжина ідентифікатора за стандартом не обмежена, але деякі компілятори і компонувальники накладають на неї обмеження. Ідентифікатор створюється на етапі оголошення змінної, функції, типу тощо. Після цього його можна використовувати в подальших операторах програми. При виборі ідентифікатора необхідно мати на увазі наступне:

- ідентифікатор не повинен збігатися з ключовими словами та іменами використовуваних стандартних об'єктів мови;
- не рекомендується починати ідентифікатори з символу підкреслення, оскільки вони можуть збігтися з іменами системних функцій або змінних, і, крім того, це знижує мобільність програми;
- на ідентифікатори, використовувані для визначення зовнішніх змінних, накладаються обмеження компонувальника.

На відміну від математики та фізики в інформатиці використовуються довгі імена, наприклад, *high, myName, MyNumber, my_program* тощо, які мають логічний смисл і дають розуміння призначення даного ідентифікатору.

Приклад. Наступні користувацькі ідентифікатори утворені правильно: *a, b, c, x, z, a1, a2, ... , a100, alpha, cat, my_number*. А такі імена утворені неправильно: *10a, 11b, a+2, a?*, оскільки при їх записі не дотримано правил, що наведені вище.

Для додаткового читання.

Щоб текст програми був більш зрозумілим, рекомендується дотримуватися загальноприйнятих угод про імена об'єктів.

- ім'я змінної пишеться малими літерами, наприклад index;
- з великої літери починаються імена функцій, класів, типів – Index;
- повністю великими літерами позначаються константи – INDEX;
- якщо ім’я складається з декількох слів, як, наприклад, birth_date, то прийнято або розділяти слова символом підкреслювання (birth_date), або писати кожне наступне слово з великої літери (birthDate). Який з двох способів краще – це справа власних уподобань програміста;
- не бажано використовувати скорочення або акроніми, особливо, якщо вони не є загальновідомими, оскільки це призводить до труднощів у розумінні програми;
- імена функцій та методів бажано задавати дієсловами;
- для створення ідентифікаторів слід використовувати англійську мову.

Це цікаво. Існує угода про правила створення імен, названа угорською нотацією (оскільки запропонував її співробітник компанії Microsoft угорець за національністю), за якою кожне слово, з якого складається ідентифікатор, починається з великої літери, а спочатку ставиться префікс, який відповідає типу величини, наприклад, iMaxLength, ipfnSetFirstDialog. Така нотація стала внутрішнім стандартом Microsoft. Префікси зарані обговорюються і є загальними для компанії.

Проте на сьогодні специфікація мови програмування С++ не рекомендує програмістам застосовувати таку нотацію.

§ 2.6. Стандартні ідентифікатори.

При створенні програм доводиться використовувати ряд стандартних функцій, які є загальновизнаними, як то виведення інформації на екран, відкриття файлів, застосування математичних функцій, назви бібліотек, тощо. Саме тому у мові програмування завжди є набір стандартних ідентифікаторів, які мають своє призначення і широко використовуються за умови підключення відповідних бібліотек. Як правило, стандартні імена

описані у бібліотеках, які входять до стандарту мови. Стандарт періодично оновлюється і до мови додаються нові стандартні ідентифікатори.

До *стандартних* (*визначених*) *ідентифікаторів* відносяться:

- імена вбудованих у мову функцій (наприклад, *cin, cout, sin, sqrt*),
- імена директив (наприклад, *#include, #define*);
- імена класів, що входять до стандарту мови, тощо.

Використання таких ідентифікаторів в якості імені змінної допускається, однак у цьому випадку їхня стандартна дія для даної програми буде втрачена. Проте деякі з них можна перевизначати, тобто дещо уточнювати, або змінювати їх призначення.

§ 2.7. Знаки операцій.

Знак операції – це один або більше символів, що визначають дію над операндами. Усередині знака операції пропуски не допускаються. Операції поділяються на унарні, бінарні та тернарну за кількістю операндів, що беруть участь у операції. Знаки операцій розглянемо далі.

Один і той же знак операції може інтерпретуватися по-різному в залежності від контексту. Всі знаки операцій за винятком [], (), ? і : є окремими лексемами.

Більшість стандартних операцій можна перевизначити (перевантажити), тобто визначити для них нове розуміння або взагалі змінити їх дію.

§ 2.8. Константи (літерали) в С++.

Будь-які значення, які використовуються у програмах – це або значення змінних, або константи (літерали). Принципова відмінність літерала – для нього не виділяється окрема пам'ять. Літерал є частиною коду програми, тому в процесі виконання програми його значення змінитись не може.

Константа у тексті програми може бути лексемою одного з двох видів: або це деяке фіксоване значення, або ідентифікатор, оголошений як константа. Частіше *літералом* називають фіксоване значення, а ідентифікатор, значення якого не можна змінити, *константою*. Нагадаємо, що

константу, описану ідентифікатором прийнято оголошувати повністю великими літерами.

Коли в програмі зустрічається певне число, наприклад 15, то це число називається літералом, або літеральної константою. Константою, тому що ми не можемо змінити його значення, і літералом, тому що його значення фігурує в тексті програми. Літерал є безадресною величиною: хоча реально він, звичайно, зберігається в пам'яті машини, немає ніякого способу дізнатися його адресу. Кожен літерал має певний тип, який визначається компілятором за замовчуванням, або можна задать його тип явно за потребою.

Компілятор виділяє літерал, як лексему і відносить її до тієї чи іншої групи, а потім всередині групи до певного типу по її формі запису в тексті програми і по числовому значенню.

Константа (літерал) *– це лексема, що представляє зображення фіксованого числового, рядкового або символьного значення.*

Якщо значення деякої величини (даного) не змінюватиметься протягом виконання всієї програми, то таке дане варто задати як постійну (константу).

Константи бувають не типізовані (літеральні) і типізовані, тобто під час оголошення ідентифікатора константи можна вказати її тип.

Літерали діляться на 6 груп:

- цілі;
- дійсні (з фіксованою та плаваючою крапкою);
- логічні;
- символьні;
- рядкові;
- типізовані.

§ 2.8.1 Цілі літерали.

Цілі константи можуть бути десятковими, вісімковими і шістнадцятковими.

Десяткові константи визначаються як послідовність десяткових цифр, що починається не з 0, якщо це не саме число 0 (приклади: 8, 0, 192345).

Вісімкові константи – це константи, які завжди починаються з 0. За 0 слідують вісімкові цифри (приклади: 01, 016 – десяткове значення 14,).

Шістнадцяткові константи – послідовність шістнадцяткових цифр, яким передують символи 0x або 0X (приклади: 0xA, 0X00F).

Залежно від значення цілого літерала компілятор по-різному представить його в пам'яті комп'ютера (тобто компілятор сам визначить тип даних цього літерала).

Наприклад, одне й те ж саме число 20 у різних системах числення має такий вигляд

Десяткова	20
Вісімкова	024
Шістнадцяткова	0x14

За замовчуванням всі цілі літерали мають тип signed int. Можна явно визначити цілий літерал як має тип long long, приписавши в кінці числа букву L (можна використовувати як прописну літеру L, так і рядкову – l, однак для зручності читання не слід вживати малу: її легко переплутати з цифрою 1).

Буква U (або u) поставлена після числа визначає літерал як unsigned int, а дві букви – UL або LU – як тип unsigned long. наприклад:

128u 1024UL 1L 8Lu

§ 2.8.2 Дійсні літерали.

Дійсні константи мають іншу форму внутрішнього представлення в пам'яті комп'ютера. Компілятор розпізнає такі константи за їхнім виглядом. Дійсні константи можуть мати дві форми подання: з фіксованою точкою і з плаваючою точкою.

Літерали з фіксованою точкою мають такий вигляд:

[Цифри].[цифри]

Наприклад, 5.7, 0, .0001, 41.

Як видно з прикладів, така форма запису допускає відсутність або цілої частини, або дробової, але не двох одразу.

Вид константи з плаваючою точкою (експоненціальна або показникова форма):

[цифри] [цифри]{E | e} [+ | -] [цифри]

або вигляд

mEn

Наприклад, 0.5e5, .11e-5, 5E3.

У цій формі запису числа можна виділити такі основні характеристики:

- знак числа (додатнє чи від'ємне),
- *m* – мантиса числа – для нормалізованої форми $m \in [1;10)$,
- *e* – експонента,
- знак порядку,
- *n* – порядок числа (виражає степінь основи числа, на яке множиться мантиса).

Зазначені характеристики дійсного числа зберігаються у пам'яті комп'ютера. Число у показниковій формі може бути представлено, наприклад, так: 1.0123E-10. Мантиса записується ліворуч від знака експоненти(***E чи e***), порядок – праворуч. Символ **E(e)** означає основу степеня – 10, і компілятор розпізнає цей запис як форму представлення дійсного числа. Символ пропуску всередині числа не допускається, а для відділення цілої частини від дробової використовується не кома, а крапка. При додатних значеннях числа і мантиси знак «+» можна не вказувати.

Експоненціальна форма дійсного числа використовується для запису дуже великих або дуже малих чисел, для яких задавати зайві нулі не зовсім зручно, наприклад:

$$1.0123*10^{20}=1.0123e20=10123000000000000000,$$

$$1.0123*10^{-10}=1.0123E-10=0.0000010123.$$

Експоненціальна форма запису також допускає відсутність цілої або дробової частини.

За замовчуванням всі дійсні константи мають тип **double** – подвійну точність, що найчастіше займає в пам’яті 64 біти, тобто 8 байтів. Але у випадку, якщо програміста не влаштовує тип за замовчуванням, його можна вказати явно за допомогою спеціальних літер. Так, додавши літеру **f чи F**, константі надають дійсний тип **float** з одинарною точністю, наприклад, 8.5f. Якщо в представленні константи використовується літера **L чи l**, то вона має тип **long double**.

Зображення від’ємної цілої чи дійсної константи вважається константним виразом, що складається зі знака унарної операції зміни знака (-) та константи.

Наприклад, -273, -2730.e-1, -273L.

§ 2.8.3 Логічні літерали.

Логічні константи представлені двома словами, що позначають логічні значення – *true* (істина) і *false* (неправда).

§ 2.8.4 Символьні константи.

Символьні константи – це один або два символи, заключені в одинарні прямі лапки.

Наприклад, 's', '2', '#', 'c', ' ' (пробіл).

Символьні константи, що складаються з одного символу, мають тип *char* і займають в пам'яті один байт, символьні константи, що складаються з двох символів, мають тип *int* і займають два байти. Символьні константи мають цілий тип і їх можна використовувати як цілочислові операнди у виразах.

Всі символьні константи та їх відповідні числові значення (коди) представлені у стандартній таблиці ASCII (таблиця кодування). Причому числові значення для зручності представлені у десятковому, вісімковому та шістнадцятковому кодах.

Окремої уваги заслуговують послідовності, що починаються зі зворотної риски \ (back-slash, бек-слеш, обернений слеш). Вони називаються керуючими або ***escape***-послідовностями, вони використовуються:

- Для представлення символів, які не мають графічного відображення, наприклад:

  \a – звуковий сигнал,

  \v – вертикальна табуляція,

  \n – перехід на новий рядок,

  \t – горизонтальна табуляція.
- Для представлення (екранування) символів \ , ' , ? , ", тобто \\, \', \? ,\".
- Для представлення будь-яких символів за допомогою шістнадцяткових (\0x2b – '+') або вісімкових (\053 – '+') кодів.

escape -послідовності – це комбінації символів, що представляють розділові (CR, Tab та ін.) і нетрадиційні символи.

Керуюча послідовність інтерпретується як одиночний символ. Але в останніх стандартах мови програмування С++ керуючу послідовність потрібно заключать у подвійні прямі лапки для їх коректної роботи. Якщо безпосередньо за косою рискою знаходиться символ, який не передбачено стандартними escape-комбінаціями, то результат інтерпретації непередбачуваний, але частіше всього бек-слеш буде просто проігнорований.

Порожня символьна константа недопустима.

Це цікаво! Символьний літерал може містити префікс L (наприклад, L'a'), що означає спеціальний тип ***wchar_t*** – двобайтний символьний тип, який застосовується для зберігання символів національних алфавітів, якщо вони не можуть бути представлені звичайним типом *char*, такі як, наприклад, китайські або японські літери.

§ 2.8.5 Рядкові константи.

Рядкова константа – це послідовність символів, яка заключається в подвійні прямі лапки. У кінець кожного рядкового літералу компілятором автоматично додається нуль-символ, який представляється керуючою послідовністю "\0". Саме тому навіть порожній рядок "" має довжину 1 байт.

Існує суттєва різниця між символом 'ф' і рядком "ф", оскільки символ має 1 байт, а рядок 2 байта.

Приклад рядка: "Студенти – веселий народ."

Усередині рядків також можуть використовуватися керуючі символи.

Наприклад, рядок "\nМова програмування" – напис «Мова програмування» на екрані виводиться з нового рядка.

Якщо всередині рядкового літерала потрібно записать подвійну пряму лапку, то перед нею ставиться бек-слеш, за яким компілятор відрізняє її від подвійної лапки, яка обмежує (закінчує) рядок.

Наприклад, рядок "Ми вивчаємо \"Мову програмування С++\"." на екрані має вигляд

Ми вивчаємо "Мову програмування С++".

Рядки, які записані у програмі підряд при компіляції конкатенуються («склеюються»). Тобто послідовність двох рядків

"Літерали – це фіксовані значення."
"Вони застосовуються в програмах."

На екрані виглядають наступним чином:

Літерали – це фіксовані значення. Вони застосовуються в програмах.

Довгу рядкову константу можна розмістити в декількох рядках, використовуючи в якості знака переносу бек-слеш. При виведенні ці символи ігноруються компілятором, а наступний рядок сприймається як продовження попереднього.

Наприклад, рядок
```
"Ми розглядаємо\
типи та види літералів\
у мові програмування"
```
є повністю еквівалентним рядку

«Ми розглядаємо типи та види літералів у мові програмування».

Зауваження. У С++ існують стандартні константи, описані у різних бібліотеках. Наприклад, математичні константи: число π позначається M_PI, π/2 – M_PI_2, ln2 – M_LN2 тощо. Їх можна безпосередньо використовувати в програмі, попередньо підключивши бібліотеку math.h або cmath.

Додатково. Максимальна допустима довжина рядкового літерала в Microsoft C++ приблизно 2048 байтів. Однак якщо рядковий літерал складається з двох частин, взятих в подвійні лапки, препроцесор об'єднує ці частини в один рядок, і для кожного об'єднаного рядка додає додатковий байт до загальної кількості байтів.

Наприклад, припустимо, що рядок складається з 40 рядків з 50 символами в кожному рядку (2000 символів) і одного рядка з 7 символами і що кожен рядок заключений в подвійні лапки. Це означає, що додається до 2007 байтів плюс один байт для завершального нуль-символу, тобто всього 2008 байтів. В об'єднанні додатковий символ додається для кожного з перших 40 рядків. В результаті ми отримуємо 2048 байтів. Зверніть увагу, що якщо замість подвійних лапок використовуються продовження рядків (\), препроцесор не додає додатковий символ для кожного рядка. І хоча окремі рядки в лапках не можуть бути довшими 2048 байтів, об'єднавши рядки, можна створити рядковий літерал, що складається приблизно з 65535 байтів.

§ 2.8.6 Типізовані константи.

Типізовані константи використовуються як змінні, значення яких не може бути змінено після ініціалізації. Під час виконання програми значення констант змінювати не можна. Типізована константа оголошується за допомогою ключового слова const, за яким слід вказати тип константи, але, на відміну від змінних, константи завжди повинні бути ініціалізовані, тобто їм має бути присвоєне конкретне значення.

Зауваження. Створення та використання іменованих і типізованих констант буде розглянуто у посібнику далі.

Тема 3. Структура програми на С++. Типи даних.

§ 3.1. Структура програми на С++.

Перш ніж перейти до написання програм необхідно ознайомитись зі структурою програми на мові програмування С++. Структура програми – це розмітка робочого коду з метою точного визначення меж основних блоків програми та синтаксису. Структура програми дещо відрізняється в залежності від обраного середовища програмування. В цьому посібнику ми орієнтуємося на вільне програмне забезпечення типу CodeBlocks.

Загалом, програма мовою С++ складається з

- директив препроцесора,
- функцій,
- описів,
- коментарів.

Одна з функцій обов'язково повинна мати ім'я ***main*** і саме з неї починається виконання програми.

Програма на мові С++ має наступну структуру:

```
1.  //program.cpp – файл, який містить код програми
2.  #include <назва бібліотечного файлу1>
3.  #include <назва бібліотечного файлу N>
4.  <інші директиви препроцесора>
5.  <описи>
6.  <оголошення глобальних змінних>
7.  <оголошення глобальних констант>;
8.  <оголошення і створення функцій користувача>;
9.  int main()
10. {
11. <оператори, які складають тіло програми (функції)>;
12. }
```

Спершу перерахуємо елементи цієї структури.

- У першому рядку міститься коментар, який починається символом //.
- З другого по четвертий рядки після символу # розташовуються директиви препроцесора.
- У 9 рядку – заголовок головної функції.
- Фігурні дужки {} у 10 та 12 рядках означають початок та кінець функції *main*. Оператори у тілі функції розділяються символом **;** (крапка з комою).
- З 5 по 8 рядки можуть міститися описи та оголошення, які не є обов'язковими.

Далі розглянемо основні елементи, які можуть бути у коді програми С++.

§ 3.2. Коментарі.

Коментар – це фрагмент тексту програми, який служить для пояснення призначення програми або окремих команд і не впливає на виконання команд.

Коментарі потрібні для кращого розуміння програми як для самого розробника, так і для сторонніх людей, яким потрібно вникати в суть написаної програми. При створенні великих проектів групою розробників коментарі набувають особливого значення, оскільки документований код спрощує роботу колективу в цілому.

У мові С++ є два види коментарів – *однорядкові* та *багаторядкові*.

Їх записують так:

```
// однорядковий коментар
/* багаторядковий коментар*/
```

У першому випадку коментар повинен бути або в кінці рядка, або єдиним у рядку. Він є однорядковим і діє тільки до кінця рядка. Отже, коментарі такого виду можуть знаходитися лише праворуч від оператора. Вкладення однорядкових коментарів один в один не має сенсу, оскільки лівий обмежувач вкладеного коментарю при цьому нічим не відрізняється від інших вже закоментованих символів.

Другий спосіб більш універсальний: багаторядковий коментар можна записувати будь-де. Компілятор ігнорує все, що міститься між символами /* і */, включаючи самі ці символи, і отже, між ними можна вставляти скільки завгодно рядків тексту. Вкладення багаторядкових коментарів не використовується, оскільки перший символ */ у вкладеному коментарі завершить коментар взагалі і це проведе до помилки компіляції, оскільки подальший текст коментаря буде сприйнято компілятором як фрагмент основного коду програми.

Як правило, однорядкові коментарі пояснюють зміст окремого рядка програми. Багаторядкові коментарі можуть містити умову задачі, пояснення призначення та параметрів функцій або класів, тощо. Крім того, коментарі дозволяють локалізувати помилки при налагодженні програми адже, закоментувавши підозрілий фрагмент програми, можна досить швидко знайти місце можливої помилки. Таким чином, багаторядкові коментарі доцільно застосовувати для тимчасового виключення блоків при налагодженні програми.

Наведемо кілька порад стосовно раціонального застосування коментарів:

- коментарі можна записувати будь-якою мовою, враховуючи аудиторію для якої вони призначені;
- коментарі бажано записувати правильними та логічними реченнями, за правилами мови коментаря і використовувати правильну пунктуацію
- розміщувати в коментарях тільки потрібну для супроводу інформацію;
- пропуск рядка – один з найбільш ефективних коментарів, що значно поліпшує розуміння програми, видимо розбиваючи її на фрагменти;
- штрихові лінії коментаря або порожні рядки застосовуються для поділу функцій та інших логічно завершених фрагментів програм.

§ 3.3. Директиви препроцесора.

Препроцесор – це програма, яка обробляє директиви на першому етапі компіляції. Вона виконує макропідстановку, умовну компіляцію і включення іменованих файлів. Кожна з цих операцій кодується у виді особливого оператора (*директиви*), що починається символом #.

Директиви препроцесора – це команди компілятора відповідної мови програмування, які виконуються до початку компіляції програми.

Препроцесор не виконує синтаксичний аналіз тексту. Він просто розпізнає макропідстановки і виконує їх. Директиви препроцесора не залежать від синтаксису мови, за одним виключенням – їх імена чуттєві до регістра букв.

У мові С++ визначено такі директиви

#define	#elif	#else	#endif	#error	#if
#ifdef	#ifndef	#include	#line	#pragma	#undef

Кожна директива повинна розташовуватися в окремому рядку програми. При цьому необхідно уважно стежити за пробілами всередині директив. Деякі з них не мають значення, а інші приводять до помилок. Як правило, в кінці рядка з директивою не ставиться ніяких символів-роздільників.

Директиви препроцесора можна розташовувати в довільному місці програми.

Далі розглянемо деякі з цих директив.

§ 3.3.1. Директива #include.

Директивою ***#include*** до тексту програми включаються заголовочні або інші файли. Фактично необхідно приєднати програмний код із файлу, зазначеного після цієї директиви.

Заголовочні файли мають розширення **.h** Їх також називають файлами заголовків (*header* -файлами, бібліотеками, модулями). У таких файлах зазвичай оголошують константи і змінні, заголовки (сигнатури) функцій тощо. У файлах заголовків визначені всі стандартні команди і функції мови С++.

Ім’я файлу може бути вказане двома способами:

#include <fileName.h>

#include "OwnFile.h"

Тобто після директиви #include потрібно в кутових лапках <…> записати назву файлу та вказати його розширення. Кутові лапки означають, що заданий файл є стандартним і компілятор шукає його у визначених для даного середовища програмування місцях, спеціально призначених для зберігання таких файлів. В переважній більшості середовищ всі стандартні бібліотеки розміщені в папці INCLUDE. Тоді стандартний файл підключається таким чином:

наприклад, #include <math.h>

або, більш нові формати, #include <cstdio>

Подвійні лапки означають, що заголовочний файл – користувацький, тобто створений програмістом і його пошук компілятором починається з того каталогу, в якому розташований проект (початковий код програми).

Якщо потрібний файл розміщений не в папці проекту, то зазначається повний шлях до файлу, починаючи з диску і т.д. Наприклад, якщо деякий файл MyLib.h є в папці ***stud*** на диску **d**:, то потрібно писати так: **#include** "d:***stud***\MyLib.h".

Згідно з останніми стандартами ISO/ANSI файли заголовків в директиві **#include** прийнято записувати без розширення, наприклад: **#include** <stdlib>, але в такому випадку потрібно вказувати *простір імен*, в якому розв’язується задача. Файли заголовків мови С, які використовуються в С++-програмах, починаються з літери ***c***, наприклад # **include** <cmath>. Ця можливість реалізована в усіх сучасних компіляторах мови.

Вживання директиви **#include** не підключає відповідну стандартну бібліотеку, а тільки дозволяє вставити в текст програми опису із зазначеного заголовка. Підключення кодів бібліотеки здійснюється на етапі компонування, тобто після компіляції. Хоча в стандартних заголовочних файлах містяться всі описи стандартних функцій, в код програми включаються тільки ті функції, які використовуються в програмі.

Заголовочний файл сам по собі може містити директиви #include. Вони називаються *вкладеними директивами* #include. Тому іноді важко зрозуміти, які ж конкретно заголовочні файли включені в даний текст, і, таким чином, деякі заголовочні файли можуть виявитися включеними декілька разів. Запобігти цьому дозволяють *умовні директиви препроцесора*. Розглянемо приклад:

```
#ifndef MYTEXT_H
#define MYTEXT_H
    /*вміст файлу mytext.h*/
#endif
```

Умовна директива #ifndef перевіряє, чи не було значення MYTEXT_H визначено раніше. MYTEXT_H – це константа препроцесора. Такі константи прийнято записувати великими буквами (про це вже говорилося раніше). Препроцесор опрацьовує наступні рядки аж до директиви #endif. В іншому випадку він пропускає рядки від #ifndef до #endif.

Директива #define MYTEXT_H
визначає константу препроцесора MYTEXT_H. Розмістивши цю директиву безпосередньо після директиви #ifndef, можна гарантувати, що змістовна частина заголовочного файлу mytext.h буде включена у текст програми тільки один раз, скільки б разів не включався в текст сам цей файл.

§ 3.3.2. Директива #define.

***Директива* #define** слугує для заміни констант, ключових слів, операторів або виразів, які часто застосовуються у програмі деякими ідентифікаторами. Тобто ця директива має подвійне значення:

- визначення констант;
- визначення макросів.

Ідентифікатори, які замінюють текстові або числові значення (константи), називають *іменованими константами*. Ідентифікатори, що замінюють фрагменти програм, називають макровизначеннями, причому макровизначення можуть мати аргументи.

Директивою #**define** можна встановити постійне значення (оголосити іменовану константу). Наприклад, якщо в програмі є рядок #**define** N 25, то *N* під час виконання програми буде мати значення 25, тобто всі входження змінної *N* буде замінено на 25. У програмі змінювати це значення не можна, тобто, якщо вже *N* оголошено, як іменовану константу, то операції типу *N*++ або *N**=7 викличуть помилку компіляції.

Крім того директива #**define** дає можливість описати макроси (макровизначення) – короткі команди (перевизначити команди) або записати функції.

Наприклад,

#**define** D(a, b, c) b*b-4*a*c.

Тепер скрізь для обчислення дискримінанту замість команди d = b*b-4*a*c можна записувати d = D(a, b, c), тобто D(a, b, c) в даній програмі є макросом – скороченим записом команди обчислення виразу.

При створенні макросу з параметрами однією з поширених помилок початківців, є наявність пробілу між іменем макросу і відкритою дужкою при описуванні директиви **#define**. Це неправильно, тобто відкрита дужка повинна розташовуватись впритул до імені макросу, інакше при виконанні програми виникне помилка компіляції, оскільки в такому випадку макрос вважається непараметризованим, а параметри є частиною подальшого виразу.

Також необхідно враховувати, що макрос не є функцією і в якості параметрів йому не можна передавати вирази.

Наприклад, якщо задано макровизначення для знаходження квадрату числа такого виду

#define Mult(x) x*x

то коректним викликом макросу у програмі є, наприклад, такий Mult(c), де c – деяке число (вже відоме на момент виклику). А виклик Mult(c+5) приведе до помилкового результату, оскільки буде виконана підстановка c+5*c+5, що за правилами порядку дії приводить до результату 6*c+5 і є неправильним для нашого макросу.

Щоб запобігти таким проблемам при написанні макросів з параметрами в усіх місцях використання параметрів їх потрібно заключать в круглі дужки, а для підсилення коректності і весь вираз також взяти у круглі дужки. Таким чином більш коректним варіантом наведеного вище прикладу буде наступний: **#define** Mult(x) ((x)*(x))

§ 3.3.3. Інші директиви.

Тема директив препроцесора у мові програмування С++ є досить обширною і глибокою. Кожна з перерахованих на початку параграфу директив має своє призначення і особливості застосування і це виходить за рамки даного посібника. Таким чином, про інші директиви з прикладами застосування можна ознайомитись додатково у літературі [1]

§ 3.4. Код програми

Початкова програма, яка підготовлена на С++ у вигляді текстового файлу (*початковий код програми*), проходить 3 етапи обробки:

1) препроцесорне перетворення тексту;
2) компіляція;
3) компонування (редагування зв’язків або збірок).

Після цих трьох етапів формується *виконуваний код програми*. Завдання препроцесора – перетворення тексту програми до її компіляції. Правила препроцесорної обробки визначає програміст за допомогою директив препроцесора.

Як вже було зазначено раніше, суттєвою особливістю мови С++ є те, що програми складаються з функцій, які виконують роль підпрограм в інших мовах. Головна функція, яка повинна бути в кожній програмі, – це функція виду

int **main()**
{
    тіло функції (можна з командою **return** 0;)
}

де ***int*** **main()** – ***заголовок функції.***

Тіло функції – це послідовність оголошень, визначень, описів і виконуваних операторів, заключених у фігурні дужки {}. Кожне оголошення, визначення, опис або оператор закінчується символом **;** (крапка з комою).

Визначення – задають об'єкти (об'єкт – це іменована область пам'яті; окремий випадок об'єкта – змінна), необхідні для представлення в програмі оброблюваних даних. Напиклад,

```
const int y=10;          //іменована константа
float x;                 //змінна.
```

Описи – повідомляють компілятор про властивості і імена об'єктів і функцій, описаних в інших частинах програми.

Оператори – визначають дії програми на кожному кроці її виконання.

§ 3.5. Типи даних

Навколишній світ можна відобразити в програмі у вигляді конкретних даних. Дані мають деяку величину, тобто *величина* – одиниця даних, якими оперує програма.

Величина характеризується

- назвою (ім’я, ідентифікатор);
- типом (залежить від того, що саме ця величина описує і не змінюється протягом виконання програми);
- значенням, яке може змінюватись за час виконання програми безліч разів.

Мета програми полягає в обробці різноманітних даних. Дані можуть бути *сталими* та *змінними*. Константи (літерали) – це сталі значення, які описані у попередній темі. Як правило, літерали вже мають конкретне значення і за їх виглядом *автоматично* (за замовчуванням) визначається їх тип. Змінні величини обов’язково мають ім’я (ідентифікатор), яке не дає жодного уявлення про їх тип. Тому кожен ідентифікатор у програмі мовою С++ повинен мати асоційований з ним тип даних. Дані різних типів зберігаються і обробляються по-різному. Всі змінні, які фігурують у програмі, ретельно класифікують за типами.

Тип даних визначає:

- обсяг оперативної пам'яті, який резервується для зберігання значень зазначеного типу;
- множину значень, які можуть приймати величини цього типу;
- операції і функції, які можна застосовувати до даних цього типу.

Зауваження: Обсяг пам’яті може залежати від різновиду операційної системи комп'ютера.

Залежно від вимог завдання програміст вибирає тип для об'єктів програми.

Типи С++ можна розділити на *прості* (вбудовані) і *складені* (похідні).

До простих типів відносять типи, які характеризуються одним значенням. В С++ визначено 7 простих типів даних:

- ***int*** (цілий);
- ***char*** (символьний);
- ***wchar_t*** (розширений символьний);
- ***bool*** (логічний);
- ***float*** (дійсний);
- ***double*** (дійсний з подвійною точністю);
- ***void*** (порожній тип, який не має значення).

На основі цих типів вводиться опис складених типів (масивів, функцій, структур, класів, вказівників, тощо). Оскільки похідні типи будуються на основі простих, то їх може бути велика кількість, тому для початку розглянемо прості вбудовані типи.

Для уточнення внутрішнього подання та діапазону значень стандартних типів мова С++ використовує чотири *специфікатори типу*:

- ***short*** (короткий);
- ***long*** (довгий);
- ***signed*** (знаковий);
- ***unsigned*** (беззнаковий).

Всі величини повинні бути описані до першого їх використання.

§ 3.6. Опис змінних

Для зберігання різних даних використовуються змінні. Поняття змінних відоме зі шкільного курсу математики. У програмуванні принципи досить схожі. Величини, які під час виконання програми можуть мати різні значення, називають *змінними*.

Змінна (ідентифікатор користувача) – це іменована область пам'яті, у якій зберігаються дані визначеного типу. Змінна має ім'я, розмір та інші атрибути, такі як область видимості, час існування, тощо. Ім'я змінної служить для звертання до області пам'яті, у якій зберігається її значення. Перед використанням будь-яка змінна повинна бути описана, при цьому для неї резервується деяка область пам'яті, розмір якої залежить від конкретного типу змінної. Під час виконання програми змінна може приймати різні значення.

Ім'я змінній дає програміст під час написання програми. Правила створення імен змінних розглядалися у *§ 2.5.*

Кожне ім'я змінної у програмі – це ідентифікатор користувача і для того, щоб його розпізнавав компілятор, змінна повинна бути описана. Для опису вказується її тип та ім'я.

Загальний вигляд опису змінних такий:

тип_змінної ім'я_змінної;

або

тип <список імен змінних, перерахованих через кому>;

де ***тип*** – це коректний тип даних мови С++,

ім'я_змінної – надає програміст на власний розсуд у відповідності з рекомендаціями, наведеними у *§ 2.5.*

Наприклад, змінні оголошують так:

int *a, dob_numbers*;

float *minimum, x1, suma*;

char *simv*;

При оголошенні змінних для них виділяється ячейка пам'яті, розмір якої залежить від обраного типу даних. Але в цій ячейці не має ніякого значення, тобто в пам'яті в якості значення цієї змінної знаходиться так

зване «сміття». Автоматично ніяке значення змінній не надається. Тому бажано присвоїти оголошеній змінній значення, тобто *ініціалізувати* її. Якщо деяка змінна ініціалізується, то в списку вона вказується у вигляді:

<ім’я>=<константа>;

або

<ім’я>=<константний_вираз>;

або

<ім’я>=<вираз>;

або

<ім’я>=<інше_ім’я>;

де <ім’я> – ім’я деякої вже описаної змінної;

= – оператор присвоєння;

<константа> – визначає конкретне встановлене значення;

<константний_вираз> – вираз, який складається тільки з констант та знаків операцій, наприклад, 20*3+8;

<вираз> – вираз, який складається з констант, змінних та знаків операцій, наприклад, 25+*x*-2**b*.

```
int a=12;
double x1, dobutok=12*a;
char simb=12;
```

§ 3.7. Цілі типи даних

Назва типу	Назва типу на мові С++	Обсяг, байтів	Діапазон допустимих значень
цілі числа	***int***	2 або 4	-32768 ... 32767 або -2147483648 ... 2147483647
короткі цілі числа	***short int***	2	-32768 ... 32767
короткі цілі числа без знаку	***unsigned short int***	2 або 4	0 ... 65535 або 0 ... 4294967295
довгі цілі числа	***long int***	4	-2147483648 ...

			2147483647
довгі цілі числа без знаку	***unsigned long int***	4	0 ... 4294967295

Приклад. Оголосимо три змінні цілого типу:

int *x*, *b*;

short int *z*;

На етапі компіляції для змінних *x, b, z* буде надано певний обсяг оперативної пам'яті. Дати значення цим змінним можна на етапі виконання програми за допомогою команд присвоєння, наприклад, так:

$x = 157; b = -68; z = 15;$

У ділянку пам'яті, виділену для змінної *x*, буде записано число 157, для *b* – -68, а для *z* – 15.

Під час виконання програми значення змінних можна змінювати. Наприклад, команда присвоєння $x = 2003$ занесе до відповідної для змінної *x* ділянку пам'яті число 2003 (попереднє значення 157 видаляється автоматично).

§ 3.8. Дійсні типи даних

Назва типу	**Назва типу на мові С++**	**Обсяг, байтів**	**Діапазон допустимих значень**
дійсні числа одинарної точності	***float***	4	$\pm 3{,}410^{-38}$ ... $\pm 3{,}410^{38}$; 0
дійсні числа подвійної точності	***double***	8	$\pm 1{,}710^{-308}$ ... $\pm 1{,}710^{308}$; 0
розширення дійсного числа подвійної точності	***long double***	10	$\pm 1{,}1810^{-4932}$ ...$\pm 1{,}1810^{4932}$; 0

Зауваження. У десяткових числах ціла і дробова частини числа відокремлюються крапкою. Внутрішнє представлення дійсного числа складається з 2 частин: мантиси і порядку. У IBM-сумісних ПК величини типу ***float*** займають 4 байта, з яких один розряд відводиться під знак мантиси, 8 розрядів під порядок і 24 - під мантису. Величини типу ***double*** займають

8 байтів, під порядок і мантису відводяться 11 і 52 розряду відповідно. Довжина мантиси визначає точність числа, а довжина близько його діапазону.

Приклад. Розглянемо фрагмент програми

float *h*, *pi*=3.1415926;
double *v*=365.976;
const float *w*=12, *s*=23.4;

Дійсні числа можна записати у форматі з *фіксованою точкою*, наприклад −2.3, 5.0041, або в *науковому форматі* (в форматі с *плаваючою точкою*), наприклад, −0.2e+2 (це є число –20), 3.27e−3 (це є 0.00327).

Запис $\pm me\pm n$ означає множення числа m на 10 в степені $\pm n$, тобто за визначенням $me\pm n=m*10^{\pm n}$, m – це мантисса, а n – це порядок.

Знак «+» можна не ставити, знак «−» писати обов'язково.

§ 3.9. Символьний тип даних (char)

Символьний тип – це множина символів кодової таблиці ASCII (Американський стандартний код для міжнародного обміну). Символьна константа – це один символ, взятий в одинарні лапки, або число в 8-, 10- або 16-ковій системі числення, яке є кодом символу в таблиці ASCII. Кожному символу ставиться у відповідність число, яке називається кодом символу. Під величину символьного типу відводиться 1 байт. Символи з кодами від 0 до 31 відносяться до службових і мають самостійне значення тільки в операторах введення-виведення. Величини типу ***char*** також застосовуються для зберігання чисел з діапазонів від -128 до 127 (***signed char***) або від 0 до 255 (***unsigned char***).

Приклад. Розглянемо опис символьних змінних, де змінним *m*1, *m*2, *m*3 і *m*4 дамо значення латинської літери А чотирма способами:

char *m*l='A',
char *m*2=0101,
char *m*3=65,
char *m*4=0x41;

Число 65 - це десятковий код символу 'A', 101 – вісімковий, 41 – шістнадцятковий. На початку останніх двох кодів (101, 41) записують префікси "0" або "0x" відповідно.

Приклад. Розглянемо спосіб визначення десяткового ASCII-коду деякого символу, наприклад 'A':

char *c* = 'A';

int *n* = *c*;

Змінній *n* буде присвоєно значення 65.

Ще в С++ існує тип ***wchar_t***, який призначений для роботи з набором символів, для кодування яких недостатньо 1 байта, наприклад символів таблиці *Unicode*. Розмір цього типу, як правило, відповідає типу ***short***. Рядкові константи такого типу записуються з префіксом *L*: *L*"String #1".

§ 3.10. Логічний тип даних (bool)

Логічний тип характеризується двома значеннями даних: *false* (неправда) і *true* (істина). Наприклад,

bool *b=true*.

Змінні цього типу займають 1 байт в пам'яті комп'ютера. У С++ значення змінних типу ***int*** можна асоціювати з логічними значеннями: нулю відповідає значення *false*, всім іншим числам, відмінним від нуля – *true*.

Зауважимо, що не всі компілятори підтримують тип даних ***bool***. Тому, перед тим як його використовувати, варто з'ясувати можливості компілятора.

§ 3.11. Тип void

Тип ***void*** застосовують до функцій, які *не повертають значення в точку виклику* або до функцій без параметрів. Множина значень цього типу – порожня. Детальніше цей тип будемо розглядати в подальшому.

§ 3.12. Команда присвоєння

Команда *присвоєння* має такий загальний вигляд:

<назва змінної>=<вираз>

або

<ім’я змінної *1*>=<ім’я змінної *2*>=...=< ім’я змінної *N*>=<вираз>;

Дія команди. Обчислюється вираз і його значення записується в комірку відповідної змінної або декількох змінних одночасно. Вираз призначений для опису формул, за якими будуть виконуватися обчислення. Вираз може містити числа, літерали (константи), змінні, назви функцій, з'єднані символами операцій.

Змінна і вираз не обов'язково повинні бути одного типу. Крім того, в виразі можуть бути дані різних числових типів (змішані вирази). Якщо тип змінної не збігається з типом виразу, то в С ++ відбувається автоматичне перетворення (узгодження, перетворення) типів. Про коректність такого узгодження ми поговоримо в кінці лекції.

§ 3.13. Типи користувача

Крім вищеописаних стандартних типів даних, можна створювати ***типи користувача***

typedef <опис типу> <назва нового типу>;

Приклад. Опишемо тип *kilkist* для позначення коротких цілих даних без знаку:

typedef *unsigned short int* *kilkist*;

Змінні (*kil*1, *kil*2) цього типу в програмі можна оголосити так:

kilkist *kil*1, *kil*2;

§ 3.14. Змінні

Дані, значення яких необхідно ввести з клавіатури або які під час виконання програми можуть мати різні значення, називають змінними. Їх оголошують так:

<тип змінних 1> <список змінних 1>;

...

<тип змінних *N*> < список змінних *N*>;

Елементи списків записують через кому. Наприклад, змінні оголошують так:

int *a, c*;

float *b, d, z*;

char *w*;

Змінна в С++ – іменована область пам'яті, в якій зберігаються дані певного типу. У змінної є тип, ім'я та значення. Тип визначає розмір комірки пам'яті, виділеної для цієї змінної, ім'я служить для звернення до цієї області пам'яті, в якій зберігається значення. Перед використанням будь-яка змінна повинна бути описана.

Приклади:

int *a*;

float *x*;

Змінним можна надавати початкове значення відразу під час оголошення. Це називається *ініціалізацію* даних.

Наприклад,

float *b*, *d*=2.5, *a*=4;

char *w*='t';

Отже, в загальному випадку змінні одного типу можна оголошувати так:

<тип змінних> <ім’я змінної 1>=<значення 1>, ..., <ім’я змінної *N*>=<значення *N*>, <список інших змінних>;

§ 3.15. Перетворення типів

В С ++ існує *явне* і *неявне* перетворення типів.

У загальному випадку неявне перетворення типів зводиться до участі в виразі змінних різного типу (так звана арифметика змішаних типів). Якщо подібна операція здійснюється над змінними базових типів, розглянутих вище, вона може спричинити за собою помилки: у разі, наприклад, якщо результат займає в пам'яті більше місця, ніж відведено під приймаючу змінну, неминуча втрата значущих розрядів.

Для явного перетворення змінної одного типу в інший перед ім'ям змінної в дужках вказується присвоєний їй новий тип:

p=(***float***) *k*;

Тема 4. Операції. Стандартні функції. Потоки введення-виведення

§ 4.1. Операції в С++.

Для здійснення маніпуляцій з даними С++ має в своєму розпорядженні широкий набір операцій. Операції представляють собою певну дію над операндом (операндами), результатом якої є значення, що повертається.

Операнд це об’єкт (об’єкти), над яким виконується операція.

Знаки операцій забезпечують формування виразів. Вирази складаються з операндів, знаків операцій та дужок. Кожний операнд є, в свою чергу, виразом або окремим випадком виразу – сталою або змінною.

В залежності від кількості операндів операції поділяються на

- унарні (дія виконується над одним операндом);
- бінарні (у операції приймає участь два операнда);
- тернарні (у операції приймає участь три операнда).

Таблиця 4.1. Операції в С++

Знак	**Зміст операції**	**Приклад застосування**
Унарні операції (дія з одним операндом, знак операції, як правило, ставиться ПЕРЕД операндом)		
&	операція визначення адреси операнда (адреси зображуються 16-річними числами)	**&***a*
*****	вказівник (pointer) – засіб визначення адреси (звернення за адресою, розіменування). Надає можливість оперувати не з іменами змінних, а безпосередньо звертатися до областей пам'яті комп'ютера.	**int** **r, *nom;* **float*** *ptrA;*
−	унарний мінус, змінює знак арифметичного операнда.	*−k*
~	порозрядне інвертування внутрішнього двійково-	~10011010 =

	го коду цілочисельного операнда (побітове заперечення)	01100101
!	логічне заперечення (НЕ). В якості логічних значень використовується 0 − неправда і не 0 − істина, запереченням 0 буде 1, запереченням будь-якого ненульового числа буде 0.	**!**18 = 0 **!**0 = 1
++	операція інкременту (збільшення на одиницю)	**int** *a*=2, *b*;
	префіксна операція – збільшує операнд ДО його застосування, постфіксна операція – збільшує операнд ПІСЛЯ його застосування.	*b*=3*++*a*; *a*=3; *b*=3*3=9 (префіксна)
--	операція декремента (зменшення на одиницю)	**int** *f*=20, *g*;
	префіксна операція - зменшує операнд ДО його застосування, постфіксна операція зменшує операнд ПІСЛЯ його застосування.	*g*=(*f*−−)−10; *g*=20−10=10; *f*=19 (постфіксна)
Бінарні операції (дія з двома операндами, знак операції ставиться МІЖ операндами)		
Адитивні		
+	бінарний плюс (додавання арифметичних операндів)	3+5=8
−	бінарний мінус (віднімання арифметичних операндів)	7−12=−5
Мультиплікативні		
*	множення операндів арифметичного типу	2*(−5)+4=−6
/	ділення операндів арифметичного типу (якщо операнди цілочисельні, то виконується цілочисельне ділення)	12/4−2=1 18/(6−1)=3 (але не 3.6)
%	отримання остачі від ділення цілочислових операндів	7%3=2 4%13=4
Операції зсуву (визначені тільки для цілочислових операндів).		

Основне призначення цих операторів – швидкі обчислення, так як їх підтримка здійснюється на апаратному рівні (процесор), то алгоритми, виконані з використанням даних операторів, виконуються найбільш високопродуктивно.		
<<	зсув вліво бітового представлення значення лівого цілочисельного операнда на кількість розрядів, що дорівнює значенню правого операнда (звільнені розряди обнуляються). *Примітка.* Таким чином, наприклад, виконується швидке піднесення до степеня числа 2. 1<<6=64, т. я. $1_{10}=1_2$, то якщо 1 зсунути на 6 розрядів вліво, то отримаємо 1000000, тоді $1000000_2=2^6=64_{10}$ (таким чином $1<<6=2^6$)	6<<2=24, т. я. $6_{10}=110_2$, то якщо 110 зсунути на 2 розряди вліво, то маємо 11000, тоді $11000_2=24_{10}$
>>	зсув вправо бітового представлення значення правого цілочисельного операнда на кількість розрядів, що дорівнює значенню правого операнда (звільнені зліва розряди обнуляються, а праві розряди зникають, якщо операнд без-знакового типу) і заповнюються знаковим розрядом, якщо знакового.	10>>1=2, т. я. $10_{10}=1010_2$, то 1010 на 1 розряд вправо $=0101_2=5_{10}$
Порозрядні операції		
&	порозрядна кон'юнкція (І) бітових представлень значень цілочисельних операндів (біт =1, якщо відповідні біти обох операндів=1, в інших випадках = 0).	6&5=4, $6_{10}=110_2$, $5_{10}=101_2$, 1&1=1, 1&0=0, 0&1=0, тобто $100_2=4_{10}$
|	порозрядна диз'юнкція (АБО) бітових представлень значень цілочисельних операндів (біт =1, якщо відповідний біт одного з операндів=1, якщо обидва =0, то і біт =0).	6|5=7, $6_{10}=110_2$, $5_{10}=101_2$, 1|1=1, 1|0=1, 0|1=1, тобто $111_2=7_{10}$
^	порозрядне виключне АБО бітових представлень значень цілочисельних операндів(біт =1, якщо відповідний біт тільки одного з операн-	6^5=3, $6_{10}=110_2$, $5_{10}=101_2$, 1^1=0, 1^0=1, 0^1=1, тоб-

	дів=1)	то $11_2=3_{10}$
Операції порівняння: результатом є *true*(не 0) або *false*(0)		
<	менше, ніж	7<4=*false*
>	більше, ніж	5>1=*true*
<=	менше чи дорівнює	6<=6=*true*
>=	більше чи дорівнює	
==	дорівнює	7==2=*false*
!=	не дорівнює	5!=6=*true*
Логічні бінарні операції		
&&	кон'юнкція (І) цілочисельних операндів або відношень, істинна лише тоді, коли обидва операнди істинні.	1&&1=1; 1&&0=0; 0&&1=0; 0&&0=0;
||	диз'юнкція (АБО) цілочисельних операндів або відношень, є хибною лише тоді, коли обидва операнди помилкові.	1||1=1; 1||0=1; 0||1=1; 0||0=0;
Операції з присвоєнням		
	=, +=, −=, *=, /=, %=і т.д.	a+=10 ↔ a=a+10

§ 4.2. Вирази в С++.

З символів, змінних, роздільників і знаків операцій можна конструювати ***вираз***. Кожен вираз являє собою *правило обчислення нового значення*. Будь-який вираз, за яким йде крапка з комою утворює *пропозицію* або *інструкцію мови*.

Якщо вираз повертає ціле або дійсне число, то він називається арифметичним. Пара арифметичних виразів, об'єднана операцією порівняння, називається ***відношенням***. Якщо відношення має ненульове значення, то воно *істинне*, інакше – *помилкове*.

Пріоритети операцій у виразах наводяться у наступній таблиці.

Пріоритет	**Операції**	**Зміст операції**
1	:: ( ) [ ] . –>	дужки
2	! ~ – ++ -- & * ^	унарні
3	* / %	мультиплікативні бінарні
4	+ –	адитивні бінарні
5	<< >>	порозрядного зсуву
6	< > <= >=	відношення
7	== !=	відношення
8	&	порозрядна кон’юнкція «І»
9	^	порозрядне виключне «АБО»
10	|	порозрядна диз’юнкція«АБО»
11	&&	логічна кон’юнкція «І»
12	||	логічна диз’юнкція «АБО»
13	? :	умовна операція
14	= *= /= %= -= &= ^= |= <<= >>=	операція з присвоєнням
15	,	оператор кома

§ 4.3. Вирази в С++.

Всі стандартні математичні функції в С++ описані в бібліотеці *cmath* (або, більш старіша, *math.h*). Тому, якщо в програмі використовуються якісь математичні функції, то на початку такої програми потрібно записати рядок підключення заголовкового файлу

#include <cmath>

або, #include <math.h>

Примітка. Бібіліотека *cmath* є більш сучасною, працює стабільніше і містить більш розширений функціонал.

Основні математичні функції бібліотеки *cmath* наведені в наступній таблиці. Більш розширену характеристику бібліотеки *cmath* можна переглянути в додатках.

Назва функції	Математичний запис
***abs*(*x*)**	$\|x\|$ (цілі числа)
***cos*(*x*)**	*cos x*
***sin*(*x*)**	*sin x*
***tan*(*x*)**	*tg x*
***log*(*x*)**	*ln x*
***pow*(*x*,*y*)**	x^y
***sqrt*(*x*)**	$\sqrt{x}$
***exp*(*x*)**	e^x
***pow*10(*x*)**	10^x
***log*10(*x*)**	*lg x*
***int* (*x*)**	відкидає дробову частину числа, але результат дійсний
***fabs*(*x*)**	$\|x\|$ (дійсні числа)
***acos*(*x*)**	*arccos x*
***asin*(*x*)**	*arcsin x*
***atan*(*x*)**	*arctg x*
***ceil*(*x*)**	округлює число *x* до більшого цілого
***floor*(*x*)**	
***floor*(*x*)**	округлює число *x* до меншого цілого
***fmod*(*x*,*b*)**	

Всі наведені функції, крім **abs**(*x*) і **pow10**(*x*), мають тип аргумента і результат ***double*.** Для функції **abs**(*x*) і **pow10**(*x*) типом аргумента и результатом є ***int*.**

Приклад. Нехай оголошено змінні

int *x* = –2, *x1*, *a* = 3;

float *pi* = 3.1415926, *m* = 16, *kut*, *k*;

Тоді в результаті виконання наведених команд змінним *x1*, *a*, *kut*, *k*, *m* будуть присвоєні наступні значення:

x1=**abs**(*x*)	*x1*=**abs**(–2)=|-2|=2;
a=**pow10**(*a*)	*a*=**pow10**(3)=10^3=1000;
kut=**cos**(2**pi*)	*kut*=**cos**(2*π)=1;
k=**pow**(*m*,1./4)	*k*=**pow**(16,1./4)=$16^{1/4}=\sqrt[4]{16}$ =2;
m=**sqrt**(*m*)	*m*=**sqrt**(16)= $\sqrt{16}$ =4

Приклад. Нехай у програмі оголошені змінні

double *b=7.6, b1, b2*

Тоді після виконання команд

*b1=**ceil**(b)*	*b1=**ceil**(*7.6*)*=8
*b2=**floor**(b)*	*b2=**floor**(*7.6*)*=7

Всі інші математичні функції можна виразити через основні. Наприклад, $ctg\ x = 1/tg\ x$, $\log_b a = \ln a / \ln b$ тощо.

Послідовність виконання операцій у виразах така ж, як у математиці, і визначається пріоритетом виконання операцій.

Операції одного рівня виконуються послідовно зліва направо. Для зміни порядку виконання операцій використовують круглі дужки. Спочатку обчислюються вирази в дужках – в першу чергу у внутрішніх, потім – у зовнішніх. Кількість відкритих та закритих дужок у виразі має бути однаковим.

Всі елементи виразів (дроби, показник степеня, індекси) записують в горизонтальні рядки. У багатьох випадках їх беруть у квадратні дужки. Вирази можна записувати в декількох рядках. "Розривати" вирази можна, наприклад, після символу арифметичної операції, але власне символ дублювати не потрібно.

§ 4.4. Потоки. Введення й виведення даних

На відміну від класичного стандарту мови, у мові С++ немає вбудованих засобів введення і виведення – він здійснюється з допомогою функцій, типів і об'єктів, які знаходяться в стандартних бібліотеках. Для організації введення-виведення тут реалізована концепція потоків, яка визначена в спеціальних модулях. У модулі ***istream.h*** описані *команди введення*, в модулі ***ostream.h*** – *команди виведення*, а в модулі ***iostream.h*** – команди введення і виведення. Тобто, при використанні бібліотеки класів С++, використовується бібліотечний файл *iostream.h*, в якому визначені стандартні потоки введення даних з клавіатури **cin** і виведення даних на екран монітора **cout**, а також відповідні операції

<< – операція запису даних у потік;

>> – операція читання даних з потоку.

Наприклад:

```
#include <iostream.h>;
. . . . . .
cout << "\n Введіть кількість елементів: ";
cin >> n;
```

Можливе використання традиційних функцій мови С – ***printf*** і ***scanf***.

Під потоком розуміють процес вводу-виводу інформації в файл. Периферійні пристрої вводу-виводу, такі як клавіатура, монітор, принтер, розглядаються як текстові файли. Під час виконання будь-якої програми підключаються стандартні потоки для введення даних з клавіатури (**cin**), виведення на екран (**cout**), виведення повідомлення про помилки (**cerr**) і допоміжний потоці (**clog**).

Стандартні потоки використовують операції введення (>>) і виводу (<<) даних. За замовчуванням стандартним пристроєм для потоків виводу даних і повідомлень про помилки є монітор, а для потоку вводу даних – клавіатура. Однак можна перенаправляти потоки, наприклад, можна зчитувати вхідну інформацію для програми не з клавіатури, а з деякого текстового файлу на диску.

§ 4.4.1. Команда введення даних

Задавати значення змінних можна двома способами: за допомогою *команди присвоювання*, наприклад x=3.1, або команди введення даних з клавіатури. Команда введення даних з клавіатури дає можливість виконувати програму для різних вхідних даних, що робить її більш універсальною (масовою). Команда введення має такий загальний вигляд: **cin** >> <змінна>;

Дія команди. Виконання програми зупиняється. Система переходить у режим очікування введення даного (екран темний, блимає курсор). Користувач набирає на клавіатурі значення змінної і натискає на клавішу вводу. В результаті виконання цієї команди, змінної буде присвоєно конкретне значення (те, що користувач введе з клавіатури).

Якщо необхідно ввести значення відразу для декількох змінних, можна використовувати кілька потоків вводу, або записати всі змінні в одному потоці **cin**, застосувавши для цього кілька команд ">>", а саме:

cin >> < зміна 1> >> < зміна 2> >> ... >> < змінна N>;

Значення змінних можна вводити через пробіл або після введення кожного даного натискати клавішу Enter.

Якщо в списку введення (який набрали на клавіатурі через пропуск) даних більше, ніж змінних, то зайві дані будуть зчитані наступною командою введення. Якщо такої команди в програмі немає, вони будуть проігноровані.

Програми необхідно складати так, щоб ними могли користуватися не лише автори, а й інші особи. Тобто програми повинні бути масовими і зрозумілими. Перед командою введення даних варто записувати команди виводу на екран текстової підказки – повідомлення про те, що саме слід ввести.

\

§ 4.4.2 Команда виведення даних

Для виводу на екран повідомлень і результатів обчислень використовують стандартний потік виводу **cout** і операцію <<. Команда виведення має такий загальний вигляд:

cout << <змінна>; або для виведення декількох змінних

cout << <вираз 1> << <вираз 2> << ... << <вираз N>;

У списку виводу можуть бути константи, змінні або виразу. Елементи списку в потоці ***cout*** відокремлюють операціями <<.

Дія команди. Константи, значення змінних і виразів, що виводяться на екран у вікно виведення. Це вікно можна переглянути за допомогою команди *Window → User screen* головного вікна компілятора або комбінації клавіш *Alt+F5*.

Текстові повідомлення в команді виведення записують в лапках. Лапки на екран виводитися не будуть.

Для того, щоб дані виводилися в потрібному для користувача вигляді і не зливалися не екрані, використовують керуючі послідовності.

§ 4.4.3 Керуючі послідовності

Керуючі послідовності (escape-послідовності) – це комбінації спеціальних символів, які використовуються для вводу і виводу даних. Керуюча послідовність складається з символу «зворотний» слеш "\" і наступного за ним символу. Вони призначені для форматного виведення результатів обчислень на екран, наприклад, для переходу на новий рядок, звукового сигналу, а також для виведення на екран деяких спеціальних символів: апостроф, лапки тощо. Основні керуючі послідовності наведені в таблиці.

Символи керуючих послідовностей	**Коментар**
\a, \7	Подати звуковий сигнал
\b	Повернути курсор на один символ назад (знищити попередній символ)
\f	Перейти на нову сторінку
\n	Перейти на новий рядок
\r	Повернути курсор на початок рядка

\t	Перевести курсор на наступну позицію табуляції
\v	Вертикальна табуляція
\\	Вивести символ похилої риски
\'	Вивести символ апострофа
\"	Вивести символ лапок
\?	Вивести знак питання

Керуючі послідовності разом з коментарями записують в лапках. Розглянемо дію цих послідовностей на прикладі. Якщо записати команди

cout << "Увага! \a\n ";

cout << "Дане \"a\" введено некоректно \n";

cout << "Виконати програму ще раз";

то буде поданий звуковий сигнал і на екрані побачимо

Увага!
Дане "a" введено некоректно
Виконати програму ще раз

Якщо використовувати послідовність, невизначену в мові С++ (наприклад, \z), то компілятор пропустить символ послідовності (зворотний слеш) і виведе на екран лише символ, записаний після риски (в нашому випадку літеру z).

Зауваження 1. Замість керуючої послідовності \n можна використовувати команду **endl** – кінець рядка. Наприклад, ці дві команди рівносильні:

cout << "f=" << *f* << "\n"; і ***cout*** << "f=" << *f* << *endl*;.

Зауваження 2. Як ви вже знаєте значення дійсних чисел (тип float або double) можна виводити на екран в стандартному або науковому форматах. Якщо значення даного необхідно округляти до п значущих цифр, то перед командою виведення потрібно записати **cout.precision(n).** Ця команда дасть можливість визначити кількість нових знаків після коми. Для подання результатів у науковому форматі необхідно під'єднати заголовковий файл **iomanip.h** і перед командою виведення записати

cout << setiosflags(ios::scientific);

Зауваження 3. Для задання розміру поля, в якому буде показана змінна можна використовувати команду **cout.width(n);** Виведене число буде притискатися до правого краю поля, тобто «зайві» знаки в полі будуть заповнені пробілами перед виведеним числом. Якщо вам потрібна така функція, то її потрібно викликати перед кожним використанням команди ***cout.***

Зауваження 4. При виведенні на **cout** або **cerr** програми можна вказати ширину виводу кожного числа, використовуючи модифікатор **setw** (установка ширини). З допомогою setw програми вказують мінімальну кількість символів, займане числом. Щоб використовувати модифікатор setw, ваша програма повинна включати заголовковий файл *iomanip.h*. Модифікатор setw(n) використовується в потоці виводу безпосередньо перед виведеним числом.

§ 4.5.Класи пам'яті

Загальний вигляд оператора опису змінної:

[клас пам'яті][const]тип ім'я [инициализатор];

[] – такі дужки означають, що даний параметр може бути в цьому місці, але він не обов'язковий.

Клас пам'яті може приймати значення: **auto, extern, static, register**. Клас пам'яті визначає час життя і область видимості змінної. Якщо клас пам'яті не вказаний явно, то компілятор визначає його виходячи з контексту оголошення. Час життя може бути постійним, – протягом виконання програми або тимчасовим – протягом блоку.

Область видимості – частина тексту програми, з якої припустимо звичайний доступ до змінної. Зазвичай область видимості збігається з областю дії. Крім того випадку, коли у внутрішньому блоці існує змінна з таким же ім'ям.

const – показує, що цю змінну не можна змінювати (іменована константа). При описі можна присвоїти початкове значення змінної (ініціалізація).

Класи пам’яті:

auto – автоматична локальна змінна. Специфікатор auto може бути заданий тільки при визначенні об'єктів блоку, наприклад, в тілі функції. Цим змінним пам'ять виділяється при вході в блок і звільняється при виході з нього. Поза блоку такі змінні не існують.

extern – глобальна змінна, вона знаходиться в іншому місці програми (в іншому файлі або далі по тексту). Використовується для створення змінних, які доступні у всіх файлах програми.

static – статична змінна, вона існує тільки в межах того файлу, де визначена змінна.

register – аналогічний *auto*, але пам'ять під такі змінні виділяється в регістрах процесора. Якщо такої можливості немає, то змінні обробляються як auto.

Приклад:

```
int a;                          //глобальна зміна
void main()
    {
        int b;                  //локальна змінна
        extern int x;           //зміна x визначена в іншому місці
        static int c;           //локальна статична змінна
        a=1;                    //ініціалізація глобальної змінної
        int a;                  //локальна змінна а
        a=2;                    //ініціалізація локальної змінної
        ::a=3;                  //ініціалізація глобальної змінної
    }
int x=4;                        // оголошення та ініціалізація x
```

У прикладі змінна *a* визначена поза всіх блоків. Областю дії змінної *a* є вся програма, крім тих рядків, де використовується локальна змінна *a*.

Змінні *b* і *c* – локальні, область їх видимості – блок. Час життя порізному: пам'ять під *b* виділяється при вході в блок (т. к. за замовчуванням клас пам'яті **auto**), звільняється при виході з нього. Змінна (***static***) існує, поки працює програма.

Якщо при визначенні початкове значення змінним не задається явно, то компілятор обнуляє глобальні та статичні змінні. Автоматичні змінні не ініціалізуються.

Ім'я змінної повинно бути унікальним в своїй області.

Опис змінної може бути виконано або як оголошення, або як визначення. Оголошення містить інформацію про клас пам'яті і тип змінної, визначення разом з цією інформацією дає вказівку виділити пам'ять. У прикладі **extern int x;** – оголошення, а решта – визначення.

Для додаткового читання

Функція стандартного виводу printf()

Функція **printf ()** - функція стандартного виводу. З допомогою цієї функції можна вивести на екран рядок символів, число, значення змінної.

Функція **printf()** має прототип у файлі *stdio.h*, тобто для використання цієї функції потрібно підключити зазначену бібліотеку директиви препроцесора.

Керуючий рядок містить два типи інформації: символи, які безпосередньо виводяться на екран, і специфікатори формату, що визначають, як виводити аргументи.

Функція **printf()** це функція форматованого виводу. Це означає, що в параметрах функції необхідно вказати формат даних, які будуть виводитися. Формат даних вказується специфікаторами формату. Специфікатор формату починається з символу % за яким слід код формату.

Специфікатори формату:

%c	символ
%d (%i)	ціле десяткове число
%e (%E)	десяткове число у вигляді x.xx e+xx
%f (%F)	десяткове число з плаваючой комою xx.xxxx
%o	восьмирічне число

%s	рядок символів
%u	беззнакове десяткове число
%x (%X)	шістнадцятирічне число
%%	символ %
%p (%n)	вказівник

Крім того, до команд формата можуть бути застосовані модифікатори *l* и *h*.

%ld	друк long int і long long
%hu	друк short unsigned
%Lf	друк long double

У специфікаторі формату, після символу % може бути вказана точність (кількість цифр після коми). Точність визначається наступним чином:

%m.n<код формату>.

де m – загальна кількість позицій виводу, n – число цифр після коми, а <код формату> – один з специфікаторів наведених вище.

Наприклад, є змінна x=10.35635 типу float і потрібно вивести її значення з точністю до 3-х цифр після коми, то потрібно написати:

printf("Змінна x = %.3f", x);

Результат: Змінна x = 10.356

І можна вказати мінімальну ширину поля відведеного для друку. Якщо рядок або число більше вказаної ширини поля, рядок або число друкується повністю.

Наприклад, якщо написати: printf("%d 5.0",20); то результат буде наступним: _ _ _ 20

Зверніть увагу на те, що число 20 надрукувалось не з самого початку рядка. Якщо потрібно щоб невикористані місця поля заповнювалися нулями, то потрібно поставити перед шириною поля символ 0.

Наприклад: printf("%05d",20);

Результат: 00020

Крім специфікаторов формату даних у керуючому рядку можуть перебувати керуючі символи:

\b	BS, забій
\f	Нова сторінка, перевід сторінки
\n	Новий рядок, перевід рядка
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\"	Подвійні лапки
\'	Апостроф
\\	Зворотна похила риска
\0	Нульовий символ, нульовий байт
\a	Сигнал
\N	Восьмирічна константа
\xN	Шістнадцятирічна константа
\?	Знак питання

Найчастіше використовується символ \n. За допомогою цього керуючого символу здійснюється перехід на новий рядок.

Розглянемо приклади програм.

Приклади програм.

```
/* Приклад 1 */
#include <stdio.h>
int main()
{
    int a, b, c;
    a=5; b=6; c=9;
    printf("a=%d, b=%d, c=%d", a, b, c);
}
```

Результат роботи програми:

a=5, b=6, c=9

```
/* Приклад 2 */
#include <stdio.h>
int main()
{
    float x, y, z;
    x=10.5; y=130.67; z=54;
    printf("Координати об’єкта: x: %.2f, y:%.2f, z:%.2f", x, y, z);
}
```

Результат роботи програми:

Координати об’єкта: x: 10.50, y:130.67, z:54.00

```
/* Приклад 3 */
#include <stdio.h>
int main()
{
    int x;
    x=5;
    printf("x=%d", x*2);
}
```

Результат роботи програми:

x=10

```
/* Приклад 4 */
#include <stdio.h>
int main()
{
    printf("\"Текст в лапках\"");
    printf("\n Вміст кисню: 100%");
}
```

Результат роботи програми:

"Текст в лапках"

Вміст кисню: 100%

```
/* Приклад 5 */
#include <stdio.h>
int main()
{
    int a;
    a=11;                    // 11 в десятковій дорівнює b в шіст-
надцятирічній
    printf("a-dec=%d, a-hex=%X", a, a);
}
```

Результат роботи програми:

a-dec=11, a-hex=b

```
/* Приклад 6 */
#include <stdio.h>
int main()
{
    char ch1, ch2, ch3;
    ch1='A';
    ch2='B';
    ch3='C';
    printf("%c%c%c", ch1, ch2, ch3);
}
```

Результат роботи програми:

ABC

```
/* Приклад 7 */
#include <stdio.h>
int main()
{
```

```
    char *str="Мій рядок.";
    printf("Это %s", str);
}
```

Результат роботи програми:

Мій рядок.

```
/* Приклад 8 */
#include <stdio.h>
int main()
{
    printf("Добрий день!\n"); // Після виводу – перехід на новий рядок
    printf("Мене звати Павло."); // Це буде виводитись на новому рядку
    }
```

Результат роботи програми:

Добрий день!
Мене звати Павло.

Функція стандартного вводу scanf()

Функція **scanf()** – функція форматованого вводу. З її допомогою можна вводити (зчитувати дані зі стандартного пристрою вводу. Введеними даними можуть бути цілі числа, числа з плаваючою комою, символи, рядки і покажчики.

Функція **scanf()** має прототип у файлі stdio.h, тобто його треба підключити.

Керуючий рядок містить три види символів: специфікатори формату, прогалини та інші символи. Специфікатори формату починаються з символу %.

Специфікатори формату:

%c	читання символа

%d (%i)	читання десяткового цілого
%e	читання числа типу float (плаваюча кома)
%h	читання short int
%o	читання восьмирічного числа
%s	читання рядка
%x	читання шістнадцятирічного числа
%p	Читання вказівника
%n	Читання вказівника в збільшеному форматі

При читанні рядка за допомогою функції **scanf()** (специфікатор формату %s), рядок зчитується до першого пробілу!! тобто якщо ви вводите рядок "Привіт, світ!" з використанням функції **scanf()**

char str[80**];** // масив на 80 символів

scanf("%s",str**);**

Після зчитування результуючий рядок, яка буде зберігатися в масиві str буде складатися з одного слова "Привіт". ФУНКЦІЯ ВВОДИТЬ РЯДОК ДО ПЕРШОГО ПРОБІЛУ! Якщо потрібно зчитувати рядки з пробілами, то використовується функція gets(char *str).

За допомогою функції **gets()** можна зчитувати повноцінні рядки. Функція gets() читає символи з клавіатури до появи символу нового рядка (\n). Сам символ нового рядка з'являється, коли натискається клавіша Enter.

Приклад програми, яка дозволяє ввести цілий рядок з клавіатури і вивести її на екран.

```
#include <stdio.h>
int main()
{
    char buffer[100];                    // масив (буфер) для рядка,
    що вводиться
    gets(buffer);                        // вводимо рядок и натискаємо
    Enter
    printf("%s", buffer);                // вивід введеного рядка на екран
}
```

Ще одне важливе зауваження! Для введення даних за допомогою функції **scanf()**, їй у якості параметрів потрібно передавати адреси змінних, а не самі змінні. Щоб отримати адресу змінної, потрібно поставити перед ім'ям змінної знак &(амперсанд). Знак & означає взяття адреси.

Якщо в програмі є змінна, то вона зберігає своє значення в пам'яті комп'ютера. Адреса, яку ми отримуємо за допомогою & – це адреса в пам'яті комп'ютера, де зберігається значення змінної.

Розгянемо приклад програми, який показує, як використовувати &:

```
#include <stdio.h>
int main()
{
    int x;
    printf("Введіть змінну x:");
    scanf("%d",&x);
    printf("Змінна x=%d", x);
}
```

Тепер повернемося до керуючої рядку функції scanf().

Символ пробілу в керуючому рядку дає команду пропустити один або більше прогалин у потоці вводу. Крім пробілу може сприйматися символ табуляції або новий рядок. Ненульовий символ вказує на читання і відкидання цього символу.

Роздільниками між двома введеними числами є символи пробілів, табуляції або новий рядок. Знак * після % і перед кодом формату (специфікатором формату) дає команду прочитати дані зазначеного типу, але не присвоювати значення.

Наприклад: **scanf**("%d%*c%d", &i, &j); при введенні 50+20 присвоїть змінній і значення 50, змінної j – значення 20, а символ " + " буде прочитаний і проігнорований.

У команді формату може бути вказана найбільша ширина поля, яка підлягає зчитуванню.

Наприклад: scanf("%5s", str); вказує необхідність прочитати з потоку вводу перші 5 символів. При введенні 1234567890ABC масив str буде містити тільки 12345, решта символи ігноруються. Роздільники: пробіл, символ табуляції, символ нового рядка при введенні символу сприймаються, як і всі інші символи.

Якщо в керуючому рядку зустрічаються якісь інші символи, то вони призначаються для того, щоб визначити і пропустити відповідний символ. Потік символів 10plus20 оператором **scanf**("%dplus%d", &x, &y); присвоїть змінній x значення 10, змінної y – значення 20, а символи plus пропустить, так як вони зустрілися в керуючій рядку.

Однією з потужних особливостей функції **scanf()** є можливість завдання множини пошуку (scanset). Множина пошуку визначає набір символів, з якими будуть порівнюватися символи scanf(), що читаються функцією. Функція scanf() читає символи до тих пір, поки вони зустрічаються у безлічі пошуку. Як тільки символ, який введений, не зустрівся в множині пошуку, функція scanf() переходить до наступного специфікатору формату. Множина пошуку визначається списком символів, укладених у дужки. Перед відкриваючою дужкою ставитися знак %. Розглянемо це на прикладі.#**include** <*stdio.h*>

```
int main()
{
    char str1[10], str2[10];
    scanf("%[0123456789]%s", str1, str2);
    printf("\n%s\n%s", str1, str2);
}
```

Введемо набір символів: 12345abcdefg456

На екрані програма видасть:

```
12345
abcdefg456
```

При заданні безлічі пошуку можна також використовувати символ "дефіс" для завдання проміжків, а також максимальну ширину поля вводу.

scanf("%10[A-Z1-5]", *str1*);

Можна також визначити символи, які не входять в безліч пошуку. Перед першим з цих символів ставитися знак ^. Безліч символів розрізняє малі та великі літери.

Нагадаємо, що при використанні функції scanf(), їй у якості параметрів потрібно передавати адреси змінних. Вище був написаний код:

char str[80]; // масив на 80 символів

scanf("%s", *str*);

Зверніть увагу на те, що перед str не стоїть символ &. Це зроблено тому, що str є масивом, а ім'я масиву – str є вказівник на перший елемент масиву. Тому знак & не ставиться. Ми вже передаємо функції scanf() адреса.

Приклади програм

Приклад 1. Ця програма виводить на екран запит "Скільки вам років?:" і чекає введення даних. Якщо, наприклад, ввести число 20, то програма виведе рядок "Вам 20 років.". При виклику функції scanf(), перед змінній " age " ми поставили знак &, так як функції scanf() потрібні адреси змінних. Функція scanf() запише введене значення за вказаною адресою. У нашому випадку введене значення 20 буде записано за адресою змінної age.

```
#include <stdio.h>
int main()
{
    int age;
    printf("\n Скільки вам років?:");
    scanf("%d",&age);
```

```
    printf("Вам %d років.", age);
}
```

Приклад 2.

Програма-калькулятор. Цей калькулятор може тільки додавати числа. При вводі 100+34 програма видасть результат: 100+34=134.

```
#include <stdio.h>
int main()
{
    int x, y;
    printf("\nКалькулятор:");
    scanf("%d+%d", &x, &y);
    printf("\n%d+%d=%d", x, y, x+y);
}
```

Приклад 3.

Цей приклад показує, як встановити ширину поля зчитування. У нашому прикладі ширина поля дорівнює п'яти символів. Якщо вводиться рядок з великою кількістю символів, то всі символи після 5-го будуть відкинуті. Зверніть увагу на виклик функції **scanf().** Знак & не стоїть перед ім'ям масиву *name* так як ім'я масиву *name* є адреса першого елемента масиву.

```
#include <stdio.h>
int main()
{
    char name[5];
    printf("\nВведіть ваш логін (не більше 5 символів):");
    scanf("%5s", name);
    printf("\nВи ввели %s", name);
}
```

Пример 4.

Приклад показує, як можна використовувати множину пошуку. Після запуска програми вводиться число от 2 до 5.

```
#include <stdio.h>
int main()
{
        char bal;
        printf("Ваша оцінка 2,3,4,5:");
        scanf("%[2345]", &bal);
        printf("\nОцінка %c", bal);
}
```

Тема 5. Базові алгоритмічні структури

§ 5.1. Розв'язання задач з використанням основних операторів С ++

«Початкові програмісти, особливо студенти, часто пишуть програми так: отримавши завдання, тут же сідають за комп'ютер і починають кодувати ті фрагменти алгоритму, які їм вдається придумати відразу. Змінним дають перші-ліпші імена типу x і y. Коли комп'ютер зависає, робиться перерва, після якого все написане стирається, і все повторюється заново. Періодично висловлюються сумніви в правильності роботи компілятора, комп'ютера і операційної системи. Коли програма доходить до стадії виконання, в неї вводяться довільні значення, після чого екран стає об'єктом пильного здивованого вивчення. «Працює» така програма зазвичай тільки в дбайливих руках господаря на одному наборі даних, а внесення в неї змін може привести автора до втрати віри в себе і ненависті до процесу програмування.

Але завдання полягає в тому, щоб навчитися підходити до програмування професійно. Зрештою, професіонал відрізняється тим, що може досить точно оцінити, скільки часу у нього займе написання програми, яка буде працювати в повній відповідності з поставленим завданням. Крім «розуму, смаку і терпіння», для цього потрібен досвід, а також знання основних принципів, вироблених програмістами протягом більш, ніж півстоліття розвитку цієї дисципліни. Навіть до написання найпростіших програм потрібно підходити послідовно, дотримуючись певної дисципліни. »(Павловська Т. А., стор.109)

Рішення задач з програмування передбачає ряд етапів:

1. Розробка математичної моделі. На цьому етапі визначаються вихідні дані і результати виконання завдання, а також математичні формули, за допомогою яких можна перейти від вихідних даних до кінцевого результату.
2. Розробка алгоритму. Визначаються дії, виконуючи які можна буде від вихідних даних прийти до необхідного результату.
3. Запис програми на деякій мові програмування. На цьому етапі кожно-

му кроці алгоритму ставиться у відповідність конструкція обраної алгоритмічної мови.

4. Виконання програми (вихідний модуль -> компілятор -> об'єктний модуль -> компоновщик -> виконуваний модуль)
5. Тестування і налагодження програми.

При виконанні програми можуть виникнути помилки 3 типів:

a. синтаксичні - виправляються на етапі компіляції;

b. помилки виконання програми (ділення на 0, логарифм від негативного числа і т. п.) - виправляються при виконанні програми;

c. семантичні (логічні) помилки - з'являються через неправильно зрозуміле завдання, неправильно складений алгоритм.

Щоб усунути ці помилки програма повинна бути виконана на деякому наборі тестів. Мета процесу тестування - визначення наявності помилки, знаходження місця помилки, її причини та відповідні зміни програми - виправлення. Тест - це набір вихідних даних, для яких заздалегідь відомий результат. Тест, який виявив помилку, вважається успішним. Налагодження програми закінчується, коли достатня кількість тестів виповнилася успішно, тобто програма на них видала правильні результати.

Для визначення достатньої кількості тестів існує два підходи. При першому підході програма розглядається як «чорний ящик», в який передають вихідні дані і отримують результати. Пристрій самого ящика невідомо. При цьому підході, щоб здійснити повне тестування, треба перевірити програму на всіх вхідних даних, що практично неможливо. Тому вводять спеціальні критерії, які повинні показати, яка кінцева множина тестів є достатнім для програми. При першому підході найчастіше використовуються наступні критерії:

1. Тестування класів вхідних даних, тобто набір тестів повинен містити по одному представнику кожного класу даних.

2. Тестування класів вихідних даних, набір тестів повинен містити дані достатні для отримання по одному представнику з кожного класу вихідних даних.

При другому підході програма розглядається як «білий ящик», для

якого повністю відомий пристрій. Повне тестування при цьому підході закінчується після перевірки всіх шляхів, що ведуть від початку програми до її кінця. Однак і при такому підході повне тестування програми неможливо, так як шляхів в програмі з циклами безліч. При такому підході використовуються наступні критерії:

1. Тестування команд. Набір тестів повинен забезпечувати проходження кожної команди не менше одного разу.

2. Тестування гілок. Набір тестів в сукупності має забезпечити проходження кожної гілки не менше одного разу. Це найпоширеніший критерій в практиці програмування.

§ 5.2. Лексеми

Коли компілятор обробляє програму, він розбиває програму на групи символів, які називаються лексемами. *Лексема* - це мінімальна одиниця тексту програми, яка має певний сенс для компілятора і яка не може бути розбита в подальшому. Операції, константи, ідентифікатори і ключові слова, описані раніше, є прикладами лексем. Знаки пунктуації, такі як квадратні дужки ([]), фігурні дужки ({}), кутові дужки (<>), круглі дужки і коми, також є лексемами. Межі лексем визначаються пробільними символами та іншими лексемами, такими як операції і знаки пунктуації. Щоб попередити неправильну роботу компілятора, забороняються пробільні символи між символами ідентифікаторів, операціями, що складаються з декількох символів і символами ключових слів.

Коли компілятор виділяє окрему лексему, він послідовно об'єднує стільки символів, скільки можливо, перш ніж перейти до обробки наступної лексеми. Тому лексеми, не розділені пробільними символами, можуть бути проінтерпретовані невірно.

Наприклад, розглянемо наступний вираз: i+++j

У цьому прикладі компілятор спочатку створює з трьох знаків плюс найдовшу з можливих операцій (++), а потім обробить знак, що залишився +, як операцію додавання (+). Вираз проінтерпретують як (i ++) + (j), а не як (i) + (++ j). У таких випадках необхідно використовувати пробільні

символи або круглі дужки, щоб однозначно визначити ситуацію.

§ 5.3. Пробільні символи

Пробіл, табуляція, переведення рядка, повернення каретки, нова сторінка, вертикальна табуляція і новий рядок - це символи, звані пробільними, оскільки вони мають те ж саме призначення, як і пропуски між словами і рядками на друкованій сторінці. Ці символи поділяють об'єкти, визначені користувачем, такі, як константи і ідентифікатори, від інших об'єктів програми.

Компілятор С ++ ігнорує пробільні символи, якщо вони не використовуються як роздільники або як компоненти константи-символу або рядкових літералів. Це потрібно мати на увазі, щоб додатково використовувати пробільні символи для підвищення наочності програми (наприклад, для перегляду редактором текстів).

§ 5.3.1 Порожній оператор

Найпростішою формою оператора є оператор: «;»

Він не робить нічого. Однак він може бути корисний в тих випадках, коли синтаксис вимагає наявність оператора, а вам оператор не потрібен. Він зазвичай використовується в наступних випадках:

- в операторах do, for, while, if в рядках, коли місце оператора не потрібно, але по синтаксису потрібно хоча б один оператор;
- при необхідності позначити фігурну дужку.

Синтаксис мови С ++ вимагає, щоб після мітки обов'язково слідував оператор. Фігурна ж дужка не є оператором. Тому, якщо треба передати управління на фігурну дужку, необхідно використовувати порожній оператор.

Пример:

```
int main ( )
 {
    :
    {
        if (...)  goto a;    // переход на скобку
            {
            ...
            }
        a:;
    }
    return 0;
}
```

§ 5.3.2 Блоки

Блок – це список операторів (можливо пустий), укладений у фігурні дужки:

```
{
    a=b+2;
    b++;
}
```

Блок дозволяє розглядати кілька операторів як один. Крім того, в блоці можна визначити змінну, яка буде автоматично знищена при виході з блоку.

§ 5.3.3 Опис

Опис - це оператор, який запроваджує ім'я в програмі. Опис задає тип цього імені. Тип визначає правильне використання імені або виразу. Опис може також ініціювати об'єкт з цим ім'ям. Виконання опису означає, що коли потік управління доходить до опису, обчислюється ініціалізований вираз (ініціалізатор) і виробляється ініціалізація.

§ 5.3.4 Літерали

Літерали (*literals*) – це постійні значення, такі як 1 або 3.14159 (π). Для кожного типу С ++ існують літерали, включаючи символьний і булевский типи, цілі числа і числа з плаваючою крапкою. Можливі рядкові літерали, хоча типу для зберігання рядків в С ++ не існує.

Деякі приклади

5	ціла константа
5u	u або U означає беззнакові константи
5l	l або L означає **long**
true	логічна константа
5.0	константа з плаваючою точкою, розуміється як **double**
5.0f	f або F — з плаваючою точкою, розуміється як **float**
0.3e-2	константа з плаваючою точкою **double**, e або E відокремлюють експонентну частину
5.0l	l або L в даному випадку розуміється как long double
'd'	символьна константа
"Visual"	рядкова константа

Ви звернули увагу при розгляді типів даних, що серед розглянутих типів даних відсутній «строковий» тип, на відміну від Pascal. Справа в тому, що компілятори С ++ підтримують лише рядкові літерали. З самим **поняттям строковий літерал** ви вже добре знайомі. Наприклад, в операторі cout << "Hello, World!"; використовується строковий літерал «Hello, World!». Іншими словами, строковий літерал - це набір довільних символів, укладений в лапки. Компілятор сприймає його саме як набір символів і ніяк обробляти його не збирається, навіть якщо в лапках виявляться якісь ключові слова і операції. Винятком є використання **escape**-послідовностей (керуючих послідовностей).

§ 5.4 Основні оператори мови С++.

§ 5.4.1 Базові конструкції структурного програмування.

У теорії програмування доведено, що програму для вирішення завдання будь-якої складності можна скласти тільки з трьох структур: лінійн-

ної, розгалуженої і циклічної. Ці структури називаються базовими конструкціями структурного програмування.

Лінійною називається конструкція, що представляє собою послідовне з'єднання двох або більше операторів.

Розгалуження - задає виконання одного з двох операторів, в залежності від виконання будь-якого умови.

Цикл - задає багаторазове виконання одного і того ж оператора або групи операторів.

§ 5.4.2 Слідування. Розгалуження. Цикл.

Метою використання базових конструкцій є отримання програми простої структури. Таку програму легко читати, налагоджувати і при необхідності вносити в неї зміни. Структурне програмування також називають програмуванням без goto, так як часте використання операторів переходу ускладнює розуміння логіки роботи програми. Але іноді зустрічаються ситуації, в яких застосування операторів переходу, навпаки, спрощує структуру програми.

Оператори управління роботою програми називають керуючими конструкціями програми. До них відносять:

- складові оператори;
- оператори умови;
- оператори циклів;
- оператори переходу.

§ 5.4.3 Оператор «вираз»

Будь-який вираз, що закінчується крапкою з комою, розглядається як оператор, виконання якого полягає в обчисленні цього виразу. Окремим випадком виразу є *порожній оператор*; (крапка з комою).

Приклади:

i++; *a*+=2; *x*=*a*+*b*;

§ 5.4.4 Складені оператори

До складених операторів відносять власне *складові оператори* і *блоки*. В обох випадках це послідовність операторів, укладена в фігурні дужки.

Блок відрізняється від складового оператора наявністю *визначень* в тілі блоку. *Наприклад*:

```
{
    n++;                    //це складовий оператор
    summa+=n;
}
                    {
                        int n=0;
                        n++;                    //це блок
                        summa+=n;
                    }
```

Весь вміст блоку розглядається компілятором мови як один оператор.

§ 5.4.5 Оператор вираз

Будь-який вираз, який закінчується крапкою з комою, є оператором. Виконання оператора виразу полягає в обчисленні виразу. Отримане значення виразу ніяк не використовується, тому, як правило, такі вирази викликають побічні ефекти. Зауважимо, що викликати функцію, неповернутих значення можна тільки за допомогою оператора вираження.

Приклади:

++*i*; – этот оператор представляет выражение, которое увеличивает значение переменной *i* на единицу.

a=***cos***(*b**5); – этот оператор представляет выражение, включающее в себя операции присваивания и вызова функции.

a(*x*, *y*); – этот оператор представляет выражение, вызывающее функцию ***a***.

§ 5.5 Оператори умови

Всі приклади, розглянуті раніше, виконувалися в порядку проходження операторів, що виконуються обов'язково по одному разу. Однак на практиці можливості таких програм досить обмежені. Більшість завдань вимагає від програми прийняття рішень в залежності від різних ситуацій. Мова С ++ має багато конструкцій для управління порядком виконання гілок програми. Для здійснення розгалуження використовуються оператори умови.

Оператори умови - це умовний оператор і оператор вибору (селектор). Вони застосовуються для перевірки деяких умов.

§ 5.5.1 Умовний оператор

Умовний оператор має *повну* (оператор **if-else**) та *скорочену* (оператор **if**) форму.

Скорочена форма (оператор **if**): **if (**вираз-умова**)** *оператор*;

<u>В якості виразу-умови</u> можуть використовуватися.

- арифметичні вирази,
- відношення з операціями порівняння,
- логічний вираз.

Якщо значення виразу-умови відмінне від нуля (тобто <u>істинно</u>), то виконується <u>*оператор*</u>, в іншому випадку не виконується <u>*нічого*</u> і управління передається наступному після **if** оператору.

Наприклад: **if** (x <y && x <z) *min* = x; В даному випадку, якщо x = 5, y = 6, z = 7, тобто x <y і x <z і вираз-умова істинно (приймає значення *true*), тоді виконується оператор *min* = x і змінна min приймає значення 5. якщо ж x = 9, y = 6, z = 11, тобто x не «y, але x <z, тоді вираз-умова помилкова (приймає значення false) і оператор взагалі не виконується, тобто значення min не зміниться.

Оператор може бути *блоковим*, тобто, в залежності від прийнятого рішення виконується не один, а цілий блок операторів. Весь вміст блоку розглядається компілятором мови як один оператор.

Однією з типових помилок при використанні оператора **if** є пропуск

фігурних дужок для позначення блоку виконуваних операторів.

```
int main()
{
        int x=0;
        int y=8;
        int z=0;
        if(x!=0)
        z++;
        y/=x; // Помилка!!! Ділення на 0!!!
        cout<<"x="<<x<<"   y="<<y;
}
```

```
int main()
{
        int x=0;
        int y=8;
        int z=0;
        if(x!=0)
        {
                z++;
                y/=x; //  Помилки немає!
        }
        cout<<"x="<<x<<"   y="<<y;
}
```

У лівій колонці блок після **if** не використовується, тому виконується тільки оператор z ++; і далі, без всяких умов y / = x, що призводить до помилки ділення на нуль. У правій колонці, завдяки блоку, такий розподіл виконується тільки в разі x! = 0, що допомагає уникнути помилки.

Перевірка на нульове значення використовується дуже часто в програмуванні.

Повна форма (оператор **if-else**):

if (вираз-умова) *оператор1*; **else** *оператор2*;

Якщо значення виразу-умови істинно (*true*, відмінно від нуля), то виконується *оператор1*, при помилковому (нульовому, *false*) значення виразу-умови виконується *оператор2*.

Слід зазначити, що комбінація **if-else** дозволяє значно спростити код програми.

Наприклад:

```
if (d>=0)
    {
        x1=(-b-sqrt(d))/(2*a);      x2=(-b+sqrt(d))/(2*a);
        cout <<  "\n x1=" << x1 << "x2=" << x2;
    }
    else cout << "\nРозв'язків немає";
```

Типовою помилкою програмістів є використання в умовних конструкціях оператора присвоювання замість оператора порівняння (= замість ==), що не приводить до помилки компілятора, але веде до невірного результату.

На практиці часто перевіряється умова являє собою перевірку значення деякої цілочисельний змінної на нульове значення, тому часто зустрічаються записи виду if (! X) ..., що означає **if** (x == 0), або **if** (x) ..., що означає **if** (x ! = 0)

Існує **альтернативна форма** оператора умови:

(вираз-умова)**?** *оператор1* **:** *оператор2*;

Його дія аналогічна повній формі розгалуження: якщо значення виразу-умови істинно (відмінно від нуля), то виконується *оператор1*, при помилковому (нульовому) значення виразу-умови виконується *оператор2*.

Повна і альтернативна форми оператора умови абсолютно аналогічні, тому вибір використання тієї чи іншої форми - це особиста справа програміста, але альтернативну форму краще використовувати, якщо входять оператори досить прості.

Наприклад: *max*=(*a*>*b*)? *a*:*b*;

§ 5.5.2 Оператор вибору switch (селектор)

Ще однією альтернативою умовного оператора може служити оператор вибору **switch**. Він використовується в тих випадках, коли необхідно проводити порівняння деякої змінної з великою кількістю однотипних змінних або літералів. Ця конструкція являє собою своєрідний перемикач. Перемикач визначає множинний вибір.

```
switch (вираз-селектор)
{
        case константа1 : оператор1;
        case константа2 : оператор2; break;
        . . .
        [default: операторы;]
}
```

При виконанні оператора **switch**, аналізується вираз-селектор, записаний після **switch,** він повинен бути цілочисельним.

Отримане значення послідовно порівнюється з константами, які записані слідом за **case**. При першій збіжності виконуються оператори помічені даною міткою. Якщо виконані оператори не містять оператора переходу **breake**, то далі виконуються оператори всіх наступних варіантів, поки не з'явиться оператор переходу або не закінчиться перемикач. Якщо значення виразу, записаного після **switch** не співпало з жодною константою, то виконуються оператори, які слідують за міткою **default**. Мітка **default** може бути відсутньою. Ключове слово **break** не є обов'язковим, але якщо воно відсутнє, то відбувається перегляд всіх варіантів, що сильно уповільнює роботу програми.

Якщо при виконанні різних констант слідує один і той же оператор, то їх можна об'єднувати в такий спосіб:

```
        case константа1 :
        case константа2:
        case константа3:
        case константа4:
                        оператор1;
```

§ 5.6 Оператори циклів.

Наступним потужним механізмом управління ходом послідовності виконання програми є використання циклів.

Цикл (повторення) - це процес виконання певного набору команд деяку кількість разів. Він має точку входу, перевіряючу умову і (необов'язково) точку виходу. Цикл, що не має точки виходу, називається нескінченним. Для нескінченного циклу перевіряюча умова завжди приймає істинне значення. Цикли можуть бути вкладеними один в одного довільним чином.

У мові С ++ є три оператора циклу - **for** (цикл з параметром), **while** (цикл з передумовою) і **do-while** (цикл з постумовою).

Група дій, що повторюються в циклі, називається його тілом. Одноразове виконання циклу називається його кроком.

§ 5.6.1 Цикл з передумовою (while).

Оператор циклу **while** виконує оператор або блок до тих пір, поки перевіряюча умова (вираз) залишається істинним. Він повинен виглядати так:
while (вираз-умова)

оператор;

В якості виразу-*умови* може використовуватися:

- відношення,
- логічний вираз,
- константа
- змінна логічного типу.

Якщо воно істинне, тобто не дорівнює 0, то тіло циклу виконується до тих пір, поки вираз-умова *істинна* (не стане *хибним*). Якщо потрібно перевірити кілька умов, то застосовують оператор *«кома»*. Оператор в даному циклі може бути порожнім, простим або складеним. Для того, щоб стався вихід з циклу, необхідно *змінювати* значення параметра в циклі в операторі. *Параметр циклу* - це дане, яке входить у вираз-умову.

Дія команди.

1. Обчислюються значення виразу-умови. Якщо значення виразу-умови *істинне*, то переходимо до пункту 2), якщо *хибне* - до пункту 3).

2. Виконується оператор і відбувається перехід до пункту 1).

3. Виконується перехід до наступної після **while** команди.

Циклу **while** необхідний початковий оператор, який ініціалізує змінну управління циклом (параметр, лічильник циклу). Зауважимо також, що всередині циклу **while** повинен знаходиться оператор, що змінює значення змінної управління циклом.

Якщо вираз являє собою константу з *істинним* значенням, тіло циклу буде виконуватися завжди, і, отже, виходить нескінченний оператор. Цикл також виявиться нескінченним, коли умова, визначена в виразі-умові спочатку, *істинно* і ніде далі в тілі циклу не змінюється. Якщо ж перевірочне умова повертає *хибне* значення, здійсниться вихід з циклу і тіло оператора **while** буде пропущено.

Досить часто в якості виразу-умови використовується оператор присвоювання. Так як при цьому повертається деяке число, в операторі **while** фактично проводиться порівняння отриманого значення з нулем (нуль - еквівалент помилкового значення) з подальшим прийняттям рішення про вихід з циклу або про його продовження.

Таким чином, команда **while** може бути виконана один раз, кілька разів або жодного разу, в залежності від виразу-умови.

Приклад.

```
int a, s=0;
while (a!=0)
  {
    cin>>a;
    s+=a;
  }
```

Приклад.

```
int x=4, s=0;
while (x<=8)
    {
      s+=x;
```

Кроки	*x*	*s*
0	4	0
1	5	4
2	6	9
3	7	15
4	8	22
5	9	30

```
        x++;
    }
```

Після виконання команди *s*=30, *x*=9. В цьому випадку цикл виконується 5 разів.

А ось після виконання команди

```
int x=4, d=1;
while (x>10) d*=x;
```

змінна *d* свого значення не змінить, так як значення виразу *x*>10 одразу хибне і тому команда *d**=*x* в циклі **while** виконуватися не буде. Тобто цикл не виконається ні разу.

§ 5.6.2 Цикл з післяумовою (do-while)

На відміну від оператора **while** цикл **do-while** спочатку виконує тіло (оператор або блок), а потім вже здійснює перевірку вираження на істинність. Синтаксис оператора має вигляд:

```
do
    оператор;
while (вираз-умова);
```

Тіло циклу виконується до тих пір, поки вираз-умова є *істинною*.

Дія команди.

1. Виконується оператор
2. Обчислюється значення виразу-умови.
3. Якщо значення виразу-умови *істинне*, то переходимо до пункту 1), якщо *хибне* - виконується перехід до наступної після **do-while** команди.

Оператор в циклі **do-while**, на відміну від попереднього циклу, буде виконуватися хоча б один раз завжди.

Одне з часто використовуваних застосувань даного оператора - запит до користувача на продовження виконання програми:

```
int main()
{
    char ch;
    do
```

```
    {
        // Тіло програми
        …
        cout<<"Продовжувати виконання?";
        cin>>ch;
    }
    while(ch!='N')
}
```

Таким чином, тіло програми буде повторюватися до тих пір, поки користувач на питання «Продовжувати виконання?» не відповість символом *N*. При цьому в якості змінної, використовуваної для зберігання цього значення, виступає *ch*.

Приклад:

```
int a, s=0;
do
  {
     cin>>a;   s+=a;
  }
while(a!=0);
```

Зверніть увагу, що складовий оператор або блок обов'язково укладається в операторні дужки {}.

Приклад.

```
int x=5, y=0;
do
  {
     y+=x; z=2*x; x–
     =2;
  }
while(x>1);
```

Кроки	*y*	*z*	*x*
0	0	1	5
1	5	10	3
2	8	6	1

Після виконання команди *y*=8, z=6, x=9. В цьому випадку цикл виконується 2 рази.

§ 5.6.3 Цикл з параметром (for)

Синтаксис циклу **for** має вигляд:

for (вираз_1**;** вираз-умова**;** вираз_3)

оператор або блок операторів;

вираз 1 і вираз 3 можуть складатися з декількох виразів, розділених комами.

Вираз_1 – найчастіше задає початкові умови для циклу (ініціалізація), воно виконується один раз.

Вираз-умова – визначає умову виконання циклу (умова виходу з циклу), до тих пір, поки воно *істинно* (не дорівнює 0), тіло циклу виконується, а потім обчислюється значення виразу_3.

Вираз_3 – задає зміна параметру циклу або інших змінних (корекція).

Цикл триває доти, поки вираз-умова не стане *хибною* (дорівнює 0).

Будь-який вираз може бути відсутнім, але розділяючі їхні порожні оператори **;** повинні бути обов'язково. Тому найпростіший приклад нескінченного циклу **for** (виконується постійно до примусового завершення програми ззовні) виглядає наступним чином: **for (;;) cout** << "Нескінченний цикл ...";

Дія команди.

1. Обчислюються значення виразу_1.
2. Обчислюються значення виразу -умови.

3. Якщо значення виразу-умови *істинне*, то виконується оператор. Якщо *хибне* - програма переходить до наступної після циклу **for** команди.

3. Обчислюються значення виразу_3 і вираз-умова і виконується пункт 3.

Типова помилка програмування циклів **for** - зміна значення лічильника як в конструкції (вираз_3), так і в тілі циклу. Це може призводити до таких негативних наслідків, як «випадання» або повтор ітерацій.

Приклади використання циклу з параметром.

1) Зменшення параметра:

```
for (int n=10; n>0; n--)
    {
```

оператор
}

2) Зміна кроку коректування:

for (*n*=2; *n*<60; *n*+=13)
{
оператор
}

3) Можливість перевіряти умову відмінну від умови, що накладається на число ітерацій:

for (*num*=1;*num***num***num*<216; *num*++)
{
оператор
}

4) Корекція може здійснюватися не тільки за допомогою додавання або віднімання:

for (*d*=100.0; *d*<150.0;*d**=1.1)
{
оператор
}

for (*x*=1; *y*<=75; *y*=5*(*x*++)+10)
{
оператор
}

5) Можна використовувати декілька ініціюючих або коригувальних виразів:

for (*x*=1, *y*=0; *x*<10; *x*++, *y*+=*x*);

Приклад. Сумму цілих чисел з проміжку від 1 до 15 можна обчислити одним із способів:

1) **int** *n*=1, S=0; **for** (; *n*<16; *n*++) *S*+=*n*;
2) **for** (**int** *n*=1, S=0; *n*<16; *n*++) *S*+=*n*;
3) **for** (**int** *n*=1, S=0; *n*<16; *S*+=*n*++);

4) **for** (**int** *n*=1, S=0; *n*<16; *S*+=*n*, *n*++);

Зверніть увагу на те, що в якості тіла циклу в 3) і 4) може використовуватися порожній оператор (;). Його застосування при відсутності оператора обов'язково, так як компілятор вимагає, щоб тіло циклу було не пустим. В 4) випадку ілюструється застосування оператора «кома».

Приклад. Кількість і добуток всіх парних чисел з проміжку від 4 до 11 можна обчислити так:

```
int n, D, kil;
   for ( D=1, kil=0, n=4; n<=11; n+=2)
         {
              D*=n;
              kil++;
         }
```

§ 5.7 Оператори переходу

Оператори переходу виконують передачу управління.

1) **break** – оператор переривання циклу. Якщо потрібно передбачити вихід з оператора циклу в кількох місцях, не чекаючи виконання решти коду, то для цих цілей служить оператор **break**.

```
{
     <оператори>
     if (<вираз_умова>) break;
     <оператори>
}
```

Тобто оператор **break** доцільно використовувати, коли умову продовження ітерацій треба перевіряти в середині циклу. На практиці оператор **break** часто використовують для виходу з нескінченного циклу.

Приклад: пошук суми чисел, що вводяться з клавіатури, до тих пір, поки не буде введено 100 чисел або 0

```
for (s=0, i=1; i<100;i++)
```

```
{
    cin>>x;
    if (x==0) break;    // якщо ввели 0, то підсумовування
закінчується
    s+=x;
}
```

2) **continue** – перехід до наступної ітерації циклу. Так само як і **break**, оператор **continue** перериває виконання тіла циклу, але він наказує програмі перейти на наступної ітерації циклу. Він використовується, коли тіло циклу містить розгалуження.

Приклад: Пошук кількості і суми додатних чисел

```
for( k=0,s=0,x=1; x!=0;)
{
    cin>>x;
    if (x<=0) continue;
    k++;  s+=x;
}
```

3) Оператор безумовного переходу **goto** має формат:

goto *мітка*;

У тілі тієї ж функції повинна бути присутня конструкція: *мітка: оператор;*

Мітка - це звичайний ідентифікатор, областю видимості якого є функція, з розташованим за ним символом двокрапки (:). Мітками позначають будь-який оператор, на який в подальшому повинен бути здійснений безумовний перехід. Оператор **goto** передає управління оператору, який стоїть після мітки. Використання оператора **goto** виправдано, якщо необхідно виконати перехід з декількох вкладених циклів або перемикачів вниз по тексту програми або перейти в одне місце функції після виконання різних дій.

Застосування **goto** порушує принципи структурного і модульного програмування, за якими всі блоки, з яких складається програма, повинні мати тільки один вхід і тільки один вихід.

Не можна передавати управління всередину операторів **if**, **switch** і *циклів*. Не можна переходити всередину блоків, що містять ініціалізацію, на оператори, які стоять після ініціалізації.

Взагалі кажучи, використання структурного та об'єктно-орієнтованого підходів до програмування дозволяє повністю відмовитися від застосування операторів безумовного переходу. Однак на практиці часто бувають випадки, коли **goto** значно спрощує код програми. В особливій мірі це твердження стосується вкладених конструкцій **switch-case** і **if-else.**

Приклад:

```
int k;
goto m;
. . .
    {
        int a=3, b=4;
        k=a+b;
        m: int c=k+1;
        . . .
    }
```

У цьому прикладі при переході на мітку *m* не виконуватиметься ініціалізація змінних *a, b* і *k.*

4) Оператор **return** - оператор повернення з функції. Він завжди завершує виконання функції і передає управління в точку її виклику. Вид оператора:

return [вираз];

Про нього ми ще поговоримо пізніше більш детально (при вивченні функцій користувача).

§ 5.8 Перетворення типів

При виконанні операцій відбуваються неявні перетворення типів в наступних випадках:

- при виконанні операцій здійснюються звичайні арифметичні перетворення;
- при виконанні операцій присвоювання, якщо значення одного типу присвоюється змінній іншого типу;
- при передачі аргументів функції..

Крім того, в С ++ є можливість явного приведення значення одного типу до іншого. В операціях присвоювання тип значення, яке присвоюється, перетвориться до типу змінної, яка отримує це значення. Допускається перетворення цілих і плаваючих типів, навіть якщо таке перетворення веде до втрати інформації.

Перетворення цілих типів зі знаком. Ціле зі знаком перетвориться до більш коротшого цілого зі знаком, за допомогою усічення старших бітів. Ціле зі знаком перетвориться до більш довгого цілому зі знаком, шляхом розмноження знака. При перетворенні цілого зі знаком до цілого без знаку, ціле зі знаком перетвориться до розміру цілого без знаку і результат розглядається як значення без знаку.

Перетворення цілого зі знаком до плаваючого типу відбувається без втрати інформації, за винятком випадку перетворення значення типу **long long** або **unsigned long long** до типу **float**, коли точність часто може бути втрачена.

Перетворення цілих типів без знаку. Ціле без знаку перетворюється до більш коротшого цілого без знаку або зі знаком шляхом усічення старших бітів. Ціле без знаку перетворюється до більш довгого цілого без знаку або зі знаком шляхом доповнення нулів зліва. Коли ціле без знаку перетворюється до цілого зі знаком того ж розміру, бітове подання не змінюється. Тому значення, яке воно представляє, змінюється, якщо знаковий біт встановлений (дорівнює 1), тобто коли вихідне ціле без знакау більше ніж максимальне додатнє ціле зі знаком, такої ж довжини. Цілі значення без знаку перетворюються до плаваючого типу, шляхом пере-

творення цілого без знаку до значення типу **signed long**, а потім значення **signed long** перетворюється в плаваючий тип. Перетворення з **unsigned long** до типу **float**, **double** або **long double** виробляються з втратою інформації, якщо значення, яке потрібно більше, ніж максимальне додатнє значення, яке може бути представлено для типу **long**.

Перетворення плаваючих типів. Величини типу **float** перетворюються до типу **double** без зміни значення. Величини **double** і **long double** перетворюються до **float** с деякою втратою точності. Якщо значення занадто велике для **float**, то відбувається втрата значущості, про що повідомляється під час виконання. При перетворенні величини з плаваючою точкою до цілих типів вона спочатку перетворюється до типу **long** (дрібна частина плаваючою величини при цьому відкидається), а потім величина типу **long** перетвориться до необхідного цілого типу. Якщо значення занадто велике для **long**, то результат перетворення не визначений.

Перетворення з **float, double** або **long double** до типу **unsigned long** проводиться з втратою точності, якщо значення, яке потрібно більше, ніж максимально можливе додатнє значення, представлене типом **long.**

§ 5.8.1 Перетворення при обчисленні виразів

При виконанні операцій проводиться автоматичне перетворення типів, щоб привести операнди виразів до загального типу або щоб розширити короткі величини до розміру цілих величин, використовуваних в машинних командах. Виконання перетворення залежить від специфіки операцій і від типу операнда або операндів.

Розглянемо загальні арифметичні перетворення.

1. Операнди типу **float** перетворюються до типу **double**.
2. Якщо один операнд **long double**, то другий перетворюються до цього ж типу.
3. Якщо один операнд **double**, то другий також перетворюються до типу **double**.
4. Будь-які операнди типу **char** і **short** перетворюються до типу **int**.

5. Будь-які операнди **unsigned char** або **unsigned short** перетворюються до типу **unsigned int**.
6. Якщо один операнд типу **unsigned long**, то другий перетворюється до типу **unsigned long**.
7. Якщо один операнд типу **long**, то друий перетворюється до типу **long**.
8. Якщо один операнд типу **unsigned int**, то другий операнд перетворюється до цього ж типу.

Таким чином, можна відзначити, що при обчисленні виразів операнди перетворюються до типу того операнда, який має найбільший розмір.

Тема 6. Вказівники. Масиви

§ 6.1. Поняття вказівника

Будь-який об'єкт програми (змінна) займає в пам'яті певну область. Місце розташування об'єкта в пам'яті визначається його адресою. При оголошенні змінної для неї резервується місце в пам'яті, розмір якого залежить від типу даної змінної, а для доступу до вмісту об'єкту служить його ім'я (ідентифікатор). При зверненні до змінної визначається її фактичне розташування в пам'яті, і всі дії ведуться вже з цією конкретною ділянкою.

Для вирішення різних завдань програмування (робота з масивами, побудова графічних зображень і використання динамічних ефектів) потрібно знати не тільки значення деякої змінної, але і її адресу в оперативній пам'яті. Для визначення адреси в пам'яті є унарная операція визначення адреси:

&<назва даного>

Приклад 1. Розглянемо програму:

```
#include <iostream.h>
#define nr '\n'
int main()
{
    int a = 40000;
    cout << "Значення змінної a = " << a << nr;
    cout << "Адреса змінної a: " << &a << nr;
}
```

В результаті виконання цих команд на екрані отримаємо:

Значення змінної *a* = 25

Адресою змінної *a* є 0xaf72254

Адреси зображуються шістнадцятковими числами. Адреси локальних змінних розміщуються в стеку і йдуть у зворотному порядку (стек росте в напрямі молодших адрес). Результат може відрізнятися навіть при повторному запуску програми, так як неможливо передбачити, за якою адресою почнуть розміщуватися змінні. Різниця в адресах між першою та другою змінними завжди буде однакова і складе розмір однойменних типів. Для різних типів механізм виділення пам'яті більш складний.

§ 6.2.1. Вказівники.

Потужним засобом розробника програмного забезпечення на С++ є можливість здійснення безпосереднього доступу до пам'яті. Для цієї мети передбачається спеціальний тип змінних – вказівник (pointer). Вказівник являє собою змінну, значення якої є адресою комірки пам'яті. Він вказує на початок області оперативної пам'яті комп'ютера, де зберігається дане. Вказівники дають можливість оперувати не з іменами даних, а безпосередньо звертатися до областей пам'яті комп'ютера. Найбільша ефективність застосування вказівників у розробці програм досягається при використанні їх з масивами і символьними рядками.

Оголошення вказівника має наступний синтаксис:

<тип об’єкта>* <ім’я вказівника>

Тут <тип об'єкта> визначає тип даних, на які посилається вказівник з назвою <ім'я вказівника>. Можна створювати вказівники на константи, змінні, функції, інші вказівники тощо. Символ «зірочка» повідомляє компілятору, що оголошена змінна є вкпзівником і належить до типу змінної, тому його рекомендується ставити поряд з типом, а від імені змінної відокремлювати пробілом, за винятком тих випадків, коли описуються кілька вказівників. При описі кількох вка-

зівників знак * ставиться перед ім'ям змінної-вказівника, так як інакше не зрозуміло, що ця змінна також є вказівником.

Приклад 2.

int* *nomer*;

float **rost*, **ptrA*, **ptrB*;

Тут оголошений вказівник на цілий тип *nomer* и вказівники на дійсний тип *rost, ptrA, ptrB.*

Базовий тип вказівника визначає тип даних, на які він буде посилатися.

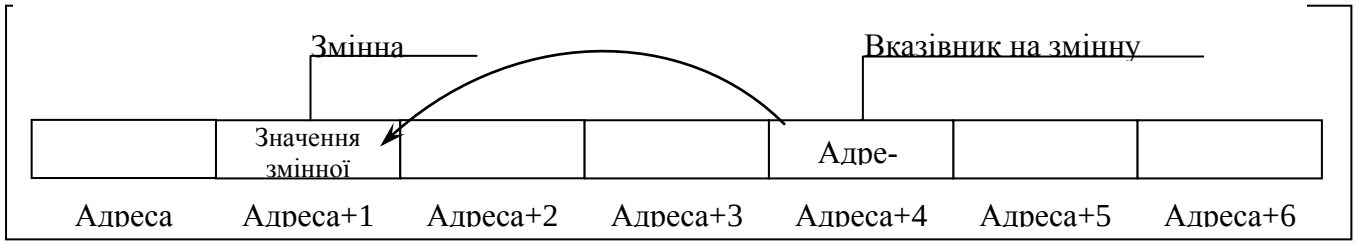
Символ * означає "вказівник на".

Номер байта, з якого починається в пам'яті змінна, називається ***адресою*** змінної. Також про адресу кажуть, що вона ***вказує*** на визначений байт. Таким чином, ***вказівник*** є просто адресою байта пам'яті.

Присвоїти значення вказівниками можна так:

<ім'я вказівника>=<адреса змінної>.

Оскільки вказівник являє собою посилання на деяку область пам'яті, йому може бути присвоєна тільки адреса деякої змінної (або функції), а не саме значення. У випадку некоректного присвоєння компілятор видасть відповідне повідомлення про помилку. Присвоїти



значення вказівникам можна так:

<ім’я вказівника>=<адреса змінної>.

Приклад 3. Нехай в програмі оголошенні змінні:

int *n* = 10;

float *stud1*, *stud2*, *stud3*;

Тоді можна визначити вказівник на ці змінні: *nomer*=&*n*;

rost=&*stud1*; *rost*=&*stud3*;

Приклад 4. Знайти довжини (в байтах) вказівників на різні типи даних можна так:

```
int *prt_i;
double *ptr_d;
char *ptr_c;
cout << "\nНа цілий тип " << sizeof(ptr_i);
cout << "\nНа дійсний тип " << sizeof(ptr_d);
cout << "\nНа символьний тип " << sizeof(ptr_c);
```

Якщо виконати цю програму, то буде видно, що всі вище описані вказівники в пам'яті комп'ютера мають однаковий обсяг. Це тому, що для вказівників в залежності від операційної системи і версії компілятора резервується 2 або 4 байта в оперативній пам'яті.

§ 6.1.2 Розіменування вказівників

Вказівники допомагають здійснювати безпосередній доступ до пам'яті і для того, щоб отримати (прочитати) *значення*, записане в області пам'яті, на яку посилається вказівник, використовують операцію непрямого звернення або ***операцію розіменування*** (*). При цьому використовується ім'я вказівника із зірочкою перед ним.

Приклад 5.

```
int x=1, y=22;
int *ptr=&y;
int* ip;
ip=&x;
y=*ip;
*ip=*ip+10;
```

У наведеному фрагменті оголошуються дві змінні цілого типу *x* і *y* і два вказівника на тип **int**, причому один з них відразу ініціалізова-

ний першою адресою змінної y, а другий відразу оголошується як покажчик на цілий тип, а потім йому присвоюється значення адреси змінної x. Крім того присвоюється нове значення змінної y, застосовуючи операцію розіменування. Значення *ptr також тепер зміниться, оскільки цей вказівник вказував на змінну y, а її значення в передостанньому рядку змінилося. В останньому рядку демонструється спосіб виконання додавання, використовуючи разіменований вказівник.

На практиці досить часто застосовується так званий порожній вказівник (типу **void**), який може вказувати на об'єкт будь-якого типу. Для отримання доступу до об'єкта, на який посилається вказівник **void**, його необхідно попередньо привести до того ж типу, що й тип самого об'єкта.

Приклад 6.

```
char Let='T';
int nNum=9;
void *ptr;
ptr=&Let;
*(char*)ptr='L';
ptr=&nNum;
*(int*)ptr=43;
```

Спочатку створюються два різнотипних об'єкта *Let* та *nNum* і порожній вказівник *ptr*, який ініціалізується адресою символьної змінної. Далі *ptr* розіменовується, приводиться до типу **char*** і за допомогою непрямої адресації модифікується значення символу *Let*. Аналогічним чином покажчик ініціалізується значенням адреси змінної *nNum*, приводиться до цілочисленного типу, після чого стає можливим зміна вмісту змінної *nNum*.

Таким чином, з покажчиками використовуються два оператора:

"**&**" – повертає адресу пам'яті, за яким розташований операнд;

"*****" – повертає значення змінної, на яку вказує операнд.

§ 6.1.3 Арифметика вказівників.

Операції, що виконуються з допомогою вказівників, часто називають ***операціями непрямого доступу***, оскільки ми побічно отримуємо доступ до змінної за допомогою іншої змінної.

До вказівників можуть застосовуватися арифметичні операції і операції порівняння.

Компілятор, знаючи тип вказівника, обчислює розмір змінної того ж типу, після чого модифікує адресу, що міститься у вказівнику відповідно до заданої арифметичною операцією, але з урахуванням обчисленого для даного типу розміру. Це означає, що якщо оголошений вказівник типу **double**, що займає 8 байт пам'яті, то операція, наприклад, інкремента вказівника збільшить значення адреси не на один, а на 8 байт.

Вказівники можна віднімати один від одного, тим самим визначаючи кількість елементів (одного і того ж типу, на який вказують вказівники), розташованих між ними. Цей прийом буває корисним при операціях з масивами і символьними рядками.

Операція	Приклади й пояснення
==, !=, >=, <=, >, <	Порівнює значення двох вказівників (адреси, на які вони вказують). *Наприклад*, якщо вказівники вказують на одне і те ж дане, то результатом порівняння *vk*1==*vk2* буде істина, інакше – брехня.
++, --	Інкремента і декремента вказівників
–	*vk*1–*vk*2. Використовується для визначення кількості елементів, які знаходяться між двома вказівниками.
+, –	*vk*1+*k*, *vk*1–*k*. Знаходить вказівник, який зміщений відносно даного на *k* одиниць.

Існує можливість використання при оголошенні вказівників ключового слова **const**. При цьому слід брати до уваги, що застосування даного специфікатора трактується наступним чином:

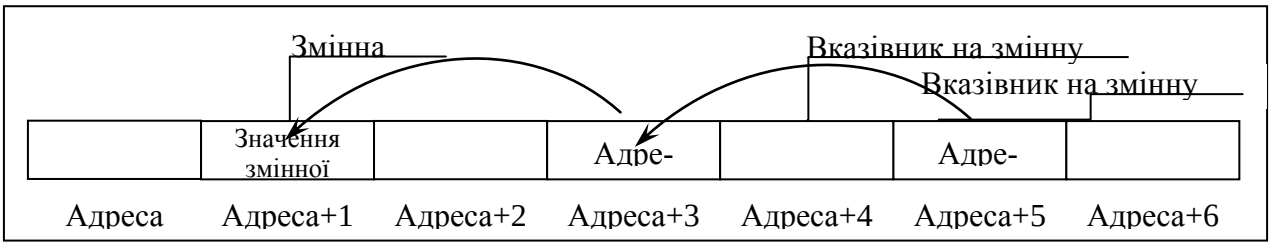
Приклад 7.

```
int* pi=&i;                    // вказівник на цілу змінну.
const int* pci=&ci;            // вказівник на цілу константу.
int* const cpi=&i;             // вказівник-константа на змінну цілого типу.
const int* const cpc=&ci;      // вказівник-константа на цілу константу.
```

Вказівники *можна переадресовувати.* Зокрема, у прикладі 3 вказівником *rost* спочатку присвоєно адресу змінної *stud1*, а потім – змінної *stud3*. Це правило не діє, якщо вказівник є константою. Постійний вказівник вказує на одну і ту ж адресу і оголошують його так: **const int ****god*;

§ 6.2 Вказівники на вказівники

Вказівники самі можуть посилатися на інші вказівники. При цьому в комірках пам'яті, на які посилається вказівник, міститься не значення, а адреса якого-небудь об'єкта. Сам об'єкт також може бути



вказівником і т. д.

Тут за Адресою+1 зберігається значення деякої змінної, на яку вказує звичайний вказівник, що знаходиться за Адресою+3 (зберігає значення Адреса+1), на яку в свою чергу посилається вказівник, що знаходиться за Адресою+5 (містить в якості значення Адресу+3).

Синтаксис вказівника на вказівник виглядає так:

int ***pPtrInt*;

Оголошення вказівника на вказівник, який сам є вказівником:

char ****ppSymbol*;

При оголошенні вказівник на вказівник може ініціалізуватися адресою об’єкта:

char **pSymb* = &*myChar*;

char ***pPtr* = &*pSymb*;

char ****ppPtr* = &*pPtr*;

Число символів * при оголошенні говорить про «порядок» вказівника. Щоб отримати доступ до значення, такий вказівник повинен бути розіменованний відповідну кількість разів (по числу символів *).

§ 6.3. Посилання.

Посилання – особливий тип даних, що є прихованою формою вказівника, який при використанні автоматично розіменовується. Іншими словами, він може використовуватися просто як інше ім'я, або псевдонім (синонім) об'єкта. При оголошенні посилання перед її ім'ям ставиться знак амперсанда (&), а сама вона повинна бути тут же проініціалізована ім'ям того об'єкта, на який посилається:

тип &ім'я посилання = ім'я;

Тип об'єкта, на який вказується посилання, може бути будь-яким. Оголошення неініціалізованого посилання викликає повідомлення компілятора про помилку.

Приклад 8.

int *x*;

int &*sx* = *x*;

Будь-яка зміна значення посилання веде за собою зміну того об'єкта, на який дане посилання вказує.

Приклад 9.

```
int x;
int &sx = x;
int sx+=10;                    // те ж, що й
x+=10;
```

Використання посилань не пов'язано з додатковими витратами пам'яті.

Слід зазначити, що посилання не можна перепризначувати – ініціалізувавши посилання якось адресою деякої змінної, будь-яка дія з посиланням позначається на самому об'єкті. Спроба змінити наявне посилання якоїсь іншої змінної призведе до присвоєння оригіналу об'єкта значення другої змінної.

Крім того, слід врахувати, що посилатися можна тільки на сам об'єкт. Не можна оголосити посилання на тип об'єкту.

Ще одне обмеження, що накладається на посилання, полягає в тому, що вони не можуть вказувати на нульовий об'єкт (приймає значення NULL). Таким чином, якщо є ймовірність того, що об'єкт в результаті роботи програми стане нульовим, від посилання слід відмовитися на користь застосування вказівника.

§ 6.4. Масиви

Масив – це впорядкований скінчений набір даних одного типу, які зберігаються в *послідовно розташованих* комірках оперативної пам'яті і мають спільну назву. Назву масиву надає користувач.

Масив складається з *елементів*. Кожний елемент має індекси, якими його можна знайти в масиві. Кількість індексів визначає розмірність масиву. Розрізняють одно - та багатовимірні масиви. Наприклад, двовимірний масив даних – це таблиця, яка складається з декількох рядків і стовпців. У математиці поняття масив відповідають поняття вектора і матриці.

Загальний вид конструкції описання одновимірного масиву такий:

<тип> <ім’я масиву>[<розмір>]

Розмір – *кількість* елементів масиву. Розмір масиву необхідно знати і задавати відразу, оскільки компілятор повинен зарезервувати для нього необхідний обсяг пам'яті. Розміром може бути тільки постійна величина (не змінна).

Ім'я масиву в програмі змінювати не можна – це постійна величина, яка містить адресу першого елемента. Отже, назва масиву є вказівником на перший елемент.

Наприклад, команда **int** stud[5] оголошує масив з ім'ям *stud*, який складається з п'яти цілих чисел; команда **float** *rost*[10] оголошує масив *rost*, який містить десять чисел дійсного типу; **char** *alphavit*[6] – оголошення масиву з 6 символів.

Звернутися до елементів масиву можна двома способами: *за допомогою імені масиву* або *використовуючи вказівники*.

Зауваження 1. В навчальних цілях використання імен більш наочно, однак у відповідності з професійним підходом слід віддавати перевагу вказівниками.

Нумерація елементів масиву завжди починається з *нуля*. Щоб звернутися до деякого елементу, необхідно записати ім'я масиву, а в квадратних дужках – його номер. Наприклад, змінна *stud[2]* є третім елементом масиву *stud*, а *stud[4]* – п'ятим, оскільки масив *stud* має елементи *stud[0], stud[1], stud[2], stud[3] і stud[4]*.

Зауваження 2. Компілятор мови С++ не контролює чи належить індекс заданому діапазону. Відповідальність за це несе програміст. Наприклад, якщо в програмі оголосити масив *mas* з п'ятьма речовими числами і написати команду mas[54]=2, то повідомлення про помилку не буде, проте невідомо, в яку ділянку пам'яті потрапить число 2 і що станеться.

Назва масиву *stud* є вказівником на його перший елемент. Змінна **stud* містить значення першого елемента масиву (елемента *stud[0]).* Оскільки всі елементи масиву розміщені в послідовних комірках оперативної пам'яті комп'ютера, то покажчик (*stud+1*) буде вказувати на другий елемент масиву (зміщення відносно вказівки *stud* на одну одиницю пам'яті), а покажчик (*stud+4*) – на п'ятий (зсув на чотири одиниці).

Ініціалізувати масив (задати значення елементів масиву) можна одним із способів:

• використовуючи принцип за замовчуванням;

• безпосередньо під час його оголошення;

• застосовуючи команду присвоєння;

• під час введення даних з клавіатури.

За замовчуванням всім елементам масиву присвоюється значення 0. Масив можна ініціалізувати повністю або частково відразу під час його оголошення, записуючи значення змінних через кому в фігурних дужках.

Наприклад

```
int Stud[] = {2, 10, 5, 7, 3};
float rost[10]= {163.4, 154.6, 170, 172.8};
char alphavit[6] = "Азбука"
```

чи **char** *alphavit*[6] = {А’, ‘б’, ‘е’, ‘т’, ‘к’, ‘а’}.

Перші чотири елемента масиву *rost* були проініціалізовані, а інші – ні.

Якщо масив повністю буде ініціалізований під час оголошення, то його розмір вказувати не обов'язково. У цьому випадку компілятор сам визначає, скільки пам'яті необхідно зарезервувати. У наведеному прикладі масив *stud* буде складатися з п'яти цілих чисел.

Присвоїти значення інших елементів масиву *rost* або змінити значення вже проініціалізованих елементів можна командою прис-

воювання, наприклад, так: *rost[3]*=175.4, *rost[9*]=184.1 або так: **(rost+2)*=164.5, **(rost+7)*=148.0 тощо. Елементи масиву також можна вводити з клавіатури під час виконання програми, як ми це робимо для змінних простих стандартних типів.

Масиви-константи (значення яких змінювати в програмі можна) оголошують так: **const int** *flag[]* = {1, 2}.

Постійні масиви потрібно ініціалізувати під час оголошення, інакше елементам масиву автоматично будуть присвоєні нульові значення, а змінювати їх не можна.

Для обробки елементів масиву найчастіше використовують команду циклу *for*, хоча можна застосувати і *while* або *do-while*.

Приклад 1. Створити масив з перших ста цілих чисел і обчислити суму всіх його значень можна одним із способів:

```
int n[100];        // 1-й спосіб
int S=0;
for (k=0; k<100;)
{
     *(n+k)=++k;
     S+=*(n+k);
}
```

```
int n[100];        // 2-й спосіб
int S=0;
for (k=0; k<100; k++)
   {
      n[k]=k+1;
      S+=n[k];
   }
```

Задачі на знаходження в масиві конкретних даних вирішують методом сканування (перебору, перегляду) *всіх елементів* масиву за допомогою циклу і умовної команди, де зазначають умови пошуку потрібних даних.

Тема 7. Функції

Мова програмування С ++ дає можливість реалізовувати концепцію структурного аналізу. Структурний аналіз полягає в попередній обробці складного завдання або громіздкого алгоритму і розподілі його на окремі прості частини. В С ++ ці частини реалізують за допомогою функцій. Окремі функції об'єднують в одну загальну програму. У скомпільованому вигляді така програма утворює модуль. Функцію можна уявити як підпрограму, яка несе закінчене смислове навантаження. У кожній програмі обов'язково має бути присутня головна функція - *main ()*. Розрізняють стандартні функції і функції користувача. Функція користувача - це пойменована група команд, яка оголошена в заголовки (або в основній програмі). До меню Ви можете вибрати (викликати) з будь-якого місця програми необхідну кількість разів.

§ 7.1. Оголошення функцій користувача.

Прототипи стандартних функцій розміщені в папці **INCLUDE**. Кожну функцію користувача перед першим викликом, перш за все, необхідно оголосити (задекларувати, створити прототип, сигнатуру). Зазвичай оголошення функцій розміщують в заголовних файлах, які потім підключаються до початкового тексту програми за допомогою директиви **#include.** Але синтаксис мови надає можливість розмістити прототип і в основній програмі.

Функцію користувача оголошують так:

<Тип функції> <ім'я функції> (<список формальних параметрів>); де, тип функції – це тип даного, яке функція повертає в основну програму. Тип самої функції можна не вказувати.

За замовчуванням функція повертає в програму дане цілого типу **int.** Функцію, яка ніякі результати в програму не повертає, оголошують з типом **void**. Для функції, яка не залежить ні від яких параметрів, в круглих дужках записують службове слово **void.**

Ім'я функції дає користувач за правилами створення ідентифікаторів.

У списку формальних параметрів через кому записують змінні, вказуючи їх типи. Тип необхідно вказувати для кожної змінної окремо. Імена змінних можна пропускати (не вказувати). Якщо функція не приймає ніяких значень, то список формальних параметрів може бути відсутнім. Круглі дужки опускати не можна.

Приклад.

float *Summa* (int *kol*, float *cena*); *kod* (**int** *k1*, int *k2*);

void *drobi* (**float, float**); **double** *loto* (**void**);

Під час оголошення можна відразу форматувати формальні параметри функції, тобто присвоювати їм певні значення. Такі значення називають значеннями за замовчуванням. Їх записують в кінці списку. Значення таких параметрів в програмі можна змінювати.

Приклад.

float *Summa* (**int** *kol*, **float** *cena*=2.5);

void *drobi* (**float** *v*=1.2, **float** *n*=3**)**;

kod (**int** *k1*, **int** *k2*=5);

§ 7.2. Опис функції користувача.

Опис функції складається з заголовка без крапки з комою і тіла функції, записаного в фігурних дужках і несе смислове навантаження:

<тип функції> <им’я функции> (<список формальних параметрів>)

{

<тіло функції>;

return (<им’я змінной>);

}

При описі функції в списку формальних параметрів обов'язково повинні бути імена змінних, навіть якщо в прототипі вони опускалися. У тілі функції записують команди, які задають дію функції. Результат виконання повертається в основну програму (в точку виклику) за допомогою змінної командою **return**. Тип змінної повинен збігатися з типом функції. У тілі

функцій з типом **void** команду **return** не пишуть. У команді **return** круглі дужки не обов'язкові**.**

Приклад:

```
float Summa (int kol, float cena)
{
   float s=kol*cena;
        return (s);
}
  void drobi (float v, float n)
{
   cout<<"\n drobi="<<v/n;
}
```

Якщо потрібно проиниціализировать значення змінних, які входять в функцію, то це можна зробити в сигнатурі функції. Тоді в описі у заголовка функції значення за замовчуванням не задають.

Приклад.

сигнатура **float** *perimetr* (**int** *k*=4, **float** *r*=2.5);

опис функції

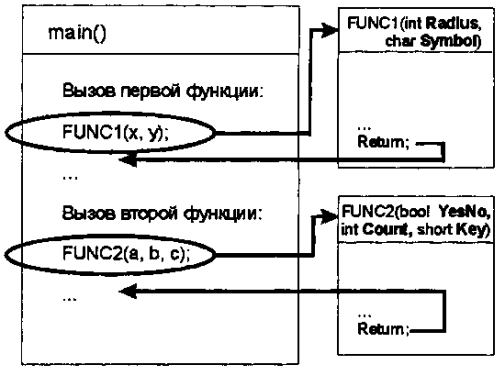
```
float perimetr (int k, float r)
{
        float p;
        p=k*r;
        return (p);
}
```

§ 7.3. Виклик функції користувача.

До функції користувача звертаються з розділу команд основної програми (функції **main** ()) або з іншої функції. Виклик функції можна записати двома способами: або просто командою виклику або викликати з виразів таким чином: <Ім'я функції> (<список фактичних параметрів>) Список фактичних параметрів може містити константи, змінні, посилання, покажчики, вирази. Списки формальних і фактичних параметрів повинні бу-

ти узгоджені за типами та кількістю елементів. Якщо в списку формальних параметрів є проініціалізувати змінні, то в списку фактичних параметрів ці змінні можуть бути відсутніми, їм будуть присвоєні значення за замовчуванням.

Будь-яка програма на С ++ обов'язково включає в себе головну функцію main (). Саме з цієї функції починається виконання програми. На малюнку показаний схематичний порядок виклику функцій.



Приклад.

Сигнатура

float *perimetr* (**int** k = 4, **float** r = 2.5);

до цієї функції можна звернутися одним із таких способів:

perimetr (7,2.8);

perimetr (7);

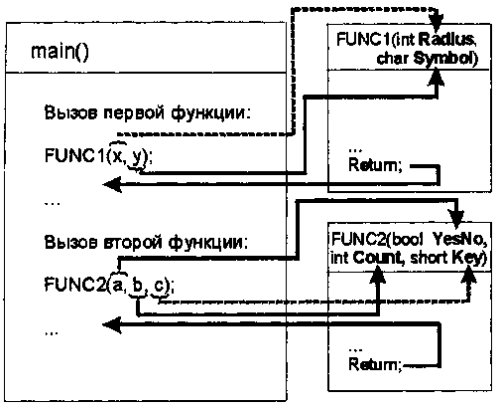
perimetr ();

У списку фактичних параметрів не можна пропускати змінні з середини списку. Тобто не можна написати *perimetr* (1.65); так як тут пропущена ініціалізація цілої змінної k. Розглянемо найпростіший приклад створення своїх функцій. Напишемо свою функцію, яку назвемо *Say*, і яка виводить напис *«Hello, world!»* на екран.

```
void Say()
  {
        cout<<"Hello, world!";
  }
```

У визначенні функції *Say ()* список параметрів порожній, тобто оголошень параметрів немає. Тіло функції складається з єдиною інструкції cout, яка виводить на екран вітання *"Hello, world!"*. У прикладі в заголовку функції *Say* () тип значення функції void. Цей тип застосовується тільки для функцій і для покажчиків, які ми будемо розглядати трохи пізніше; змінні типу void самостійно оголошувати не можна, компілятор відразу ж виведе помилку.

Якщо тип значення вказано як void (тобто «порожньо») це означає, що функція не повертатиме ніякого значення. У попередньому прикладі функція тільки виводила напис на екран і все, тобто ніяких значень вона повертати не повинна, тому її тип вказаний як void. В подальшому при створенні власних функцій необхідно уважно розібрати що функція буде робити і для чого вона призначена, визначити, чи буде функція повертати якесь значення і, якщо буде, то яке, і після цього тільки вказується тип значення функції. Наприклад, якщо раптом доведеться написати функцію користувача, яка обчислює косинус числа, то можна визначити, що функція косинуса повинна прийняти число з плаваючою точкою в списку параметрів, для цього числа знайти значення косинуса, яке теж буде числом з плаваючою точкою, і повернути це значення туди, звідки функція викликалася. Тобто для даного прикладу тип функції повинен бути **float** або **double,** так як необхідно буде повернути число з плаваючою крапкою. Синтаксично, параметри - це ідентифікатори (імена змінних) і вони можуть використовуватися всередині тіла функції. У списку формальних параметрів параметри вказуються через кому. Параметри в оголошенні і описі функції називають формальними параметрами. Тим самим підкреслюється їх сутність: формальні параметри - це те, замість чого будуть підставлені фактичні значення, передані функції в момент її виклику. Після виклику функції значення аргументу, відповідне формальному параметру і передане при виконанні функції, використовується в тілі виконуваної функції. У С ++ такі параметри є викликаються за значенням (call-by-value). Коли застосовується виклик за значенням, змінні передаються функції як аргументи, їх значення копіюються у відповідні параметри функції, а самі змінні не змінюються в зухвалій оточенні. По суті, викликані за значенням, параметри є локальними в своїй функції. Їм можуть передаватися вираження, значення яких присвоюються цим локальним змінним (параметрами).



Програма починає виконуватися з функції **main()** до виклику функції *FUNC1(x, y)*. З цього моменту управління програмою передається у функцію *FUNC1(x, y),* причому в якості значення змінної *Radius* ця функція використовує величину змінної x, а в якості змінної *Symbol* передається значення y (на малюнку проілюстрована передача параметрів у функції). Далі до оператора **return** виконується тіло функції FUNC1(x, y), після чого керування повертається в тіло функції **main(),** а саме, наступного за викликом *FUNC1(x, y)* оператору – управління повертається в те місце, звідки функція була викликана – викликає в оточення (*calling environment*).

Після цього продовжується виконання функції **main()** до виклику функції *FUNC2(a, b, з)*. При виклику цієї функції мінлива a передає значення логічної змінної *YesNo*, змінна b – цілочисельний змінної **Count**, а змінна – коротким цілого *Key*. Розглянемо невеликий приклад створення функції, яка приймає деякі параметри. Напишемо функцію, яка обчислює квадрат якого-небудь цілочисельного значення і повідомляє результат.

```
int square (int a)
{
    return a*a;
}
```

Як видно, функція *square()* приймає один цілочисельний параметр a, який буде передаватися значення з викличної функції, яке необхідно звести в квадрат. Тип повертаного значення для функції *square()* – **int**, тобто функція повинна буде повернути цілочисельне значення викликає функцію, що і робить інструкція **return** з параметром a, в якому зберігається результат роботи функції (квадрат значення параметра a).

Рядок **int main()** – це оголошення функції з ім'ям **main**, яка нічого не повертає і нічого не приймає в списку параметрів. А всі програми, які ми досі писали між фігурними дужками після рядка **int main(),** є не що інше, як тіло функції **main.**

Код програми повинен обов'язково перебувати в тілі якої-небудь функції (між фігурними дужками після заголовка функції), і не може «*висіти*» деінебудь поза функцій.

Ще одне суттєве зауваження: ***не можна визначати*** яку-небудь функцію в тілі іншої функції.

Програма на С++ складається з однієї або більше функцій, одна з яких – **main().**

Зазвичай функція, яка викликає якусь іншу функцію, називається *викликаною функцію*, а функція, яку *викликають на виконання*, називається *спричинюваною функцією*.

Для виклику функції необхідно вказати ім'я функції і список параметрів, які необхідно передати їй. Список переданих параметрів повинен бути укладений у круглі дужки.

Деякі функції повертають значення, як, наприклад, функція, вичисляє квадратний корінь від числа, повертає значення цього кореня. Якщо функція повертає значення, то потрібно прийняти це значення в якусь змінну (вивести на екран або використовувати в складеному вираженні). Наприклад, в бібліотеці *math.h* визначена функція мають квадратний корінь корінь, яка приймає як параметр значення, з якого слід витягти корінь, і повертає значення цього кореня. Як її можна використовувати

int *i*=2;

double *kor*1=***sqrt***(*i*); или**cout**<<***sqrt***(5);

Виклик функції є виразом, тому його можна використовувати як складову частину більш складного вираження.

Наприклад:

cout<<2****sqrt***(3)+1; або **float** *res*=***sqrt***(***sqrt***(7)/2.0)+10;

Викликати функцію можна тільки після оголошення функції (тобто нижче оголошення).

Оголошення функції стандартної бібліотеки виконується за допомогою підключення бібліотеки директиви препроцесора **#include.**

Існує кілька способів повернення управління до точки, з якої була викликана функція:

- Якщо функція не повинна повертати результат, керування повертається або просто при досягненні правої фігурної дужки, завершальній функцію, або при виконанні оператора **return.**

- Якщо функція повинна повертати результат, то оператор **return** вираз; повертає значення виразу в точку звернення до функції. Таким чином, оператор повернення має дві форми:
- **return;** – В цьому випадку, коли виконується оператор повернення, управління програмою негайно передається назад викликала в середу. Використовується коли функція не повертає значення (можна і без нього).
- **return** вираз; – В цьому випадку, в викликану середу повертається значення виразу, яке слід за ключовим словом **return**. Це значення повинно бути конвертованим до повертання типу з заголовка визначення функції.

Приклад:

return; не повертає значення;

return 3; обчислене значення=3;

return (a+b); обчислене значення=значення виразу (a+b); дужки необов'язкові, використовуються для поліпшення читаності коду.

Приклад 1

```
/* Програма для введення з клавіатури трьох чисел пропонує виконати наступні дії: перевірити на парність, обчислити добуток, обчислити середнє арифметичне непарних чисел */
#include < iostream.h >
// прототип функції
bool IsEven(int);                    /* функція перевіряє число на чіткість*/
int abcMultiple(int, int, int);  /* функція вичисляє суму трьох чисел */
float SrArith(int, int, int);      /* функція вичислює середнє
                                                    арифметичне непарних чисел*/
 // описання функції
void OutputMenu()    /* функція виводить на екран меню програми*/
{
cout<<"1. Ведення чисел \n"
```

```
                "2. Перевірка чисел на парність\n"
                "3. Обчислити добуток \n"
                "4.Обчислити середнє арифметичне не парних чисел \n";
cout<<"\n \n Виберіть пункт меню \n";
}

int InputNumber()       /* функція введення числа в заданім діапазоні */
{
int number=0;
do
{
cout<<"\n Введіть ціле число в діапазоні от 0 до 500: ";
cin>>number;
}
while(number<0 || number>500);
return number;
}

int main()
{
int a=0, b=0, c=0,              /* оголошення і ініціалізація змінних
                                цілого типу для збереження чисел,
                                введенних користувачем з клавіатури*/
menuVal=0;      /*оголошення и ініціализація змінной цілого
                типу для збереження значення вибраного пункта меню */
do
{
OutputMenu();                   /*виклик функції OutputMenu*/
cin>>menuVal;
switch(menuVal)
{
case 1:     /* Мітка відповідає задачі першого пункта меню */
```

```
a=InputNumber();
cout<<" \n a="<<a;
b=InputNumber();
cout<<" \n b="<<b;
c=InputNumber();
cout<<" \n c="<<c;
break;
case 2:      /* Мітка відповідає задачі другого пункта меню */
if(IsEven(a)) cout<<"\n a="<<a<<" - четное";
if(IsEven(b)) cout<<"\n b="<<b<<" - четное";
if(IsEven(c)) cout<<"\n c="<<c<<" - четное";
break;
case 3:      /* Мітка відповідає задачі третього пункта меню */
cout<<"\n Произведение  a*b*c="
<<abcMultiple(a, b, c);
break;
case 4:      /* Мітка відповідає задачі четвертого пункта меню */
cout<<"\n Середнє арифметичне не парних чисел="
                    << SrArith (a, b, c);
break;

default:
/* У виипаду, коли користувач  ввів значення за межами діапазону 1-4 */
cout<<"\n Будьте уважними. Є всього 4 пункта меню :)";
}
cout<<"\n\n Далі? "
"\n 0 - Нет, 1 - Так ";
cin>>menuVal;
}
while(menuVal);
}
// описання функції, прототип функції оголошені вищє
```

```
bool IsEven(int val)
{
bool valIsEven=val%2?  false : true;
return valIsEven;
}

int abcMultiple (int a, int b, int c)
{
        return a*b*c;
}
float SrArith(int val1, int val2, int val3)
{
int Sum=0, count=0;     /* оголошуємо допоміжні змінні для обчислення середнього арифметичного Sum/count */
if(!IsEven(val1))
{
count++;
Sum+=val1;
}
if(!IsEven(val2))
{
count++;
Sum+=val2;
}
if(!IsEven(val3))
{
count++;
Sum+=val3;
}
if(count) return float(Sum)/count;
else return 0;
}
```

Приклад 2.

```
//Сложення трьох цілих чисел – ілюстрація прототипів функції
#include <iostream.h>
int add3(int, int, int);
double sred(int);
void main()
{
int n1, n2, n3, sum;
cout<<"\nEnter three marks: ";
cin>>n1>>n2>>n3;
sum=add3(n1, n2, n3);
cout<<"\nSum= "<<sum;
cout<<"\n sred= "<<sred(sum);
sum=add3(1.5*n1, n2, 0.5*n3);
cout<<"\nWeight sum= "<<sum<<".";
cout<<"\nWeight sred= "<<sred(sum)<<"."<<"\n";
}
int add3(int a, int b, int c)
{
        return (a+b+c);
}
double sred(int s)
{
        return (s/3.0);
}
```

§ 7.4.1. Розбір програми

int *add3***(int, int, int) ;** **double** *sred***(int);**

Ці оголошення є прототипами функцій. Вони інформують компілятор про тип і кількість аргументів, передбачуваних для кожної оголошеної таким чином і визначеною в іншому місці функції.

sum=*add*3(1.5**n*1, *n*2, 0.5**n*3); – Виклик функції підсумовування трьох цілих параметрів add3, параметри в дужках будуть неявно перетворені до цілого типу.

```
int add3(int a, int b, int c)
    {
        return (a+b+c);
    }
```

Це власне визначення функції. Воно відповідає оголошення прототипу функції перед **main().** Так як список аргументів у прототипі функції може включати імена змінних, **int add3(int a, int b, int**); теж допустимо.

§ 7.5. Область видимості. Локальні і глобальні змінні

Змінні можуть бути оголошені як усередині тіла функції, так і за межами будь-якої з них. Змінні, оголошені усередині тіла функції, називаються локальними. Такі змінні розміщаються в стеку програми і діють лише всередині тієї функції, в якій оголошені. Як тільки управління повертається викликає функції, пам'ять, відведена під локальні змінні, звільняється. Кожна змінна характеризується

- ✓ областю дій,
- ✓ областю видимості
- ✓ часом життя.

Під областю дії змінної розуміють область програми, в якій змінна доступна для використання. З цим поняттям тісно пов'язані поняття області *видимості змінної.*

Якщо змінна виходить з області дії, вона стає невидимою. З іншого боку, змінна може перебувати в області дії, але бути невидимою.

Змінна знаходиться в області видимості, якщо до неї можна отримати доступ (за допомогою операції дозволу видимості, в тому випадку, якщо вона безпосередньо не видно).

Часом життя змінної називається інтервал виконання програми, протягом якого вона існує.

Локальні змінні мають своєю областю видимості функцію або блок, в яких вони оголошені. У той же час область дії локальної змінної може виключати внутрішній блок, якщо в ньому оголошена змінна з тим же ім'ям. Час життя локальної змінної визначається часом виконання блоку або функції, в якій вона оголошена.

Це означає, наприклад, що в різних функціях можуть використовуватися змінні з однаковими іменами абсолютно незалежно один від одного. У розглянутому прикладі змінні з ім'ям x визначені відразу в двох функціях - в *main()* і в *Sum(),* що, однак, не заважає компілятору розрізняти їх між собою: #**include** <*iostream.h*>

```
int Sum(int a, int b);
int main()
{
// Локальные переменные:
int x = 2;
int y = 4;
cout<<Sum(x, y) ;
}
int Sum(int a, int b)
{
// Локальна змінна x видна тільки в тілі функції Sum()
int x=a+b;
return x;
}
```

В програмі здійснюється обчислення суми двох цілочисельних змінних за допомогою виклику функції *Sum().*

Глобальні змінні, як вказувалося раніше, оголошуються поза тіла будь-якої з функцій і діють протягом виконання всієї програми.

Такі змінні доступні в будь-якої з функцій програми, яка описана після оголошення глобальної змінної.

Звідси випливає висновок, що імена локальних і глобальних змінних не повинні збігатися. Якщо глобальна змінна не проініціалізована явним чином, вона ініціалізується значенням 0.

Область дії глобальної змінної збігається з областю видимості і простягається від точки її опису до кінця файлу, в якому вона оголошена. Час життя глобальної змінної – *постійне*, тобто збігається з часом виконання програми.

Взагалі кажучи, на практиці програмісти намагаються уникати використання глобальних змінних і їх застосовують лише в разі крайньої необхідності, так як вміст таких змінних може бути змінено всередині тіла будь-якої функції, що загрожує серйозними помилками у роботі програми. Розглянемо приклад, що пояснює вищесказане:

Приклад 3.

```
#include <iostream.h>
// Оголошуємо глобальну змінну Test:
int Test=200;
void PrintTest(void);
int main()
{
// Оголошуємо локальну змінну Test:
int Test=10;
// Виклик функції печаті глобальної змінної:
PrintTest();
cout<<"Локальна: "<<Test<<'\n';
}
void PrintTest(void)
{
cout<<"Глобальна: "<<Test<<' \n';
```

```
}
```

Спочатку оголошується глобальна змінна *Test*, якій присвоюється значення 200. Далі оголошується локальна змінна з тим же ім'ям *Test*, але зі значенням 10. Виклик функції *PrintTest() main()* фактично здійснює тимчасовий вихід з тіла головної функції. При цьому всі локальні змінні стають недоступні і *PrintTest()* виводить на друк глобальну змінну *Test.* Після цього управління програмою повертається у функцію main(), де конструкцією *cout* виводиться на друк локальна змінна *Test*. Результат роботи програми виглядає наступним чином:

Глобальна: 200

Локальна: 10

В С++ допускається оголошувати локальну змінну не тільки на початку функції, а взагалі в будь-якому місці програми. Якщо оголошення відбувається всередині якого-небудь блоку, змінна з таким же ім'ям, оголошена поза тіла блоку, "ховається". Видозмінимо попередній приклад з тим, щоб продемонструвати процес приховування локальної змінної:

```
#include <iostream.h>
// Викликаємо глобальну змінну Test:
int Test = 200;
void PrintTest(void);
int main()
{
// Викликаємо локальну змінну Test:
int Test=10;
// Виклик функції печаті глобальної змінної:
PrintTest();
cout<<"Локальная: "<<Test<<'\n';
// Добавляем новий блок з ще одной локальной змінноїTest:
{
int Test = 5;
cout<<" Локальна: "<<Test<<' \n';
}
```

```
// Повертаємося до локальної Test поза блоку:
cout<<" Локальна: "<<Test<<'\n';
}
void PrintTest(void)
{
cout<<"Глобальна: "<<Test<<'\n';
}
```

Результат модифікованої програми буде виглядати наступним чином:

```
Глобальна: 200
Локальна: 10
Локальна: 5
Локальна: 10
```

§ 7.5.1. Операція ::

Як було показано вище, оголошення локальної змінної приховує глобальну змінну з таким же ім'ям. Таким чином, всі звернення до імені глобальної змінної в межах області дії локального оголошення викликають звернення до локальної змінної. Однак С++ дозволяє звертатися до глобальної змінної з будь-якого місця програми з допомогою використання операції дозволу області видимості. Для цього перед ім'ям змінної ставиться префікс у вигляді подвійного двокрапки (::):

```
#include <iostream.h>
// Объявление глобальной переменной
int Turn=5;
int main ()
{
// // Виклик функції глобальної змінної:
int Turn=70;
// Вивід локального значення:
cout<<Turn<<'\n';
// Вивід глобального значення:
cout<<::Turn<<'\n';
```

}

В результаті у два рядки буде виведено два значення: 5 і 70. З розглянутого прикладу видно, що були оголошені глобальна і локальна змінні з ім'ям *Turn*, які згодом були виведені на друк.

§ 7.5.2. Правила дій областей видимості.

Правила дії областей видимості визначають можливість отримання доступу до об'єкта і час його існування.

Областю видимості ідентифікатора назизається область програми, в якій на даний ідентифікатор можно посилатися.

На деякі ідентифікатори можна послатися в будь-якому місці програми, тоді як інші - лише в певних її частинах.

Існує чотири області видимості ідентифікатора - область видимості функції, область видимості файл, область видимості блок і область видимості прототип функції.

Ідентифікатор, оголошений поза будь-якої функції (на зовнішньому рівні), має **область видимості файл**. Такий ідентифікатор "відомий" всіх функцій від точки його оголошення до кінця файлу. Змінні, оголошення функцій і прототипи функцій, що знаходяться **поза** функції - всі мають областю видимості файл.

Змінні, викликані поза функції, називаються глобальними змінними.

Мітки (ідентифікатори з подальшим двокрапкою, наприклад, *start*: - єдині ідентифікатори, які мають областю видимості функцію. Мітки можна використовувати всюди в функції, в якій вони з'явилися, але на них не можна посилатися поза тіла функції. Мітки використовуються в структурах *switch* (як мітки *case)* і в операторів *goto*. Мітки ставляться до тих деталей реалізації, які функції "ховають" один від одного. Це приховування − один з найбільш фундаментальних принципів розробки хорошого програмного забезпечення.

Ідентифікатори, оголошені всередині блоку (на внутрішньому рівні) мають областю видимості блок. (Блок починається відкривається фігурною дужкою і завершується закриває). Область видимості блок почина-

ється оголошенням ідентифікатора і закінчується кінцевою правою фігурною дужкою блоку.

Змінні, які мають область видимості блок, називаються локальними змінними.

Змінні, оголошені в описах функцій, мають областю видимості блок так само, як і параметри функції, і є локальними змінними. Будь блок може містити оголошення змінних. Якщо блоки вкладені і ідентифікатор у зовнішньому блоці має таке ж ім'я, як ідентифікатор у внутрішньому блоці, ідентифікатор зовнішнього блоку "невидимий" (приховано) до моменту завершення роботи внутрішнього блоку. Це означає, що поки виконується внутрішній блок, він бачить значення своїх власних локальних ідентифікаторів, а не значення ідентифікаторів з ідентичними іменами охоплює блоці.

Розглянемо це на прикладі.

```
{                                   //зовнішній блок
int a=2,                            //виклик і ініціалізація змінной a
cout<<a<<'\n';                      // виводить на екран 2
{                                   // вхід в внутрішній блок
  int a=7,                          // змінна a із внутрішнього блоку
  int s=a;                          //виклик і ініціалізація змінной s
  cout<<"s="<<s;                    //виводить на екран s=7
  cout<<"a="<<a<<'\n';              //виводит на екран a=7
}                                   //вихід із внутрішній блок
        cout<<++a<<'\n';            // виводить на екран 3
        cout<<s<<'\n';              //помилка компіляції: змінна s не
}                                   // викликана, вихід із внутрішнього блока
```

Внутрішні блоки можуть бути вкладені на довільну глибину, визначену обмеженнями системи. Єдиними ідентифікаторами з областю видимості прототип функції є ті, які використовуються в списку параметрів прототи-

пу функції. Прототипи функцій не вимагають імен у списку параметрів − потрібні тільки типи. Якщо в списку параметрів прототипу функції використовується ім'я, компілятор це ім'я ігнорує.

Ідентифікатори, що використовуються в прототипі функції, можна повторно використовувати де завгодно в програмі, не побоюючись двозначності. Змінну можна оголосити у розділі ініціалізації циклу for або умовному вираженні інструкцій *if, switch* або *while*.

Приклад.

```
int main ()
{
for (int i=0; i<10; i++)
{cout<<i<<" ";
cout<<" kvadrat ="<<i*i<<'\n';
}
i=10                                    // помилка! і – невідома
}
```

Приклад 2.

```
int main()
{
int choice;
cout<<"(1) сложить числа";
cout<<"(2) конкатеніровать строки";
cin >> choice;
if (choice==1)
{
int a,b;
cout<<"Введіть два числа:";
cin >> a >> b;
cout<<"Сума рівна"<<a+b<<'\n';
}
else
{char s1[80], s2[80];
```

```
cout<<"Введіть две строки";
cin>>s1;
cin>>s2;
strcat(s1,s2);
cout<<"Конкатенація рівна"<<s1<<'\n'
}
}
```

§ 7.6. Новий стиль заголовків

Історично так склалося, що в мові С++ при підключенні заголовних файлів використовувався той же синтаксис, що і в мові С для сумісності з розробленим на той момент програмним забезпеченням. Однак при стандартизації мови цей стиль був змінений і тепер замість заголовних файлів (як це було З) вказуються деякі стандартні ідентифікатори, за яким компілятор сам знаходить необхідні файли. Системні ідентифікатори являють собою ім'я заголовка в кутових дужках без вказівки розширення (.h). Нижче наводиться приклад включення заголовків у стилі С++:

```
#include <iostream>
#include <stdlib>
#include <new>
```

Крім цього, для включення в програму бібліотек функцій мови у відповідність з новим стандартом заголовок перетворюється наступним чином: відкидається розширення **.h** і до імені заголовка додається с. Таким чином, наприклад, заголовок <*string.h*> замінюється заголовком <*cstring*>. Якщо ж використовується компілятор не підтримує оголошення заголовків у новому стилі, можна використовувати заголовки в стилі мови, хоча це і не рекомендується стандартом С++.

§ 7.7. Простір імен

Визначення функцій і змінних в заголовних файлах нерозривно пов'язані з поняттям простору імен. Це поняття з'явилося порівняно недавно. До введення поняття простору всі оголошення ідентифікаторів і констант, зроблені в заголовочном файлі, містилися компілятором в глобальний простір імен. Таке становище призводило до виникнення маси конфліктів, пов'язаних з використанням різними об'єктами однакових імен. Найчастіше непорозуміння виникали, коли в одній програмі використовувалися бібліотеки, розроблені різними виробниками. Введення поняття простору імен дозволило значно знизити кількість подібних конфліктів імен. Коли в програму включається заголовок нового стилю, його вміст не поміщається в глобальний простір імен, а в простір імен *std* Якщо в програмі потрібно визначити деякі ідентифікатори, які можуть замінити вже наявні, просто треба завести свій власний, новий простір імен. Це досягається шляхом використання ключового слова *namespace:*

namespace *им'я_простіру_імен*
{
// оголошення
}

Таким чином, оголошення в межах нового простору імен будуть знаходитися тільки в межах видимості *певного имені_простіру_імен,* запобігаючи тим самим виникнення конфліктів. В якості прикладу створимо наступне простір імен:

namespace *NewNameSpace*
{
int *x*, *y*, *z*;
void *SomeFunction*(**char** *smb*);
}

Для того, щоб вказати компілятору, що слід використовувати імена з конкретного іменного простору (в даному випадку з *NewNameSpace*), можна скористатися операцією дозволу видимості:

NewNameSpace::*x*=5;

Однак, якщо в програмі звернення до власного простору імен проводяться досить часто, такий синтаксис викликає певні незручності. В якості альтернативи можна скористатися інструкцією using, синтаксис якої має дві форми:

using namespace *им'я_простіру_імен*;

або

using *им'я_простіру_імен*::*ідентифікатор*;

При використанні першої форми компілятору повідомляється, що в подальшому необхідно використовувати ідентифікатори із зазначеного іменного простору аж до того моменту, поки не зустрінеться наступна інструкція using. Наприклад, вказавши в тілі програми

using namespace *NewNameSpace* ;

можна безпосередньо працювати з відповідними ідентифікаторами:

x=0; *y*=*z*=4;

SomePunction('A') ;

На практиці часто після включення в програму заголовків явно вказується використання ідентифікаторів стандартного простору імен:

using namespace *std*;

Друга форма запису наказує компілятору використовувати зазначене простір імен лише для конкретного ідентифікатора. Таким чином, визначивши

using namespace *std*;

using *NewNameSpace*::*z*;

можна використовувати ідентифікатори стандартної бібліотеки С++ і цілочисельну змінну *z* з простору імен *NewNameSpace* без використання операції дозволу видимості: *z=12*;

Слід розуміти, що вказівка нового простору імен інструкцією using namespace скасовує видимість стандартного простору *std*, тому для отримання доступу до відповідних ідентифікаторів з std потрібно кожен раз використовувати операцію дозволу видимості std::. Простору імен не мо-

жуть бути оголошені усередині тіла функції, однак можуть оголошуватися всередині інших просторів. При цьому для доступу до ідентифікатора внутрішнього простору необхідно вказати імена всіх вищих іменованих просторів. Наприклад, оголошено наступне простір імен:

```
namespace Highest
{
namespace Middle
{
namespace Lowest
{
int nAttr;
}
}
}
```

Використання оголошеної змінної *nAttr* буде виглядати: *Highest::Middle::Lowest::nAttr=0;*

§ 7.8. Математичні функції

Прототипи стандартних математичних функцій визначені у заголовочном файлі *math.h*. Розглянемо деякі з них, найбільш часто вживані в повсякденній роботі. Наприклад, функція *pow(),* що дозволяє зводити число до степеня. Синтаксис цієї функції виглядає наступним чином: *double pow (double x, double);* Таким чином, компілятор повідомляється, що необхідно число подвійної точності x звести до степеня числа подвійної точності. До цієї категорії також належать логарифмічні функції та функція добування кореня числа:

```
double log(double);                 // натуральні логарифми
float logf(float);
long double logl(long double);
double log10(double);               // десяткові логарифми
float log10f(float);
long double log10l(long double);
```

double *sqrt*(double); // корінь числа
float *sqrtf*(float);
long double *sqrtl*(long double);
Друга більша група - функції получення абсолютної величини числа,
int *abs*(int); // цілі
double *fabs*(double); // двійної точності
long *labs*(long); // довгі
float *fabsf*(float); // з плаваючею точкою
long double *fabsl*(long double); // довгі двійної точності
сприймають у якості параметра аргумент деякого типу (свій для кожної з функцій) і повертають його беззнаковую форму.
Для обчислення залишку від ділення числа x на y використовується функція *fmod(),* яка має наступний синтаксис: *double fmod(double x, double);*
Стандартна бібліотека має широким набором тригонометричних функцій і їх модифікацій для різних типів аргументів:
double *acos*(double); // Арккосинус
float *acosf*(float);
double *asin*(double); // Арксинус
float *asinf*(float);
double *atan*(double); // Арктангенс
float *atanf*(float);
double *atan2*(double x, **double** y); // Арктангенс отношения y/x
float *atan2f*(float, float);
double *cos*(double); // Косинус
float *cosf*(float);
double *cosh*(double); // Гиперболический косинус
float *coshf*(float);
double *sin*(double); // Синус
float *sinf*(float);
double *sinh*(double); // Гиперболический синус
float *sinhf*(float);
double *tan*(double); // Тангенс

float ***tanf***(**float**);
double ***tanh***(**double**) ; // Гіперболічний тангенс
float ***tanhf***(**float**);

Слід відзначити, що кути тригонометричних функцій зазначаються в радіанах. Нижче наводиться приклад, що здійснює переклад градусів в радіани і виведення значення синуса для введеного в градусах числа.

```
#include <iostream.h>
#include <math.h>
int main()
{
double Angle;
double PI=3.14159;
cout<<"Введіть угол в градусах: ";
cin>>Angle;
cout<<"Значення синуса: ";
cout<<sin(Angle*PI/180)<<'\n';
}
```

На жаль, в С++ немає готової реалізації функції зведення аргументу в квадрат. Як її реалізувати ми розглядали в прикладі вище.

§ 7.9. Функції округлення

Часто потрібно скористатися округленим значенням тієї чи іншої змінної. С++ пропонує набір функцій для рішення цієї задачі. Залежно від конкретної ситуації може знадобитися функція, округляюча значення аргументу в більшу або меншу сторону. Розглянемо найбільш часто використовувані варіанти викликів.

Округлення числа в меншу сторону використовується функція *floor()* та її різновиди для різних типів аргументів і повертаються параметрів. Ця функція має наступний синтаксис:

double ***floor***(**double** *x*);
long double ***floorl***(**long double** *x*);

Округлення у більшу сторону проводиться за допомогою функції *ceil():*

double ***ceil***(**double** *x*);

long double ***ceil***(**long double** *x*);

Проте в реальності проблема вибору в яку ж сторону проводити округлення, покладається на розроблювану програму.

§ 7.10. Посилання.

До даних можна звернутися за допомогою імен або посилань. Посилання служить для присвоєння ще одного імені (псевдоніма, синонімів, псевдонімів) даним. Посилання створюють так:

<тип даного> &<ім'я посилання>=<ім'я змінної>;

Приклад. *float &cena=summa*; – у цьому випадку посилання *cena* і мінлива *summa* будуть вказувати на один і той же адресу в пам'яті комп'ютера. Для посилань не резервується додаткова оперативна пам'ять. В С++ можна створювати посилання на дані, але не на їх типи. Значення посилання ініціалізують відразу під час її оголошення, тобто на етапі компіляції. У наведеному прикладі посилання cena проініціалізована змінної *summa.* Таким чином, якщо *summa*=11, то і значення посилання *cena* теж буде 11. Під час зміни значення посилання змінюється значення змінної, на яку це посилання. Таким чином, якщо в програмі записати команду *cena*=15.7, то змінної summa теж буде присвоєно значення 15.7.

Змінювати (переадресування) посилання в програмі не можна. Посилання завжди вказує на один і той же адресу в оперативній пам'яті. Це використовують під час створення і виклику функцій. Під час виклику функції копії всіх її фактичних параметрів заносяться в спеціально організовану область пам'яті. Потім виконуються відповідні команди функції і результат повертається в програму командою **return.** Оскільки всі дії виконуються з копіями параметрів, а не з самими параметрами (копії і власне параметри розміщені в різних ділянках пам'яті), то значення фактичних параметрів в основній програмі не змінюються. Як параметри функції можна використовувати посилання або покажчики. Тоді значення фактичних параметрів в основній програмі будуть змінюватися, так як функція буде повертати значення в основну програму не тільки через змінну з команди **return,** а

також через відповідні посилання і покажчики, так як вони вказують на той же самий ділянку пам'яті, де розміщені фактичні параметри. Щоб передати посилання або покажчики в функцію і не змінити значення фактичних параметрів, потрібно в оголошенні функції до кожного параметру дописати ключове слово **const.**

Приклад: **int** sort (*const* int *p, *const* **int** *g);

В С++ посиланням може бути не тільки змінна або константа, а й функція <тип> &<ім'я функції>(<список формальних параметрів>)

Така функція повертає синонім імені комірки, в яку занесено результат (посилання на змінну певного типу). Функція-посилання має двояке призначення. По-перше, як і звичайна функція, вона може повертати значення в основну програму. По-друге, їй самій можна присвоювати значення, що є унікальним випадком в програмуванні.

Приклад.

```
float *prt, u;        // Оголошуємо змінну і покажчик на дійсний тип
float &Item(float *a, int i)          // Оголошення функції-покажчика
{
return *(a+i);
}
prt=new float[10];      // Виділяємо ділянка пам'яті для зберігання
// значень десяти дійсних чисел
u=Item(prt, 3);          // Викликати цю функцію можна зазвичай// Змінної u буде присвоєно значення
// четвертого елемента.
//Введемо значення п’ятого числа так
Item(prt, 4)=10;          // У цьому випадку функції Item() присвоюємо значення,// тобто ділянка пам'яті буде внесено число 10.
```

§ 7.11. Передача параметрів в функцію.

У багатьох мовах програмування є два способи передачі параметрів у функцію за значенням та за посиланням. Коли параметр передається за значенням, створюється його копія і вона передається викликається функції. Зміни копії не впливають на значення оригіналу.

При передачі аргументів за значенням компілятор створює тимчасову копію об'єкта, який повинен бути переданий, і розміщує її в області стекової пам'яті, призначеної для зберігання локальних об'єктів. Викликається функція оперує саме з цією копією, не надаючи впливу на оригінал об'єкта. Прототипи функцій, що приймають аргументи за значенням, передбачають в якості параметрів вказівка типу об'єкта, а не його адреси. Наприклад, функція int GetMax(int, int); приймає два цілочисельних аргументів за значенням.

Якщо ж необхідно, щоб функція модифікувала оригінал об'єкта, використовується передача параметрів за посиланням. При цьому в функцію передається не сам об'єкт, а лише його адреса. Таким чином, всі модифікації в тілі функції переданих їй за посиланням аргументів впливають на об'єкт. Беручи до уваги той факт, що функція може повертати лише єдине значення, використання передачі адреси об'єкта виявляється дуже ефективним способом роботи з великою кількістю даних. Крім того, так як передається адреса, а не сам об'єкт, істотно економиться стекова пам'ять.

В С++ передача за посиланням може здійснюватися двома способами:

- ✓ використовуються безпосередньо посилання;
- ✓ за допомогою покажчиків.

Синтаксис передачі з використанням посилань передбачає застосування в якості аргументу посилання на тип об'єкту. Наприклад, функція **double**

Glue(**float**& *var*1, **int**& *var*2);

отримує два посилання на змінні типу float і int. При передачі у функцію параметра-посилання компілятор автоматично передає в функцію адресу змінної, зазначеної в якості аргументу. Ставити знак амперсанда перед аргументом у викликах функції не потрібно. Наприклад, для попередньої

функції виклик з передачею параметрів за посиланням виглядає наступним чином: Glue(var1, var2);

Прототип функції при передачі параметрів через покажчик:

void SetNumber(int*, float *);

Крім того, функції можуть повертати не тільки значення деякої змінної, але і покажчик або посилання на нього. Наприклад, функції, прототип яких:

*int Count(int);

&int Increase();

повертають покажчик та посилання відповідно на цілочисельну змінну типу int. Слід мати на увазі, що повернення посилання або вказівника зі функції може призвести до проблем, якщо змінна, на яку робиться посилання, вийшла з області видимості.

Приклад 6.

```
int Inc(int s) // передача параметра за значенням
{
s=s+1; // значення змінної s=1
return s;
}
void main()
{
int a=0;
cout<<a; // значення змінної a до виклику функції Inc, a=0
int b=Inc(a); // значення змінної b=1
cout <<a; // значення змінної a після виклику функції Inc,a=0
}
```

У цьому прикладі у функції *main* оголошується змінна a, яка потім передається у функцію *Inc* за значенням, тобто передається копія. Зміни, яким піддається змінна s у функції Inc не впливають на значення змінної a у функції *main*. Після повернення з функції *Inc* у функцію *main*, мінлива a як і до виклику функції *Inc* буде мати значення 0.

Один з недоліків передачі параметрів за значенням полягає в тому, що якщо передається великий елемент даних, створення копії цих даних може призвести до значних втрат часу виконання.

Інший спосіб передачі параметрів у функцію – передача параметрів за посиланням. У разі передачі параметрів за посиланням викликається функція отримує можливість прямого доступу до передаваних даних, а значить можливість зміни цих даних. Копії цих даних не створюються.

Приклад 7.

```
void swap(int &x, int &y);
int main ()
{
int i, j;
i=10; j=20;
cout<<"i="<<i<<' '<<"j="<<j<<'\n';
swap (i, j);
cout<<"i="<<i<<' '<<"j="<<j<<'\n';
}
void swap(int &x, int &y)
{
int temp;
temp=x;
x=y;
y=temp;
}
```

§ 7.12. Виклик функцій з масивами.

Якщо масив є аргументом функції, то при виклику такої функції їй передається тільки адресу першого елемента масиву, а не повна його копія. Це означає, що оголошення формального параметра повинне мати тип, сумісний з типом аргументу.

Існує три способи оголосити формальний параметр, який приймає покажчик на масив:

1) Параметр можна оголосити як масив, тип і розмір якого збігається з типом і розміром масиву, що використовується при виклику функції.

Приклад 4.

```
void display(int num[10]);
int main ()
{
int t[10];
int i;
for (i=0; i<10; i++)
t[i]=i;
display (t);
}
void display (int num [10])
{
int i;
for (i=0; i<10; i++)
cout<<num [i]<<' ';
}
```

2) Параметр-масив можна оголосити як безрозмірний масив:

```
void display (int num [])
{
int i;
for (i=0; i<10; i++) cout<<num [i]<<' ';
}
```

3) При передачі масиву функції її параметр можна оголосити як дороговказ:

```
void display (int *num)
{
int i;
for (i=0; i<10; i++) cout<<num [i]<<' ';
```

```
}
```

Зауваження 1. Окремий елемент масиву, що використовується в якості аргументу, обробляється подібно до звичайної змінної.

Зауваження 2. Якщо масив використовується в якості аргументу функції, то функції передається адреса цього масиву. Це означає, що код функції може змінити реальний вміст масиву.

Приклад 5.

```
void cube (int *n, int kol);
int main()
{
int nums[10];
int i;
for (i=0; i<10; i++) nums[i]=i+1;
for (i=0; i<10; i++) cout<<nums[i]<<' ';
cout<<'\n';
cube (nums, 10);
for (i=0; i<10; i++) cout<<nums[i]<<' ';
}
void cube (int *n, int kol)
{
while (kol)
{
*n=*n**n**n;
kol--;
n++;
}
}
```

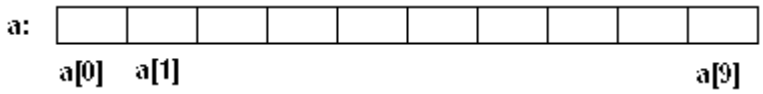
Тема 8. Багатовимірні масиви. Динамічні масиви

§ 8.1. Вказівники та масиви.

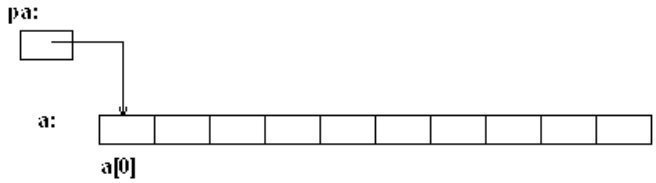
В С ++ існує тісний зв'язок між вказівниками і масивами. Будь-який доступ до елементу масиву, здійснюваний операцією індексування, може бути виконаний за допомогою вказівника.

Наприклад, коли оголошується масив у вигляді **int** a [25], то при цьому не тільки виділяється пам'ять для 25 елементів масиву, але і формується вказівник з ім'ям a, значення якого дорівнює адресі першого за рахунком (нульового) елемента масиву. Доступ до елементів масиву може здійснюватися через вказівник з ім'ям a. З точки зору синтаксису мови вказівник a є константою, значення якої можна використовувати у виразах, але змінити це значення не можна.

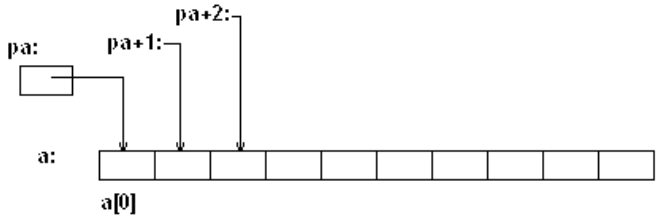
Оголошення **int** a [10]; визначає масив a розміру 10, тобто блок з 10 послідовних об'єктів з іменами a [0], a [1], ..., a [9].



Запис a [i] відсилає нас до i-му елементу масиву. Якщо pa є вказівник на **int**, тобто визначений як int * pa; то в результаті присвоювання pa = & a [0]; pa буде вказувати на нульовий елемент a; інакше кажучи, pa буде містити адресу елемента a [0].



Тепер присвоювання x = * pa; буде копіювати зміст a [0] в x. Якщо pa вказує на певний елемент масиву, то pa + 1 по визначенню вказує на наступний елемент, pa + i - на i-ий елемент після pa, а pa-i - на i-ий елемент перед pa. Таким чином, якщо pa вказує на a [0], то * (pa + 1) є зміст a [1], pa + i - адреса a [i], a * (pa + i) - зміст a [i].



Зроблені зауваження вірні до типу і розміру елементів масиву a. Сенс слів "додати 1 до вказівника", як і сенс будь-якої арифметики з вказівниками, в тому, щоб pa + 1 вказував на наступний об'єкт, а pa + i - на i-й після pa.

Між індексуванням і арифметикою з вказівниками існує дуже тісний зв'язок. За визначенням ім'я масива - це адреса його нульового елемента. Після присвоювання pa = & a [0]; pa і a мають одне і те ж значення. Оскільки ім'я масиву є не що інше, як адреса його початкового елемента, присвоювання pa = & a [0]; можна також записати в наступному вигляді: pa = a;

Крім того, a [i] можна записати як * (a + i). Зустрічаючи запис a [i], компілятор відразу перетворює її в * (a + i); зазначені дві форми запису еквівалентні. З цього випливає, що, отримані в результаті застосування оператора & записи & a [i] і a + i, також будуть еквівалентні, тобто і в тому, і в іншому випадку це адреса i-го елемента після a. З іншого боку, якщо pa - вказівник, то у виразах його можна використовувати з індексом, тобто запис pa [i] еквівалентна запису * (pa + i). Елемент масиву однаково дозволяється зображати і у вигляді вказівника зі зміщенням, і у вигляді імені масиву з індексом.

Між ім'ям масиву і вказівником, виступаючим в ролі імені масиву, існує одна відмінність. Вказівник - це змінна, тому можна написати pa = a чи pa ++. Але ім'я масиву є константою, і записи типу a = pa або a ++ не допускаються.

Відзначимо різницю в виразах: * pa + 1 і * (pa + 1).

Операція розіменування має більш високий пріоритет, ніж операція додавання. Тому перший вираз спочатку розіменовує змінну pa і отримує перший елемент масиву, а потім додає до нього 1. Другий вираз доставляє значення другого елементу.

Масиви не самодостатні в тому сенсі, що не гарантовано зберігання інформації про кількість елементів разом з самим масивом. У більшості реалізацій С ++ відсутня перевірка діапазону індексів для масивів. Такий традиційний низькорівневий підхід до масивів. Більш повне уявлення масиву можна реалізувати за допомогою класів.

В С ++ масиви тісно пов'язані з вказівниками. Ім'я масиву можна використовувати в якості вказівника на його перший елемент. Гарантується осмисленість значення вказівника на елемент, що настає за останнім елементом масиву. Це важливо для багатьох алгоритмів. Але з огляду на те, що такий вказівник насправді не вказує ні на який елемент масиву, його не можна використовувати ні для читання, ні для запису. Результат отримання адреси елемента масиву, що передує першому, не визначений, і такої операції слід уникати.

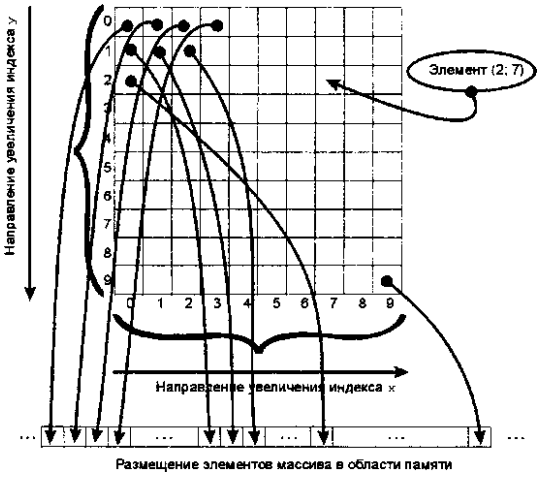
Неявне перетворення імені масиву в вказівник на його перший елемент широко використовується у викликах функцій.

§ 8.2. Багатовимірні масиви

Багатовимірні масиви – це масиви з більш ніж одним індексом.

Частіше всього використовуються двовимірні масиви.

При описі багатовимірного масива треба вказати С++, що масив матє більш ніж один вимір.



Багатовимірний масив розмірності N можні представити як одномірний масив з масивів розмірності (N–1). Таким чином, наприклад трьохвимірний масив – це масив, кожний елемент якого представляє собою двохвимірну матрицю.

Приклади об'явлення багатовимірних масивів:

// Двохвимірний масив 6×9 елементів:

char Matrix2D[6][9];

```
// Тривимірний:
        unsigned long Arr3D[4][2][8];
// Масив 7-ї степені вимірності:
        my_type Heaven[22][16][7][47][345][91][3];
```

Вираз Array [idx] [idy], що представляє двовимірний масив, перекладається компілятором в еквівалентний вираз: * (* (Array + idx) + idy).

Тобто, багатовимірні масиви в мові С - це масиви масивів, тобто масиви, елементами яких, в свою чергу, є масиви. При оголошенні таких масивів в пам'яті комп'ютера створюється кілька різних об'єктів. Наприклад, при виконанні оголошення двовимірного масиву int a2 [4] [3] в програмі створюється покажчик a2, який визначає в пам'яті розташування першого елемента масиву і, крім того, є вказівником на масив з чотирьох вказівників. Кожен з цих чотирьох вказівників містить адресу одновимірного масиву, що представляє собою рядок двовимірного масиву і складається з трьох елементів типу int, і дозволяє звернутися до відповідної рядку масиву.

Таким чином, оголошення a2 [4] [3] породжує в програмі три різних об'єкта: вказіник з ідентифікатором a2, безіменний масив з чотирьох вказівників і безіменний масив з дванадцяти чисел типу int. Для доступу до безіменним масивів використовуються адресні вирази з покажчиком a2. Доступ до елементів масиву вказівників здійснюється із зазначенням одного індексного виразу в формі a2 [2] або * (a2 + 2). Для доступу до елементів двовимірного масиву чисел типу int повинні бути використані два індексних вирази в формі a2 [1] [2] або еквівалентних їй * (* (a2 + 1) +2) і (* (a2 + 1)) [2] . Слід враховувати, що з точки зору синтаксису мови С покажчик a2 і покажчики a2 [0], a2 [1], a2 [2], a2 [3] є константами, і їх значення не можна змінювати під час виконання програми.

Розміщення тривимірного масиву відбувається аналогічно. Так, наприклад, оголошення float a3 [3] [4] [5] породжує в програмі, крім самого тривимірного масиву з 60 чисел типу float, масив з чотирьох покажчиків на тип float, масив з трьох покажчиків на масив покажчиків на float і покажчик на масив масивів покажчиків на float.

При розміщенні елементів багатовимірних масивів вони розташовуються в пам'яті поспіль по рядках, тобто швидше за все змінюється останній індекс, а повільніше - перший. Такий порядок дає можливість звертатися до будь-якого елементу багатовимірного масиву, використовуючи адресу його початкового елемента і тільки одне індексний вираз.

Наприклад, звернення до елементу a2 [1] [2] можна здійснити за допомогою покажчика ptr2, оголошеного в формі int * ptr2 = a2 [0], як звернення ptr2 [1 × 3 + 2] (тут 1 і 2 - це індекси використовуваного елемента, а 3 - число елементів в рядку) або як ptr2 [5]. Зауважимо, що зовні схоже звернення a2 [6] виконати неможливо, так як покажчика з індексом 6 не існує.

Для звернення до елемента a3 [2] [3] [4] з тривимірного масиву теж можна використовувати покажчик, описаний як float * ptr3 = a3 [0] [0], з одним індексним виразом у формі ptr3 [2 × 20 + 3 × 5 + 4] або ptr3 [59].

Багатовимірні масиви, як і одномірні, можуть бути ініційовані на етапі оголошення:

```
    int ia [4] [3] = {
{0, 1, 2},
{3, 4, 5},
{6, 7, 8},
{9, 10, 11}
     };
```

Внутрішні фігурні дужки, що розбивають список значень на рядки, необов'язкові і використовуються, як правило, для зручності читання коду. Наведена далі ініціалізація в точності відповідає попередньому прикладу, хоча менш зрозуміла: int ia [4] [3] = {0,1,2,3,4,5,6,7,8,9,10,11};

Наступне визначення инициализирует тільки перші елементи кожного рядка. Решта елементи будуть дорівнюють нулю: int ia [4] [3] = {{0}, {3}, {6}, {9}};

Якщо ж опустити внутрішні фігурні дужки, результат виявиться зовсім іншим. Всі три елементи першого рядка і перший елемент другої отримають вказане значення, а інші будуть неявно ініціалізовані 0.

int ia [4] [3] = {0, 3, 6, 9};

Доступ до елементів багатовимірного масиву через вказівники здійснюється трохи складніше. Оскільки, наприклад, двовимірний масив Matrix [x] [y] може бути представлений як одновимірний (Matrix [x]), кожен елемент якого також є одновимірним масивом (Matrix [y]), покажчик на двовимірний масив pMtrx, посилаючись на елемент масиву Matrix [x] [y], по суті, вказує на масив Matrix [y] в масиві Matrix [x]. Таким чином, для доступу до вмісту осередку покажчик pMtrx доведеться разименовивать двічі.

Важливо знати, що в багатовимірних масивах потрібен якийсь час на обчислення кожного індексу. Це означає, що доступ до елементу в багатовимірних масивах відбувається повільніше, ніж доступ в одновимірних масивах. З цієї та інших причин, якщо виникає необхідність в багатовимірних масивах, для них найчастіше пам'ять виділяється динамічно з використанням функції динамічного виділення пам'яті.

§ 8.3 Динамічне виділення масивів

У програмі кожна змінна може розміщуватися в одному з трьох місць: в області даних програми, в стеці або у вільній пам'яті (так звана купа).
Кожній змінній в програмі пам'ять може відводитися або статично, тобто в момент завантаження програми, або динамічно - в процесі виконання програми. До сих пір всі обумовлені масиви оголошувалися статично, і, отже, зберігали значення всіх своїх елементів в стековой пам'яті або області даних програми. Якщо кількість елементів масиву невелика, таке розміщення виправдано. Однак досить часто виникають випадки, коли в стековой пам'яті, що містить локальні змінні і допоміжну інформацію (наприклад, точки повернення з вкладених функцій), недостатньо місця для розміщення всіх елементів великого масиву. Ситуація ще більш ускладнюється, якщо масивів великого розміру має бути багато. Тут на допомогу приходить можливість використання для зберігання даних динамічної пам'яті.

§ 8.4 Оператори вільної пам'яті new і delete

Всі змінні, які ми розглядали дотепер, статичні. Під час їх оголошення система автоматично надає для зберігання їх значень певний обсяг оперативної пам'яті, і цей розподіл пам'яті залишається незмінним протягом виконання програми. Пам'ять, надана цим змінним, резервується (фіксується) всередині ***exe-файлу*** скомпільована. Навіть якщо не всі змінні будуть використані в програмі, пам'ять буде зарезервована, тобто буде використана неефективно. Пам'ять, надана таким змінним, вивільняється лише після виконання програми. Тому в оперативній пам'яті можна розмістити лише обмежена кількість даних.

Однак є завдання, де заздалегідь невідомо, скільки змінних потрібно для їх вирішення, а, отже, який обсяг пам'яті потрібно зарезервувати, або завдання, в яких, навпаки, відразу відомо, що змінних буде багато, наприклад, завдання обробки масивів великих розмірів. У таких випадках застосовують динамічну організацію пам'яті.

Принцип динамічної організації пам'яті полягає в тому, що для змінних пам'ять виділяється в разі потреби (за вказівкою програміста). Далі ці змінні обробляють і в потрібний момент пам'ять вивільняють (знову за вказівкою програміста). Такі змінні називаються ***динамічними.***

Унарні оператори **new** і **delete** служать для керування динамічною (вільною) пам'яттю. Вільна пам'ять - це надана системою область пам'яті для об'єктів, час життя яких безпосередньо управляється програмістом. Програміст створює об'єкт за допомогою ключового слова **new** (перекладається з англійської як " новий "), а знищує його, використовуючи delete (перекладається з англійської як " видалити ").

Для роботи з динамічними змінними використовують покажчики. Для виділення динамічної пам'яті застосовується команда **new**. В С ++ оператор new приймає такі форми: **new і'мя_типу**;

new ім'я_типу (*ініціалізатор***);**

new ім'я_типу [*вираз***];**

У кожному разі відбувається, щонайменше, два ефекта. По-перше, виділяється належний обсяг вільної пам'яті для зберігання зазначеного ти-

пу. По-друге, повертається базовий адреса об'єкта (як значення оператора **new**). Коли пам'ять недоступна, оператор new повертає значення 0 (**NULL**). Отже, можна контролювати процес успішного виділення пам'яті оператором **new**.

Дії команди ***new.*** Для відповідного типу змінній автоматично надається необхідна безперервна ділянка пам'яті. Команда **new** повертає обсяг цієї ділянки, а вказівник вказує на її початок. Наприклад, щоб зарезервувати в пам'яті комп'ютера область для зберігання значення цілого типу, застосовують таку команду: **int** **prt*=**new int**;.

Надати ділянку пам'яті і відразу занести в неї значення можна так:

float **prt*2=**new float**(3.14);.

На адресу, на який вказує *prt*2, буде занесено число 3,14.

З динамічної змінної можна виконати операції, визначені для даних відповідного базового типу.

Розглянемо наступний приклад використання оператору **new**:

```
int *p, *q;
p=new int (5);          //виділили пам'ять і ініціалізували
q=new int [10];         //отримуємо масив від q[0] до q[9]
```

У цьому фрагменті вказівнику на ціле *p* присвоюється адреса комірки пам'яті, отримана при розміщенні цілого об'єкта. Місце в пам'яті, на яке вказує *p*, инициализируется значенням 5. Такий спосіб зазвичай не використовується для простих типів на кшталт *int*, так як набагато зручніше і природніше оголосити змінну звичним для нас чином. А ось використання прикладу з покажчиком *q* на масив зустрічається значно частіше. Це, так звані, динамічні масиви.

Після обробки динамічних змінних пам'ять необхідно вивільнити, а відповідну вказівку обнулити. Якщо цього не зробити, то пам'ять можна вичерпати. Вивільняють пам'ять за допомогою команди *delete <назва покажчика>*

Оператор *delete* знищує об'єкт, створений за допомогою *new*, віддаючи тим самим розподілену пам'ять для повторного використання. Оператор *delete* може набувати таких форм:

delete *вираз*;

delete [] *вираз*;

Перша форма використовується, якщо відповідний вираз **new** розміщував не масив. У другій формі присутні порожні квадратні дужки, що показують, що спочатку розміщувався масив об'єктів. Оператор **delete** не повертає значення, тому можна сказати, що повертається їм тип - **void**.

Щоб вказівник не вказував ні на одну ділянку пам'яті, його необхідно обнулити такою командою*: <назва вказівника> = NULL*;

Значенням (адресою) такого вказівника буде нульовий адрес 0x00000000. Тут не може бути розміщено значення жодного даного.

Приклад. Розглянемо, як проходить виділення пам'яті.

Виділимо пам'ять для двох Змінних цілого типу і присвоїмо їм деякі значення. Потім виділимо пам'ять.

```
#include <iostream.h>
void main()
{
   int *c1=new int;              // Готуєм пам'ять для цілого числа
   *c1=5;                        // Змінна *c1 отримує значення 5
   int *c2=new int(7);     // Виділяється пам'ять для c2 и *c2 отримує
   значеня 7
   cout<<*c1<<'\t'<<*c2<<'\n';              // Виводимо 5 і 7
   c1=c2;       // Переадресація – c1 буде вказувати  на ту ж частну пам'яті,
   що і c2
   cout<<*c1<<'\t'<<*c2<<'\n';              // Виводимо 7 і 7
   delete(c2);                   // Пам'ять, представлена для c2, вивільня-
   ємо
   c2= NULL;                                // Анулювання  вказівника
   cout<<*c1<<"\n";                              // Виводимо 0
}
```

Справка. Замість значення NULL можна записати <назва вказівника>=0.

§ 8.5. Багатовимірні динамічні масиви

При вирішенні на комп'ютері серйозних завдань, наприклад, при розробці додатків, що інтенсивно використовують ресурси графіки, завжди потрібно мати під рукою достатню кількість ресурсів, які лімітовані системою. Тому ефективні алгоритми і способи управління динамічною пам'яттю часто набувають вирішального значення. Принцип організації динамічного двовимірного масиву, який часто використовується в подібних випадках, найпростіше усвідомити за допомогою такої схеми:

Адреса масива	Адреси масивів адрес	Серія окремих одномірних масивов		
A	*a*[0]	*a*[0][0]	*a*[0][1]	*a*[0][*n*-1]
	a[1]	*a*[1][0]	*a*[1][1]	*a*[1][*n*-1]
	...	...	...	...
	a[*n*-1]	*a*[*n*-1][0]	*a*[*n*-1][1]	*a*[*n*-1][*n*-1]

Відмінність описаної схеми від схеми статичного двомірного масиву полягає в тому, що тепер для адрес *a, a [0], a [1], ... a [n-1]* має бути відведено реальний фізичний простір пам'яті. У той час як для статичного двомірного масиву вираз *виду a, a [0], a [1], ... a [n-1]* були всього лише можливими конструкціями для посилань на реально існуючі елементи масиву, але самі ці вказівники не існували як об'єкти в пам'яті комп'ютера. Алгоритм виділення пам'яті для двовимірного динамічного масиву такий:

1. Визначаємо змінну а як адресу масива адрес:

   float ***a*;

2. Виділяємо область пам'яті для масиву з *n* покажчиків на тип *float* і присвоюємо адресу початку цієї пам'яті вказівником *a*. Оператор, який виконує ці дії виглядає так:

   a=**new float*** [*n*];

3. В циклі пробігаємо по масиву адрес *a*[], присвоюючи кожному вказівнику *a*[*i*] адреса знову виділеної пам'яті під масив з *n* чисел типу **float**.

При роботі з масивами що задаються динамічно часто забувають звільняти пам'ять, захоплену для масиву. Пам'ять слід знову повертати в розпо-

рядження операційної системи, тобто звільняти за допомогою операції **delete**. Правда, при завершенні роботи функції **main** автоматично знищуються всі змінні, створені в програмі, і покажчики сегментів пам'яті отримують свої вихідні значення. Однак при розробці складних багатомодульних комплексів програм слід пам'ятати про те, що виділена пам'ять «повисає», стає недоступною операційній системі при виході з області дії покажчика, який посилається на її початок. Це може викликати відмову у виділенні нової пам'яті в якомусь іншому програмному модулі, якщо весь обсяг вільної області пам'яті буде вичерпаний. Операція **delete** спільно з операцією **new** дозволяє контролювати процес послідовного виділення і вивільнення динамічної пам'яті. Щоб звільнити пам'ять, виділену для однієї змінної d, наприклад, за допомогою оператора **double * d = new double** ;, досить в кінці функції або блоку, де використовувалася змінна d, записати **delete d** ;. Якщо був розміщений масив змінних, наприклад float *** p = new float [200]**, то в сучасних версіях компіляторів слід звільняти пам'ять оператором **delete [] p ;.**

Тут квадратні дужки вказують компілятору на те, що звільняти слід ту кількість осередків, яке було захоплено в останній операції new в застосуванні до покажчика p. Явно вказувати це число не потрібно.

Приклад.

```
// Динамічний захват і звільнення пам'яті
    #include <iostream.h>
    double *a;                          // Одна змінна
    double *d;                          // Одномірний масив
    double **dd;                        // Двомірний масив

    void GetMem()
    {
        int i;
    // Захват пам'яті
        a=new double;                   // Одна змінна
        d=new double[4];                // Одномірний масив
```

```
// Двомірний масив
        dd=new double*[3];
        for(int i=0; i<3; i++)
                dd[i]=new double[2];
// Присвоєння
        *a=1.0;                         // Одна змінна
        cout<<"a="<<*a<<endl;
        cout<<"Address of a="<<a<<endl;
// Одномірний масив
        cout<<"Massiv nachinaetsya po adresu"<<d<<" i soder*it\n";
        for(i=0; i<4; i++)
        {
                d[i]=double(i);
                cout<<"d["<<i<<"]="<<d[i]<<endl;
        }
        cout<<"2D-massiv nachinaetsya po adresu "<<dd<<" i soder*it \n";
        for(i=0; i<3; i++, cout<<endl)                  // Двумерный массив
                for(int j=0; j<2; j++)
                {
                        dd[i][j]=(double)(i + j);
                        cout<<"dd["<<i<<"]["<<j<<"]="<<dd[i][j]<<endl;
                }
}
void FreeMem()                          //Звільнення памяті
{
        delete a;                       // Одна змінна
        delete [] d;                    // Одномірий масив
// Двомірний масив
        for(int i=0; i < 3; i++)
                        delete [] dd[i];
        delete [] dd;
}
```

```
int main()
{
        GetMem();
        FreeMem();
}
```

Після звільнення пам'яті покажчики *a, d, і dd* продовжують, проте, вказувати на ті ж адреси, що і до звільнення (проте ця пам'ять вже не наша). У цьому легко можна переконатися, вставивши до і після операцій **delete** висновок:

```
cout<<"a="<<a<<", d="<<d<<", dd="<<dd<<", dd[0]="<<dd[0];
```

Отже, необхідно акуратно працювати з вказівниками, які адресують звільнену пам'ять і стежити за зверненням до пам'яті, раніше займаної об'єктом.

§ 8.6 Масиви в якості параметрів функцій

У тіло функцій в якості аргументів можна передавати значення, що зберігаються в масивах. При виконанні функції, параметр типу масиву перетворюється компілятором в покажчик на тип масиву. Наприклад, якщо аргумент-масив має тип unsigned long, при виклику він буде перетворений в *unsigned long* *. Таким чином, зміна в функції значення будь-якого елемента масиву, що є аргументом, обов'язково вплине і на оригінал. Масиви відрізняються від інших типів тим, що їх не можна передавати за значенням - всередину тіла функції потрапляє тільки адреса масиву.

Синтаксис виклику функції при цьому може бути наступним:

FunctionName(*ArrayName*);

Тоді прототип функції включає вказівку як параметр типу переданого масиву і наступних за ним прямокутних дужок. наприклад:

FunctionName(**char**[]);

Другий варіант синтаксису передачі масива в функцію – коли прототип функції містить символ операції взяття адреси після вказівки типу аргумента:

char ***FunctionName***(**char**&);

При цьому синтаксис виклику функції приймає наступний вигляд:

FunctionName(*****ArrayName*);

Нижче приводиться приклад, ілюструючий обидва варіанти передачі масива в якості параметра функції.

```
#include <iostream.h>
// Прототипи функцій:
void Out1(int[]);
void Out2(int&);
int main()
{
        int Array[]={10,8,6,4,2,0};
        // Виклик першої функції:
        Out1(Array);
        cout<<'\n';
        // Виклик другої функції:
        Out2 (*Array);
        cout<<'\n';
}
// Реалізація обох функцій:
void Out1(int arr[])
{
        for (int i=0; i<sizeof(arr); i++)
                        cout<<arr[i]<<'  ';
}
void Out2(int& arr)
{
        for(int i=0; i<sizeof(arr); i++)
                        cout<<*(&(arr)+i)<<'  ';
}
```

У розглянутому прикладі оголошується масив *Array [],* який містить шість цілочисельних значень, і здійснюється його передача в функції *Out1 () і Out2 ().* Обидві функції виконують одну і ту ж дію - виводять вміст масиву-аргументу на екран і відрізняються тільки інтерфейсом.

Неявне перетворення масиву у вказівник при виконанні функції призводить до втрати інформації про розмір масиву. Викликаюча, повинна якимось чином визначити цей розмір, щоб виконувати осмислені дії.

При оголошенні багатовимірного масиву як параметра функції можна опустити тільки першу розмірність.

int g (..., int x [] [10], ...) {...} // Друга і наступні розмірності обов'язкові

Використання в якості параметра функції багатовимірного масиву ускладнено, тому на практиці частіше за все здійснюється передача масиву вказівників, що значно спрощує синтаксис.

Приклад.

```
/*Програма обчислює суми елементов строк двомірного динамічного масиву*/
    #include <iostream.h>
    /*функцуя, обчислююча суму елементів строки */
    int sum_str(int*, int);
    int main()
    {
        int **data;              //змінна - вказівник на двомірний масив
        int size_str;                  //тут будемо збнрігати розмір масива
        int size_stb;
        int i;
        cout<<"\nVvedite kolichestvo strok massiva:";
        cin>>size_str;
        cout<<"\n Vvedite kolichestvo stolbcov massiva:";
        cin>>size_stb;
```

```
        data=new int*[size_str];
        for(int i=0; i<size_str; i++)
                data[i]=new int[size_stb];
/* тут data використовується в якості базової  адреси динамічно роз-
міщуваного двомірного масиву*/
//організовуємо ввід елементів масиву
        for(i=0; i<size_str; i++)
        {
                for(int j=0; j < size_stb; j++)
                {
                        cout<<"Vvedite A["<<i<<"]["<<j<<"]=";
                        cin>>data[i][j];
                }
                cout<<endl
        }
//виводимо результат
        for(i=0; i<size_str; i++)
                        cout<<" Symma=" <<sum_str(data[i],size_stb)<<"\n";
// звільнюємо пам'ять
        delete[] data;}
/* функція виконує підрахунок суми елементів рядку*/
int sum_str(int *a, int size)
{
        int sum=0;
        for (int i=0; i<size; i++)
                        sum+=a[i];
        return (sum);
}
```

Розглянемо приклад, який використовує динамічний розподіл пам'яті.

Приклад.

```
/*  Програма обчислює середнє арифметичне елементів динамічного масиву.
Розмір масиву задається користувачем!*/
    #include <iostream.h>
    /*функція, що обчислює середнє арифметичне елементів масиву */
    double avg_arr(const int[], int);
    int main()
    {
        int *data; //змінна - вказівник  на ціле
        int size; //тут будемо зберігати  розмер масиву
    / * Рядок виводить на екран напис - Введіть розмір масиву * /
        cout<<"\Vvedite razmer massiva:";
    /*запросили у користувача  інформацію відносно розміру масиву */
        cin>>size;
        data=new int[size];
    /* тут data використовується в якості базової адреси динамічно розміщуваного масиву з кількістю елементів задається значенням size */
    //організуємо ввід елементів масиву
        for (int j=0; j<size; j++)
        {
            cout<<"Vvedite element A["<<j<<"]=";
            cin>>data[j];
        }
    //виводим результат
        cout<<"Srednee arifmeticheskoe elementov massiva"<<avg_arr(data,size)<<"\n";
    // звільнюємо пам'ять
        delete[] data;
    }
    /* функція виконує підрахунок среднього  арифметичного елементів масиву */
```

```
double avg_arr(const int a[], int size)
{
        int sum=0;
        for (int i=0; i<size; i++)
        sum+=a[i];
/* перетворимо sum до double, інакше отримали б тількм цілу  частину від ділення */
        return double (sum)/size;
}
```

Тема 9. Рядки

§ 9.1. Робота з рядками в С ++.

Дуже часто, на практиці, доводиться стикатися з завданнями, які зводяться до роботи над рядками. Але мова С ++ не підтримує окремий строковий тип даних

Рядок в С ++ - це масив символів, що закінчується нульовим символом ('\0').

Таким чином, можна визначити рядки двома способами: як масив символів або як покажчик на перший символ рядка, наприклад:

char str1 [] = "string1"; // оголошення рядка за допомогою масиву символів.

Отже, тепер докладніше. Масив - це набір однорідних значень. Так ось рядок є не що інше, як набір символів, і, відповідно, для зберігання рядків можна використовувати символьні масиви. Наприклад, рядок "QWERTY" має тип **char**[7], а порожній рядок "" має тип char [1]. Чому **char** [1]? Саме тому, що будь-який рядок завершується так званим нульовим символом, тобто символом, код якого в ASCII-таблиці дорівнює 0 (цей символ також є escape-символом і його символьний еквівалент представляється як '\ 0'). Завдяки цій властивості завжди можна визначити кінець рядка, якщо рядок займає меншу кількість символів, ніж та кількість, що було зазначено в квадратних дужках при оголошенні масиву, тобто визначити фактичну довжину рядка, що зберігається в масиві.

Одна з чудових особливостей при роботі з рядками - це можливість спрощеної початкової ініціалізації. Наприклад, оголошення **char** *str*[] = "ABCDE"; привласнює змінній-рядку початкове значення "ABCDE". А точніше, створює масив з 6 символів: 'A', 'B', 'C', 'D', 'E' і символу '\ 0'.

Початкова ініціалізація символьного масиву дійсно відрізняється від ініціалізації будь-якого іншого масиву - можна просто привласнити необхідний рядок імені масиву з порожніми квадратними дужками. С ++ сам підрахує довжину рядка і виділить відповідний обсяг пам'яті під масив для розміщення в ньому необхідного рядка.

Відразу необхідно зазначити, що С ++ сам автоматично зробить останній елемент масиву нульовим символом, тобто, хоча в даному випадку, масиву str присвоюється рядок "ABCDE", довжина якого становить 5 символів, С ++ виділяє пам'ять під 6 символів, записує туди рядок і потім в останній символ (п'ятий за рахунку від 0) записує нульовий символ.

Слід також зазначити, що при початковій ініціалізації символьного масиву (як і будь-якого іншого) можна вказувати в квадратних дужках його розмір з метою подальшого використання масиву ще для будь-яких цілей (наприклад, для зберігання будь-якої іншої рядка). Оголошення **char** *str*[10]="ABCDE"; створює масив з 10 символів і перші п'ять елементів цього масиву приймають значення 'A', 'B', 'C', 'D' і 'E' відповідно, інші символи будуть нуль-символи. В даному випадку, в перші 5 елементів масиву записується рядок "ABCDE", а всім іншим елементам присвоюються нулі.

Для початкової ініціалізації символьного масиву можна використовувати правила ініціалізації довільного масиву, тобто, використовуючи фігурні дужки, тільки в даному випадку доведеться явно вказувати нульовий символ, яким закінчується рядок. Давайте розглянемо попередній приклад з використанням загальних правил початкової ініціалізації масиву.

char *str*[]={'A','B','C','D','E','\0'};

Очевидно, спрощений варіант початкової ініціалізації строкового масиву значно простіший, але ще раз зазначимо, що його можна використовувати тільки для символьних масивів.

Типова помилка програмування.

1. Не виділяється достатньо місця в масиві символів для зберігання нульового символу, що завершує рядок.

2. Створення або використання "рядки", яка не містить завершального нульового символу.

3. Плутанина в символьні і рядкові константи.

Символьна константа - це один символ, взятий в апострофи, наприклад: 'A' або '\ n'. *Строкова константа* - це послідовність символів, взята в подвійні лапки. У числі символів рядка можуть перебувати будь-які си-

мвольні константи, наприклад, "Visual C ++ \ n" складається з наступних символів: 'V', 'i', 's', 'u', 'a', 'l', '', 'C', '+', '+', '\ n', '\ 0'. Таким чином, "А" - це строкова константа і складається з двох символів: 'А' і '\ 0'. Сусідні рядкові константи транслятором "склеюються", наприклад: "АБВ" "ДЕ" означає те ж, що "АБВГД".

Приклад 1.

```
// Задана рядок, скопіювати його в символьний масив
#include<iostream.h>
void main()
{
    / * Оголошуємо символьний масив str1 і инициализируем
    його * /
    char str1[]="1234567890",
       str2[11]; // оголошуємо символьний масив без ініціалі-
  зації
       / * В циклі, поки не зустрінеться кінець рядка присво-
  юємо поточного елементу масиву str2 символ з масиву str1 */
          for(int i=0; str1[i]!='\0'; i++) str2[i]=str1[i];
    str2[i] = '\0'; // копіюємо нуль-символ в str2.
    cout<<str2<<'\n'; // вивід рядка на екран
}
```

Зверніть увагу, вихід з циклу відбувається, коли str1 [i] дорівнює нуль-символу, тобто нуль-символ не буде копіюватися в str2, отже, це потрібно зробити за циклом.

Ще одна особливість роботи з символьними масивами - якщо елементи довільного масиву можна вводити з клавіатури і виводити на екран тільки по одному елементу в циклі, то в символьний масив можна ввести відразу весь рядок, використовуючи оператор введення

cin>>Ім’я_масиву;

і, аналогічним чином, вивести відразу весь рядок на екран, використовуючи оператор виведення

cout<< Ім’я_масиву;

Слід відразу зазначити, що при введенні з клавіатури рядка оператор cin автоматично додає в кінець рядка нульовий символ, так що необхідно враховувати цей факт при вказівці кількості елементів при оголошенні масиву.

Приклад 2.

```
#include <iostream.h>
void main()
{
char str[31];                          // оголошення символь-
ного масиву
cout<<"Vvedit raydok (max 30 symvoliv):";
cin>>str;                                    // введення
рядка
cout<<"\nVu vveli raydok:"<<str;             // виведення
рядка
}
```

Як видно, в даному прикладі виділяється пам'ять під 31 символ, але користувачеві в запрошенні вказується, що він може ввести рядок з розміром максимум 30 символів, з огляду на той факт, що оператор cin додасть ще один нульовий символ в кінець рядка автоматично, і під нього також необхідно передбачити виділення пам'яті. Далі після запрошення вводимо відразу весь рядок з клавіатури в масив і потім з відповідним повідомленням виводимо весь рядок на екран монітора.

Другий спосіб визначення рядка - це використання вказівника на символ. Оголошення **char** * b; задає змінну b, яка може містити адресу деякого об'єкту. Однак в даному випадку компілятор резервує місце для зберігання символів і не ініціалізує змінну b конкретним значенням. Зробити це можна, наприклад, присвоївши b покажчик на вже існуючий символьний масив, або динамічно виділити пам'ять під новий масив.

Приклад 3.

```
#include<iostream.h>
void main()
{
    char str[]="Привіт, світ!";   // оголошуємо символьний масив
    char *b;                      // оголошуємо вказівник на символ
    b=&str[8];                    // тепер b вказує на 8-ий символ str
    *b='М';                  // присвоюємо першому елементу b символ 'С'
    cout<<b;                 // виводимо рядок b на екран (Світ!)
}
```

Отже, підіб'ємо підсумки.

1. Рядок можна визначити як масив символів або як вказівник на символ.
2. Будь-яка рядок закінчується нульовим символом. (Завдяки цій властивості завжди можна визначити кінець рядка, якщо рядок займає меншу кількість символів, ніж та кількість, що було зазначено в квадратних дужках при оголошенні масиву).
3. Для рядків можлива спрощена початкова ініціалізація (*в порівнянні з не символьними масивами*).
4. У символьний масив можна ввести відразу весь рядок, використовуючи оператор введення **cin** >> *Ім'я_масиву* ;, і аналогічним чином вивести відразу весь рядок на екран, використовуючи оператор виведення **cout** << *Ім'я_масиву* ;.

§ 9.2 Функції роботи з рядками з бібліотеки обробки рядків

Розглянемо деякі типові функції стандартної бібліотеки *string.h*. Це бібліотека обробки рядків, яка забезпечує багато корисних функцій для роботи із рядковими даними, наприклад, порівняння рядків, пошук в рядках потрібних символів і інших подстрок, розмітку рядків (поділ рядків на логічні шматки) і визначення довжини рядка.

1. Функція

int *strlen* (**const char** *s);

Визначає довжину рядка s. Повертає кількість символів, що передують завершального нульового символу. Зверніть увагу, завершальний нуль-символ в довжину не включається. Наприклад,

```
cout<<strlen("Hello!");                    // на екрані буде 6
char *str="one";
cout<<strlen(str);                         // на екрані буде 3
```

2. Функція

char **strcpy* (**char** *s1, **const char** *s2);

Копіює рядок s2 в масив символів s1. Повертає значення s1. Масив символів s1 повинен бути досить великим, щоб зберігати рядок і її завершальний нульовий символ, який також копіюється. Наприклад,

```
char str[25];              // оголошуємо символьний масив з 25 елементів
char *ps = new char [25];  /*Оголошуємо вказівник на символ і
                           динамічно виділяємо пам'ять під 25 символів * /
strcpy(str, "ABCDE");      // копіюємо в str строкову константу "ABCDE"
cout<<str;                 // виводимо str на екран. На екрані буде ABCDE
strcpy(ps, "QWERTY");      // копіюємо в ps строкову константу "QWERTY"
```

```
cout<<ps;                    // виводимо ps на екран. На екрані буде
QWERTY
delete[] ps;                 // звільняємо пам'ять
```

Зверніть увагу , якщо треба, щоб один рядок містив інший, потрібно **скопіювати** його вміст, **а не присвоїти!** Так, наприклад, в даному випадку інструкція *ps* = "QWERTY" була б помилковою. Компілятор, зустрічаючи таку інструкцію, створюють рядок "QWERTY", за якою слідує нульовий символ і привласнює значення початкового адреси цього рядка (адреси символу Q) змінної ps. Таким чином, втрачається початкове значення *ps*, а значить неможливо коректно звільнити пам'ять під *ps*.

3. Функція

int ****strcmp*(const char **s1**, const char **s2**);**

Порівнює рядки s1 і s2 (по ASCII-кодами). Функція повертає значення 0, якщо рядки s1 і s2 рівні, значення менше нуля, якщо рядок s1 менше s2, і значення більше нуля, якщо s1 більше s2. Зверніть увагу, рядки порівнюються не по довжині, а посимвольно, по ASCII-кодам (тобто "g" більше "ff"). Наприклад,

```
cout<<strcmp("compare", "string");      /* на екрані буде −1,
оскільки
                                        "compare" менше "string"
*/
cout<<strcmp("abcde", "abc");      /* на екрані буде 1, оскільки
                                        "abcde" більше "abc" */
cout<<strcmp("one", "one");             /* на екрані буде  0,
оскільки
                                        рядки рівні*/
```

4. Функція

char ****strcat*(char **s1**, const char **s2**);**

Додає рядок s2 до рядка s1. Перший символ рядка s2 записується поверх нуль-символу рядка s1. Повертає s1. Під s1 має бути виділено пам'яті не менше ніж (strlen (s1) + strlen (s2) +1). Наприклад,

char *st1*[25] = "День";
cout<<***strcat***(*st1*, " добрий!"); // на екрані буде
День добрий!

5. Функція

char *****strncpy**(**char** *s1, **const char** *s2, **int** *n*);

Копіює не більше n символів рядка s2 в масив символів s1. Повертає s1.

6. Функція

int *****strncmp**(**char** *s1, **const char** *s2, **int** *n*);

Порівнює до n символів рядка s1 з рядком s2. Повертає 0, менше, ніж 0 або більше, ніж 0, якщо s1 відповідно дорівнює, менше або більше s2.

7. Функція

char *****strncat**(**char** *s1, **const char** *s2, **int** *n*);

Приєднує перші n символів рядка s2 до рядка s1. Повертає s1.

8. Функція

char *****strchr**(**const char** *s, **char** *c*);

Перевіряє рядок s на вміст символу, який зберігається в c. Результатом функції є адреса першого входження символу c в рядок s, тобто повертається рядок, який починається від першого входження заданого символу до кінця рядка s. Якщо символ не знайдено, повертається NULL. Застосовується в виразах. Наприклад,

char *str*[20]="ABCDEXYZ";
cout<<***strchr***(*str*, 'X'); // на екрані буде
XYZ

або

```
char str[20]="ABCDEXYZ";
if (strchr(str, 'q') == NULL) cout<<"Нема такого символу!";
```

9. Функція

char *__strstr__(**const char** *s1, **const char** *s2);

Перевіряє рядок s1 на утримання підрядка s2. Результатом функції є адреса першого входження підрядка s2 в рядок s1. Якщо підрядок не знайдено, повертається NULL. Наприклад,

```
char str[20]="ABCDEXYZ";
char *ps=strstr(str, "DEX");
if (ps!=NULL)
  cout<<ps;                          // На екрані буде DEXYZ
else cout<<"Нема такого підрядка!";
```

10. Функція

char *__strlwr__(**char** *s);

Конвертує рядок до нижнього регістру (тобто переводить рядок в рядкові символи). наприклад,

```
char str[30] = "ABCDE_123_ijk_XYZ";
cout<<strlwr(str);                   // на екрані буде abcde_123_ijk_xyz
```

11. Функція

char *__strupr__(**char** *s);

Конвертує рядок до верхнього регістру (тобто переводить рядок в прописні символи).

12. Функція

char *__strset__(**char** *s, **char** *ch*);

Замінює ВСІ символи в рядку s на символ ch. Наприклад,

```
char str[30] = "ABCDE";
cout<<strset(str, 'x');              // на екрані буде xxxxx
```

13. Функція

char ****strnset***(**char** **s*, **char** *ch*, **int** *n*);

Замінює перші n символів в рядку s на символ ch.

14. Функція

char ****strrev***(**char** **s*);

Змінює порядок проходження символів в рядку на протилежний (змінює перший символ з останнім, другий символ з передостаннім і т.д.). Наприклад,

char *str*[30] = "12345";
cout<<***strrev***(*str*); // на екрані буде *54321*

Типова помилка програмування.

Необхідно включити заголовки *string.h* при використанні функцій з бібліотеки обробки рядків.

Також ознайомимося з двома функціями, які можуть допомогти програмісту при читанні символів з клавіатури:

Функція

int *getch*(void);

Повертає ASCII-код натиснутої клавіші.

Функція

int *getche*(void);

Повертає ASCII-код натиснутої клавіші і виводить символ на екран.

Прототипи останніх двох функцій описані в файлі *conio.h*, який входить в стандартну бібліотеку мови С ++.

§ 9.3. Робота з рядками в С ++. Приклади.

Приклад 4.

Задача. Дано рядок символів, підрахувати скільки разів серед символів рядка зустрічається буква x.

```
#include <iostream.h>
```

```
void main( void )
{
        char str[100];                  // оголошення рядка символів
        cout<<"\nVvedite stroky: ";
// просимо користувача ввести рядок символів
        cin >> str;             // зчитуємо рядок, введену користувачем
        int count = 0;          /* Оголошення змінної-лічильника, в якій
                        зберігатимемо кількість входжень x в рядок  */
        int i = 0;
        while(str[i]!='\0')
        {
                if (str[i]=='x') count++;
                i++;
        }
        cout<<"\n Simvol x vxodit v stroky –"<<count;
// виводимо результат на екран
}
```

Приклад 5

Задача. Написати програму, яка отримує від користувача набір символів, виключаючи пробіл, і видаляє з цього набору все входження символів S і s.

Найбільший інтерес представляє аналіз рядка. Для реалізації цього аналізу потрібно поелементно рухатися від нульового індексу масиву до останнього і робити перевірку кожного елемента. Якщо буде зустрінутий такий елемент з індексом i, то нам потрібно змістити всі елементи з індексами, великими ніж i, на один індекс менше. Іншими словами:

Нехай дана наступна рядок:: A b s D e f

0 1 2 3 4 5 ------ індекси

Перевіряючи поіндексно кожен елемент масиву, бачимо, що 2 елемент масиву і є шуканий символ. Тоді, потрібно змістити кожен елемент масиву на 1 індекс менше, тобто отримати наступний результат: A b D e f

0 1 2 3 4

```
#include <iostream.h>
void main()
{
   const int CharCount=10;
//  задамо розмірність масиву через константу
   char arr[CharCount];                    //  оголошення символьного масиву
//  Попередимо користувача, що введення обмежено розмірністю масиву
   cout<<"\nVvedite stroky, no ne bolee, chem "<<CharCount-1<<"
simvolov\n";
   cin >> arr;                                          // введення рядка
   int i=0;
   while (arr[i]!='\0')
//  Цикл працює поки не зустрінеться ознака кінця рядка
         if (arr[i]=='S'||arr[i]=='s')
//  Перевірка на наявність шуканого символу
            {  /*  Якщо це шуканий символ, то перенесемо решту рядка
                  на один елемент вліво.*/
               for (int j=i;arr[j]!='\0';j++)
               arr[j]=arr[j+1];
             }
            else i++;
//  а якщо це не шуканий символ, то будемо рухатися по рядку далі
   cout<<endl<<arr<<endl;                          //  вивести результат
}
```

Зауваження до задачі. Як Ви думаєте, що станеться, якщо в програму ввести рядок містить прогалини (тобто порушити умови задачі)?

Приклад 6

Задача. Написати програму порівняння двох рядків.

Перед тим як перейти безпосередньо до програми, зробимо приміт-ку. У деяких випадках бажано вводити в масив повний рядок тексту. З цією метою С ++ забезпечений функцією **cin.getline (s).** Функція **cin.getline (s)** вимагає три аргументи - масив символів, в якому повинна зберігатися рядок тексту, довжина і символ обмежувач. Наприклад, фрагмент програми

char *sentence*[80];
cin.*getline*(*sentence*, 80, '\n');

оголошує масив sentence з 80 символів, потім зчитує рядок тексту з клавіатури в цей масив. Функція припиняє зчитування символів у випадках, якщо зустрічається символ-обмежувач '\ n', якщо вводиться вказівник кінця файлу або якщо кількість лічених символів виявляється на один менше, ніж зазначено в другому аргументі (останній символ в масиві резервується для завершального нульового символу). Якщо зустрічається символ обмежувач, він зчитується і відкидається. Третій аргумент **cin.getline (s)** має '\ n' в якості значення за замовчуванням, так що попередній виклик функції міг бути написаний в наступному вигляді:

cin.getline(sentence, 80);

```
#include<iostream.h>
#include<string.h>
void main()
{
 int len; // довжина рядка, який вводиться
 char s[81];  // місце зберігання рядка, який вводиться
 char *s1,*s2;
cout<<"Vvedite pervyyu stroky: ";
cin.getline(s, 80); //  введення першого рядка
len = strlen(s);    // визначення довжини рядка
s1 = new char[ len + 1]; // динамічне виділення пам'яті під рядок s1
strcpy(s1, s); // копіювання введеного рядка в рядок s1
cout<<" Vvedite vtoryyu stroky: ";
cin.getline(s, 80);  // введення другого рядка
```

```
len = strlen(s);
s2 = new char[len + 1];  // динамічне виділення пам'яті під рядок s2
strcpy(s2, s);
// який з введених рядків більше?
if(strcmp(s1, s2) > 0)
  cout<<"Stroka s1:\t"<<s1<<"\n\t > \n"
          << "Stroka s2:\t"<<s2<<endl;
    else if (strcmp(s1, s2) == 0)
       cout<<"String s1:\t"<<s1<<"\n\t=\n"
      <<"String s2:\t"<<s2<<endl;
    else
       cout<<"String s1:\t"<<s1<<"\n\t < \n"
      <<"String s2:\t"
      <<s2<<endl;
    delete []s1;    // видалення рядків з пам'яті
    delete []s2;
}
```

Приклад

Розглянемо таку задачу: необхідно реалізувати наступні функції для роботи

з масивами.

1) Функція введення елементів масиву;

2) Функція роздруківки масиву;

3) Функція сортування масиву;

4) Функція додавання нового елемента в кінець масиву;

5) Функція видалення заданого елемента.

Для даної задачі були використані матеріали поточного уроку, а саме: передача масивів у функцію, передача параметрів по посиланню (щоб змінювалися

реальні дані), повернення покажчика з функції.

Всі ці функції часто застосовні в реальному житті, например, при створенні всіляких довідників.

```
#include <iostream.h>
int* Add(int*, int&); // Функція додавання елемента в кінець масиву
int* Del(int*, int&); // Функція видалення елемента в заданій позиці
void Input(int*, int);// Функція введення елементів масиву
void Sort(int*, int); // Функція сортування елементів масиву
void Print(int*, int);// Функція роздруківки елементів масиву
void main()          //  Приклад реалізації
{
        int n, *a;
        cout<<"Vvedite razmer massiva:\t";
        cin >> n;          //  Кількість елементів масиву
        a = new int[n];// Виділення пам'яті для масиву з n елементів

      Input(a, n);      // Введення елементів масиву

        cout<<"\nArray:\n";
        Print(a, n);       // Виведення масиву на екран
        Sort(a, n);        // Сортування масиву

        cout<<"Otsortirovannuj massiv:\n";
        Print(a, n);       // Виведення відсортованого масиву на екран
        a = Add(a, n);  // Зміна масиву - додавання нового елемента

        cout<<"Novuj massiv:\n";
        Print(a, n);       // Виведення зміненого масиву
        a = Del(a, n);   // Зміна масиву - видалення зазначеного елемента
        cout<<" Novuj massiv:\n";
        Print(a, n);       // Виведення зміненого масиву
        delete [] a;       // Звільнення пам'яті, відведеної під масив
}

void Input(int *a, int n)
```

```
{
        for(int i = 0; i < n; i++)
        {
        cout<<"Element #"<<i + 1<<"\t";
        cin >> a[i];      // Введення елементів масиву
        }
}

void Print(int *a, int n)
{
        for(int i = 0; i < n; i++)
        cout<<a[i]<<"   ";
        cout<<endl;     // Виведення елементів масиву
}

void Sort(int *a, int n)
{
        int temp;         // Тимчасова змінна для обміну значень
        bool flag = true;// Прапор закінчення сортування
        for(int j = 1; ; j++)
        {
                for(int i = 0; i < n - j; i++)
                        if(a[i] > a[i+1])
                        {
                        temp = a[i];
                        a[i] = a[i+1];
                        a[i+1] = temp;
                        flag = false;
                        }
                if(flag == true)
                break;
                flag = true;
```

```
        }
}
int* Add(int *a, int &n)
{
        int i, m;
        int *p = new int[++n];
// Створення тимчасового масиву більшого розміру

        cout<<"Vvedite dobavlyaemuj element:\t";
        cin >> m;               // Додається елемент
        for(i = 0; i < n - 1; i++)
                p[i] = a[i];    // Збереження елементів
        delete [] a;            // Видалення старого масиву

        p[n - 1] = m;           // Новий елемент
        return p;               // Повернення адреси нового масиву
}

int* Del(int* a, int &n)
{
        int i, m, j = 0;
        n--;                    // Зменшення розмірності масиву
        int *p = new int[n];    // Створення тимчасового масиву меншого
розміру

        cout<<"Element s kakim indeksom ydalim:\t";
        cin >> m;       // Введення індексу видаляється елемента (індекс з
нуля)
        for(i = 0; i < n; i++)
        {
           if(i == m)
           j = 1;
```

```
        p[i] = a[i+j];// Збереження елементів, не враховуючи зазначений
    }

    delete [] a;      // Видалення масиву
    return p;         // Повернення адреси нового масиву
```

§ 9.3.1. Приклад на багатовимірні динамічні масиви

```
 /*
 Розглянемо наступну задачу: необхідно створити масив, що дозволяє
створювати рядки і працювати з ними (аналог довідника). Для роботи з
цим масивом рядків визначено такий набір операцій:
 • 1) додавання рядка в кінець масиву;
 • 2) вставка рядка в масив по заданому індексу;
 • 3) видалення рядка з масиву по заданому індексу;
 • 4) очистка масиву (видалення всіх рядків);
 • 5) висновок на екран вмісту масиву.
 * /
#include <iostream.h>
#include <string.h>

//  Набір констант, що представляють різні пункти меню
enum {ChoiceAddEnd=1, ChoiceInsert, ChoiceDelete, ChoiceDeleteAll,
ChoicePrint, ChoiceQuit};

int Menu();                    //  Виведення меню
char** AddLine(char**, int&);   //  Додавання рядка в кінець масиву
char** InsLine(char**, int&);   //  Вставка рядка в масив
char** DelLine(char**, int&);   //  Видалення зазначеного рядка з масиву
void DelAllLines(char**, int&); //  Видалення всіх рядків масиву
void Print(char**, int);        //  Роздруківка рядків масиву
bool IsArrayEmpty(int&);        //  Перевірка на наявність рядків в масиві
```

```
void main()
{
   char **c;       // Масив рядків
   int m = 0;      // Початкова кількість рядків масиву
   int choice = ChoiceAddEnd;

   while (choice != ChoiceQuit)    //  Поки  не обраний пункт ВИХІД
   {
      choice = Menu();           //  Виведення меню
      cin.ignore(1);             //  Очищення потоку введення

      switch (choice)            //  Вибір пункту меню
      {
      case ChoiceAddEnd:
         c = AddLine(c, m);      //  Додавання рядка в кінець масиву
      break;

      case ChoiceInsert:
         c = InsLine(c, m);      //  Вставка рядка в масив
      break;

      case ChoiceDelete:
         if (!IsArrayEmpty(m))   //  Якщо масив не порожній
            c = DelLine(c, m);  //  Видалення рядка
      break;

      case ChoiceDeleteAll:
         if (!IsArrayEmpty(m))   //  Якщо масив не порожній
            DelAllLines(c, m);  //  Видалення всіх рядків масиву
      break;
```

```
      case ChoicePrint:
         if (!IsArrayEmpty(m))   //  Якщо масив не порожній
            Print(c, m);        //  Роздруківка масиву
      break;

      case ChoiceQuit:
      break;

      default:                  //  В інших випадках
         cout<<"Error in choice!\n";
      break;
      }
   }
}

char** AddLine(char **c, int &m)
// Додавання рядка в кінець масиву
{
   char str[256];               //  Масив для введення нового рядка
   int n;                       //  Довжина введеного рядка
   int i;

   cout<<"Input string: ";
   cin.getline(str, 256);       //  Введення рядка
   n = strlen(str);             //  Обчислення довжини нового рядка

   if (m == 0)                  //  Якщо масив рядків порожній
   {
      m++;
      c = new char*[m];
      c[0] = new char[n + 1];   //  Створюємо новий рядок в масиві рядків
      strcpy(c[0], str);
```

```
                return c;
    }
    else
    {
                m++;
       char** t = new char*[m];//  Новий масив рядків
       for(i = 0; i < m - 1; i++)
       {
          t[i] = c[i];
       }
       t[m - 1] = new char[n + 1];
       strcpy(t[m - 1], str);          //  Копіювання нового рядка

       delete [] c;
       return t;                    //  Повернення нової  адреси масиву рядків }
          }
}

char** InsLine(char **c, int &m)
// Вставка рядка в масив
{
    char str[256];                //  Масив для введення нового рядка
    int n;                      //  Довжина введеного рядка
    int k;                      //  Позиція нового рядка в масиві
    int i, j = 0;

    cout<<"Input string: ";
    cin.getline(str, 256);
    cout<<"Input position # (0-"<<m<<"): ";
    cin >> k;
    while(k < 0 || k > m)          //  Перевірка на хибність введення
    {
```

```
    cout<<"Error !!!\nInput position # (0-"<<m<<"): ";
    cin >> k;
}

n = strlen(str);              //  Довжина нового рядка

if (m == 0)                   //  Якщо масив рядків порожній
{
    m++;
    c = new char*[m];
    c[0] = new char[n + 1];     //  Створюємо новий рядок в масиві рядків
    strcpy(c[0], str);
    return c;                    //  Повернення нового адреси масиву рядків
}
else
{
                m++;
    char** t = new char*[m];//  Новий масив рядків
    for(i = 0; i < m; i++)
    {
        if (i == k)
        {
            t[i] = new char[n + 1];
                                    strcpy(t[i], str);
            j = 1;
        }
        else
        {
            t[i] = c[i - j];
        }
    }
```

```
      delete [] c;                  //  Видалення масиву рядків
      return t;                     //  Повернення нового адреси масиву рядків
   }
}

char** DelLine(char **c, int &m)
//  Видалення зазначеного рядка з масиву
{
   int k;                    //  Індекс видаляється рядка
   int i, j = 0;

         cout<<"Input position # (0-"<<m - 1<<"): ";
   cin >> k;
   while(k < 0 || k >= m)              //  Перевірка на хибність введення
   {
      cout<<"Error !!!\nInput deleting position #: ";
      cin >> k;
   }

         m--;
   char** t = new char*[m];//  Створення нового масиву рядків
   for(i = 0; i < m; i++)
   {
      if (i == k)
         j = 1;
      t[i] = c[i + j];
   }

   delete [] c;               //  Видалення масиву рядків

   return t;                  //  Повернення нової  адреси масиву рядків
}
```

```
void DelAllLines(char **c, int &m)
// Видалення всіх рядків масиву
{
   for(int i = 0; i < m; i++)
      delete [] c[i];
   delete [] c;              //  Видалення всіх рядків масиву

   m = 0;
}

bool IsArrayEmpty(int &m)
//  Повертає істину, якщо масив рядків порожній; в зворотному випадку - брехня
{
   if (m == 0)
   {
      cout<<"Your line array is empty.\n";
      return true;
   }
   else
      return false;
}

int Menu()
//  Виведення меню
{
   int choice;
   cout<<"\n******* Menu *******\n";
   cout<<"1-Add  2-Insert  3-Delete  4-Delete All  5-Print  6-Quit\n";
   cin >> choice;                //  Вибір пункту меню
   if(choice < 0 || choice > 6)    //  Перевірка вибору
```

```
        choice = 0;
    return choice;                  //  Повернення значення установок
}

void Print(char **c, int m)
//  Роздруківка масиву рядків
{
    for(int i = 0; i < m; i++)
        cout<<i<<": "<<c[i]<<endl;
}
```

/*

На екран виводиться запрошення користувачеві вибрати той чи інший пункт меню, після вибору якого виконується відповідна дія./*

§ 9.4 Вказівники на функції

Перш ніж вводити вказівник на функцію, нагадаємо, що кожна Функція характеризується типом значення, що повертається, ім'ям, кількістю, порядком проходження і типами параметрів.

При використанні імені функції без подальших дужок і параметрів ім'я функції виступає в якості вказівник а на цю функцію, і його значенням служить адреса розміщення функції в пам'яті. Це значення адреси може бути присвоєно іншому вказівнику, і потім вже цей новий вказівник можна застосовувати для виклику функції. Однак у визначенні нового вказівник а повинен бути той же тип, що і повертається функцією значення, то ж кількість, порядок проходження і типи параметрів. Вказівник на функцію визначається наступним чином:

тип функції (* імя_вказівника) (специфікація_параметров);

Наприклад: int (* func1ptr) (char); - визначення вказівника func1ptr на функцію з параметром типу char, що повертає значення типу int.

Якщо наведену синтаксичну конструкцію записати без перших круглих дужок, тобто у вигляді int * fun (char); то компілятор сприйме її як прото-

тип якоїсь функції з ім'ям fun і параметром типу char, що повертає значення покажчика типу int *.

Другий Приклад: char * (* func2Ptr) (char *, int); - визначення покажчика func2Ptr на функцію з параметрами типу покажчик на char і типу int, що повертає значення типу покажчик на char.

У визначенні вказівника на функцію тип значення і сигнатура (типи, кількість і порядок опцій) повинні збігатися з відповідними типами та сигнатурами тих функцій, адреси яких передбачається привласнювати вводиться вказівником при ініціалізації або за допомогою оператора присвоювання. В якості найпростішої ілюстрації - невеликий Приклад:

Приклад №1

```
/*  Приклад визначення і використання
вказівників на функції
*/

#include<iostream.h>

// опис функції f1
void f1(void)
{
        cout<<"\n Виконується f1 () ";
}

// описание функции f2
void f2(void)
{
        cout<<"\n Виконується f2 () ";
}

void main()
{
        // объявление указателя на функцию,
```

```
    void (*ptr)(void); // ptr - вказівник на функцію
    ptr = f2;  // ptr ініціалізується адресом f2()
    (*ptr)();  // виклик f2 () на її адресу
    ptr = f1;  // присвоюється адреса f1 ()
    (*ptr)();  // виклик f1 () на її адресу
    ptr();  // виклик еквівалентний (*ptr)();
}
```

Результат виконання програми:

```
Виконується f2 ()
Виконується f1 ()
Виконується f1 ()
```

У програмі описаний вказівник ptr на функцію, і йому послідовно присвоюються адреси функції f2 і fl. Зверніть увагу на форму виклику функції за допомогою вказівника на функцію:

(*імя_ вказівник)(список_фактичних_параметрів);

Значним імені_ вказівника служить адреса функції, а за допомогою операції разименованія * забезпечується звернення за адресою до цієї функції. Однак буде помилкою записати виклик функції без дужок у вигляді * ptr (); Справа в тому, що операція () має більш високий пріоритет, ніж операція звернення за адресою *. Отже, відповідно до синтаксисом буде спочатку зроблена спроба звернутися до функції ptr (). І вже до результату буде віднесена операція разименованія, що буде сприйнято як синтаксична помилка.

При визначенні вказівник на функцію може бути ініціалізований. Як не започатковано значення має використовуватися адреса функції, тип і сигнатура (типи, кількість і порядок опцій) якої відповідають визначається вказівником.

При присвоєнні вказівників на функції також необхідно дотримуватися відповідність типів повертаються значенні функції і сигнатур для покажчиків правої і лівої частин оператора присвоювання. Те ж справедливо і при наступному виклику функцій за допомогою вказівників, тобто типи і кількість фактичних параметрів, використовуваних при зверненні до фун-

кції за адресою, повинні відповідати формальним параметрам, що викликається.

Наступна програма ілюструє гнучкість механізму викликів функцій за допомогою вказівників.

Приклад №2

```
/*   Виклик функцій за адресами через покажчик
*/
#include<iostream.h>

// опис функцій
// функциї одного типу з однаковими сигнатурами
int add (int n, int m) { return n + m;}
int div (int n, int m) { return n % m;}
int mult (int n, int m) { return n * m;}
int subt (int n, int m) { return n - m;}

void main()
{
        int (*par)(int, int); // вказівник на функцію
        int a = 6,  b = 2;  /* задаются початкові значення
                        для змінних a і b */
        char c = '+';
        while(c != ' ')
        {
                cout<<"\nАргументи: a = "<<a
                        << ", b = "<<b;
                cout<<"\n Результат для c = \'"<<c<<"\'"
                        << " дорівнює ";

                switch(c)
                {
                case '+':
```

```
                par = add; // par отримує адрес функції add
                c = '%'; break;
            case '-':
                par = subt;  // par отримує адрес функції  subt
                c = ' '; break;
            case '*':
                par = mult;  // par отримує адрес функції mult
                c = '-'; break;
            case '%':
                par = div;  // par отримує адрес функції div
                c = '*'; break;
            }
            cout<<(*par)(a,b); /* виклик функції за адресою,
            результат, який повертає функція, виводится на екран */
        }
        cout<<endl;
}
```

Результат виконання програми

```
Аргументи: a = 6, b = 2. Результат для c = '+' дорівнює 8
Аргументи: a = 6, b = 2. Результат для c = '%' дорівнює0
Аргументи: a = 6, b = 2. Результат для c = '*' дорівнює 12
Аргументи: a = 6, b = 2. Результат для c = '-' дорівнює 4
```

Цикл триває, поки значенням змінної з не стане пробіл. У кожній ітерації вказівник par отримує адресу однієї з функцій, і змінюється значення с. За результатами програми легко простежити порядок виконання її операторів.

можуть бути об'єднані в масиви. Наприклад, float (* ptrArray) (char) [4]; - опис масиву з ім'ям ptrArray з чотирьох вказівників на функції, кожна з яких має параметр типу char і повертає значення типу float. Щоб звернутися, например, до третьої з цих функцій, потрібно такий оператор:

```
float a = (*ptrArray[2]) ('f');
```

Як завжди, індексація масиву починається з 0, і тому третій елемент ма-

сиву має індекс 2.

Масиви вказівників на функції зручно використовувати при розробці всіляких меню, точніше програм, управління якими виконується за допомогою меню. Для цієї дії, пропоновані на вибір майбутнього користувачеві програми, оформляються у вигляді функцій, адреси яких містяться в масив вказівників на функції. Користувачеві пропонується вибрати з меню потрібний йому пункт і за номером пункту, як за індексом, з масиву вибирається відповідний адресу функції. Звернення функції до цією адресою забезпечує виконання необхідних дій. Найбільшу загальну схему реалізації такого підходу ілюструє наступна програма.

Приклад №3

```
/*  Приклад з масивом вказівників на функції
*/
#include<iostream.h>

// оголошення функцій
void func1(int);
void func2(int);
void func3(int);

void main()
{
/* оголошення масиву, в якому зберігаються вказівники на
функції func1, func2 и func3*/
        void (*f[3])(int) = {func1, func2, func3};
        int choice;

        cout<<"Введіти число між 1 і 3,"
        " друге число - закінчення: ";
        cin >> choice;

        while(choice >= 1 && choice < 4)
```

```
        {
                (*f[choice-1])(choice); // виклик функції f[choice]
                cout<<" Введіти число між 1 і 3,"
                " друге число - закінчення: ";
                cin >> choice;
        }
        cout<<"Ви ввели "<<choice<<" для закінчення\n";
}

// опис функцій
void func1(int a)
{
        cout<<"\nВи ввели "<<a
        << ", тому була викликана func1 \n";
}

void func2(int b)
{
        cout<<"\nВы ввели "<<b
        << ", тому була викликана func2\n";
}

void func3(int a)
{
        cout<<"\nВи ввели "<<a
        << ", тому була викликана func3\n";
}
```

Результат виконання програми

Введіть число між 1 і 3, інше число - закінчення: 1

Ви ввели 1, тому була викликана func1

Введіть число між 1 і 3, інше число - закінчення: 2

Ви ввели 2, тому була викликана func2

Введіть число між 1 і 3, інше число - закінчення: 3
Ви ввели 3, тому була викликана func3

Введіть число між 1 і 3, інше число - закінчення: 10
Ви ввели 10 для закінчення

У програмі визначено три функції - func1, func2, func3 - кожна з яких приймає цілий аргумент і нічого не повертає.Вказівн на ці три функції зберігаються в масиві f, який оголошений таким чином:

```
void (* f [3]) (int) = {func1, func2, func3};
```

Масив отримує в якості початкових значень імена 3-х функцій. Коли користувач вводить значення 1, 2 або 3, це значення (мінус 1) використовується в якості індексу в масиві вказівників на функції.

Виклик функції виконується наступним чином:

```
(*f[choice-1])(choice);
```

У цьому виклику (* f [choice-1]) визначає вказівник, розташований в елементі масиву з індексом choice-1. Вказівник разіменовує, щоб викликати функцію, і choice передається функції як аргумент. Кожна Функція друкує ім'я функції, щоб показати, що виклик вірний.

Частина 2
Практична частина

Лабораторна робота №1
Інтегроване середовище розробки code::blocks

Мета і зміст роботи:

Метою роботи є вивчення середовища програмування для мови програмування С++ і здобуття початкових навиків роботи в інтегрованому середовищі **Code::Blocks**.

Необхідно:

- □ вивчити інтегроване середовище **Code::Blocks**;
- □ вміти створювати консольний додаток;
- □ розробити консольний додаток;
- □ виконати його налагодження.

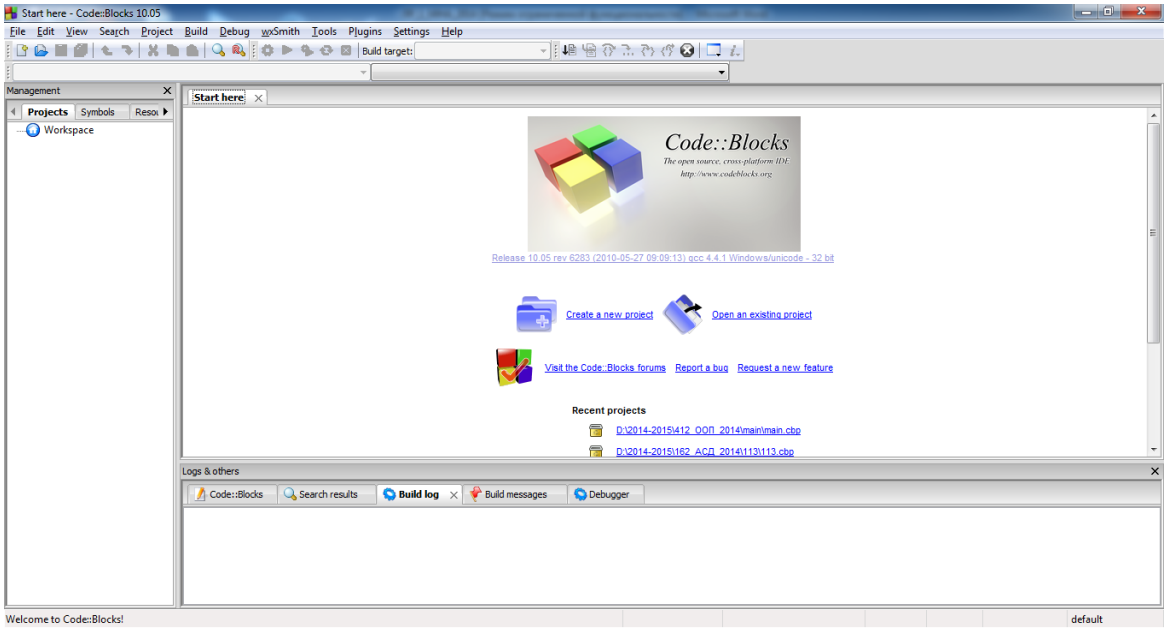
За теоретичним матеріалом лабораторної роботи та лекції 1 буде задано 5-7 питань для захисту.

Середовище програмування.

Середовище розробки являє собою інтегровані в єдину оболонку засоби, що дозволяють легко створювати, відкривати, переглядати, редагувати, зберігати, передавати і налагоджувати всі ваші програми на С та С++

Нижче буде даний огляд можливостей середовища **Code::Blocks**. У більшості випадків ви будете користуватися стандартними параметрами цього середовища. По мірі того, як буде зростати ваш досвід, ускладняться вимоги до програм, можна буде використовувати інші можливості цього середовища.

Запуск здійснюється зі стартового меню Windows командою **CodeBlocks** або ярликом на Робочому столі І ви побачите наступне вікно:



Загальні властивості меню

Перед обговоренням конкретних можливостей **CodeBlocks** розглянемо деякі загальні для всіх меню властивості. Наприклад, до кожного пункту меню можна дістатися кількома шляхами. Найпоширеніший – помістити курсор миші на потрібний пункт і натиснути ліву кнопку. Другий шлях – використовувати підкреслений символ у назві пункту. Ви можете, наприклад, звернутися до меню ***File***, натиснувши клавішу <ALT>+<F>. Ви можете виконувати команди меню з будь-якого місця середовища, натиснувши спеціально призначену комбінацію клавіш (таку комбінацію надалі будемо називати гарячою клавішею). Якщо для даного пункту меню така можливість є, позначення гарячої клавіші буде приведено праворуч від назви пункту меню. Наприклад, перший пункт меню ***File*** називається ***New***. Цю команду можна виконати безпосередньо, без звертання до меню, натиснувши комбінацію <CTRL> + <N>.

Якщо назва пункту зображено сірим кольором, це означає, що відповідна команда в даний момент нездійсненна. Середовище, таким чином, попереджає вас, що не виконано деяка обов'язкова умова. Наприклад, команда ***Save*** меню ***File*** недоступна, якщо вікно редактора порожнє.

Назва пункту, закінчується трьома крапками (...). вказує на те, що

при виборі команди з'являється вікно діалогу. Наприклад, якщо ви виберете команду ***Open...*** з меню ***File***, з'явиться вікно діалогу ***Open***.

Нарешті, ви можете виконати деякі команди меню, натиснувши відповідну кнопку панелі інструментів, яка розташовується нижче рядка меню.

Розглянемо тепер деякі корисні можливості середовища, доступні через меню.

Меню *File* (файл)

Містить стандартні команди для роботи з файлами, що зустрічаються в багатьох додатках *Windows*.

New (новий) – розгортає підменю для створення нового файлу, проекту, класу та ін. Пункту меню ***New*** відповідає кнопка на панелі інструментів із зображенням листа паперу.

Open...(відкрити) – призначена для відкриття уже існуючого і збереженого на диску файлу. Пункт викликає появу стандартного вікна діалога, у якому показані поточний пристрій, каталог і шаблон пошуку файлів, і пропонується ввести потрібні параметри.

Recent Files, Recent Project – список останніх файлів, проектів, що відкривалися.

Save (зберегти) – різні пункти меню дозволяють зберігати відповідну частину проекту, файли, що використовуються, чи все відразу.

Close (закрити) – пункти меню закриття служать для того, щоб закрити в серидовищі розробки файли, робочу область чи весь проект. Якщо ви випадково будете намагатися закрити не збережений файл, нічого страшного не відбудеться. Інтегроване середовище попередить, що файл ще не збережений и запитає, чи не хочете ви його зберегти.

Print... (друк) – друк вмісту активного вікна. Є можливість вибору друкування лише виділеного тексту. Для виділення тексту наведіть курсор миші на перший символ виділеного тексту і, утримуючи ліву кнопку натиснутою, перемістіть мишу до кінця ділянки, що виділяється.

Properties... (властивості) – показує властивості поточного файлу.

Quit (вихід) – завершує роботу середовища **CodeBlocks**. Якщо ви

забули зберегти якісь файли, середовище автоматично попередить і дасть можливість зберегти їх.

СТВОРЕННЯ КОНСОЛЬНОГО ДОДАТКУ

Консоль – це інтерфейс, який використовується програмою, що працює у текстовому режимі. Програма має вхідний і вихідний буфер. Вхідний буфер пов'язаний з клавіатурою, вихідний з екраном. Кожна програма, що працює в текстовому режимі, взаємодіє з Windows через консоль. Для консолі Windows автоматично створює вікно, яке має практично ті ж властивості, що і звичайне вікно Windows. Відповідно консольна програма – це додаток, що працює в текстовому режимі і не створює власних вікон.

Створення нового проекту

Створення будь-якої програми починається зі створення проекту. Проект містить імена вихідних файлів, що складають програму.

Для створення нового проекту виберіть  на панелі запуску **CodeBlocks**. Ця команда відкриває вікно діалогу **New form template** (тип додатку). Тут ви повинні вибрати тип створюваного додатка (Console Application) і натиснути кнопку **Go**. У наступному вікні в перший раз потрібно поставити відмітку у відповідному рядку і натиснути кнопку **Next** (більше воно з'являтися не буде). В наступному вікні вибираємо мову розробки програми (в нашому випадку С++). Далі вводимо в поле **Project title** ім'я проекту і вибираємо місце розташування папки створюваного проекту (у полі Folder to create project in), натискаючи на три крапки. Інші поля заповняться автоматично системою. Натискаємо кнопку **Next**. У наступному вікні вибирається компілятор (GNU Компілятор GCC) і натискаємо кнопку **Finish.** Проект створений.

Після виконання описаних дій зліва ви побачите дерево створеного проекту, в якому потрібно розгорнути папку **Sources** і відкрити файл **main.cpp**, який в ній знаходиться.

Введення тексту програми

У відкритому текстовому редакторі ви побачите текст готової програми. Запустіть її на виконання кнопкою на панелі інструментів Build and run

Якщо Ваш проект не був виконаний (не з'явилося консольне вікно з написом), то Вам потрібно прописати шлях до компілятору: Settings->Compiler and debugger... Відкриється діалогове вікно. Якщо в полі Selected compiler нічого немає, то натисніть Set as default (там повинен бути GNU Компілятор GCC). Далі в цьому вікні відкрийте вкладку Toolchain executables і в полі Compiler's installation directory, натиснувши на крапки виберіть шлях C:\Program Files (x86)\CodeBlocks\MinGW або інший залежно від того, куди Ви встановили середовище програмування. Натисніть кнопку Ok і після цього знову запустіть програму.

Якщо ви побачили консольне вікно з написом Hello world!, то середовище програмування працює правильно і в ньому можна створювати власні додатки.

Для цього очистіть вміст блоку int main() – видаліть два рядки між фігурними дужками.

Між фігурними дужками введіть текст програми:

```
int a,b;
cin>>a>>b;
cout<<a+b<<endl;
```

Наведений приклад містить функцію **main** і вона є точкою входу в програму. Потоки **cin, cout** відкриваються автоматично (підключена бібліотека <iostream>) при запуску програми і стають інтерфейсом (сполучною ланкою) між програмою і користувачем. Потік **cin** пов’язаний з клавіатурою. Потік **cout** – з монітором. Програма очікує від користувача введення двох цілих чисел і потім виводить на екран(в консольному вікні) їх суму.

!!!Зауваження. Перед переходом до наступного етапу роботи не забудьте зберегти файл із введеним текстом. Для цього виконайте команду **Save file** меню **File**. А краще **Save all projects** для збереження всіх файлів

проекту.

Наступне завдання для роботи з середовищем програмування Ви знайдете в кінці лабораторної роботи.

Створення файлу, що виконується

Процес створення файлу, який виконується додатком, що заключається в компіляції і компонуванні усіх модулей проекту, називається побудовою. Запустіть процес побудови командою **Build and run** меню **Build** чи на панелі інструментів кнопку  (*Build and run*).

Налагодження програми

Якщо програма містить синтаксичні помилки, при виконанні побудови вони автоматично відображаються у вікні *Logs&others* на вкладці *Build messages* або *Build log*, за замовчуванням розташованому в нижній частині вікна інтегрованої оболонки.

Причому справа, в кінці цього вікна є смуга прокрутки, піднявши її вгору, ми зможемо переглянути всі повідомлення системи про помилки (errors) і застереження (warning).

Кожне повідомлення починається з імені вихідного файлу і номера рядка, де виявлена помилка або попередження (у цьому ви переконаєтеся самі, коли почнете працювати зі своїм текстом програми). Слідом за номером рядка йде короткий опис попередження або помилки. Краще всього домогтися, щоб в остаточному варіанті не було ні того, ні іншого, хоча з попередженнями виконуваний файл створюється і може бути запущений.

Виправивши всі помилки, повторіть побудову файлу, що виконується.

Логічні помилки

Навіть якщо синтаксичних помилок немає, програма може працювати не так, як очікувалося. Такі помилки називаються логічними. Інтегрований налагоджувач середовища має різні засоби, які готові прийти вам на допомогу. Найбільш ефективні з них: виконання програми по кроках, пе-

регляд значення будь-яких змінних в будь-якій точці програми, задання точок зупину та ін.

У розглянутому прикладі програми логічні помилки є у функції виконання бульбашкового сортування (вона працює неправильно), тому можна скористатися допомогою налагоджувача при пошуку помилок у логіці цього фрагмента.

Корисні доповнення

Форматування коду. Код програми може бути введений програмістом в довільному вигляді. Ви можете записати всю програму в один рядок – довжина одного рядка в С++ не повинна перевищувати 1024 символи. Ви можете довільно розбити програму на рядки. Неприпустимий тільки розрив ідентифікаторів.

Але програма повинна не тільки коректно працювати, але і легко читатися, і не тільки вами, але і іншими програмістами. Необхідно відзначити, що *неформатований код також важко читати, як і текст без пропусків.* Тому код програми прийнято форматувати (оформляти) у відповідності з наступними правилами:

- кожен оператор записується в окремому рядку;
- оператори одного рівня вкладеності записуються один під одним з однаковим відступом;
- кожен складовий оператор містить «карниз» (встановлюється знаком табуляції) після відкритої фігурної дужки;
- кожна закрита фігурна дужка розташована на одну позицію табуляції ближче до лівого краю сторінки, ніж попередній оператор;
- в IDE CodeBlocks є можливість автоматичного форматування коду – потрібно натиснути праву кнопку миші на тексті програми і вибрати пункт меню Format use AStyle

Виконання цих правил дозволяє легко бачити структуру програми – підпорядкованість операторів. Крім того, правильне форматування дозво-

ляє легко уникнути самої фатальної для компілятора помилки – невідповідність кількості відкритих і закритиї фігурних дужок. Відсутня або зайва фігурна дужка призводить до порушення десятків правил. У цьому випадку закрита фігурна дужка, що відповідає кінцю складеного оператора, буде розташовуватися строго під фігурною дужкою, що відповідає початку складеного оператора.

Табуляція. Установку «карнизів» в програмі можна виконати шляхом введення відповідної кількості пробілів. Але більш зручно та ефективно використовувати для цих цілей символ табуляції по цілому ряду причин:

- зменшується час введення коду;
- зменшується розмір вихідного файлу, так як символ табуляції займає менше пам'яті, ніж кілька пробілів;
- якщо ви закоментуєте рядок коду шляхом введення символів // на початку рядка, рядок не зрушиться вправо, що покращує зовнішній вигляд програми;
- при копіюванні коду в текстовий редактор, його легко відформатувати за допомогою спеціальних стилів;

За замовчуванням символ табуляції відповідає 4 пробілам. Але ви можете змінити розмір на свій розсуд.

У середовищі програмування CodeBlocks код можна відформатувати автоматично *Plugins->Source code formatted*.

Виконання програми. Для налагодження, компіляції та виконання програми служать пункти меню **Build**:

Перед виконанням будь-якої з перерахованих вище команд всі файли проекту зберігайте на жорсткому диску. Хід компіляції реєструється у відповідних журнальних файлах, які зберігаються у папці **Debug**. Якщо компіляція виконана успішно, то в цій же папці зберігається і запускається на виконання виконуваний файл.

Необхідно відзначити, що розмір журнальних файлів, що зберігаються в папці **Debug**, може досягати декількох Мбайт. Тому після завершення роботи з проектом краще видалити папку **Debug**. При наступному

запуску папка з журнальними файлами буде створена знову.

Повторний запуск програми. Для відновлення роботи з проектом виконайте наступні дії: двічі клацніть на файлі проекту (файл з розширенням ***.cbp**). Після цього автоматично завантажиться програма CodeBlocks і всі файли, включені раніше в проект.

Робота із синтаксичними помилками

Пошук синтаксичних помилок. Після запуску програми на компіляцію або виконання компілятор, як правило, знаходить помилки в коді і припиняє свою роботу. При цьому на екрані відображається вікно *Logs&others* (навіть якщо воно було приховано) зі списком знайдених компілятором помилок.

Приклади характерних синтаксичних помилок.

Як неважко бачити, кількість типів помилок, обумовлених компілятором, досягає декількох тисяч. Для прикладу наведемо опис кількох помилок, що здійснюються програмістами-початківцями:

- **undeclared identifier** – неоголошений ідентифікатор. Ви забули оголосити змінну або функцію, помилилися в імені змінної або забули підключити бібліотеку, в якій оголошена функція;
- **missing ';'** – пропущена крапка з комою;

При пошуку синтаксичних помилок необхідно мати на увазі, що в багатьох випадках компілятор не може точно вказати, як саме ви помилилися. Як правило, компілятор наводить список правил, які ви порушили. Один неправильний або відсутній символ в коді програми може викликати кілька десятків помилок. Тому доцільно видаляти помилки по одній.

Попередження. Порушення деяких правил не носить фатальний характер і компілятор в змозі інтерпретувати написаний вами код. У цьому випадку компілятор виводить у вікні *Logs&others* попередження (**warning**), які носять рекомендаційний характер. Незважаючи на те, що програма не втрачає працездатності. рекомендується усувати ці попередження, хоча б для того, щоб зменшити розмір шрифту у вікні *Logs&others*.

Наведемо опис деяких попереджень, які з'являються в програмі початківця програміста:

- **conversion from 'double' to 'int', possible loss of data** – перетворення дійсного числа подвійної точності в ціле, можлива втрата точності. Використовуйте в правій і лівій частині вираз одного типу або явно викличте функцію перетворення типу;
- **unreferenced local variable** – невживана локальна змінна. Видаліть оголошення змінної, якщо вона вам не потрібна;
- **local variable used without having been initialized** – локальна змінна використовується без ініціалізації. Ви забули присвоїти змінній значення. У цьому випадку в розрахунках використовується випадкове значення, яке було в комірці пам'яті перед запуском програми.

Завдання до лабораторної роботи.

Створіть новий проект і наберіть в ньому наведений нижче приклад. Якщо ви знайомі з мовою С++, то побачите наявність помилок в програмі. Не виправляйте їх одразу. Помилки зроблені спеціально для того, щоб на практиці продемонструвати різні можливості інтегрованого середовища розробки.

```
// консольний додаток
// програма містить помилки!!!

#include<iostream.h>
#define SIZE 5

void bsort(int iArray[], int n);
void main()
   {
      char ch;
      char ii;
      int iArray[SIZE] ;
```

```
    for(ii = 0; ii<SIZE; ii + + ;)
            {
                cout << "Please enter an integer: ";
                cin >> iArray[ii];
            }
    cout << "\n Would you like to sort (Y/N) ";
    cin >> ch
    if (ch=='Y' || ch=='y')
            {
                bsort(iArray, SIZE);
            }
    for(ii=0; ii<SIZE; ii++;)
            {
                cout  <<   iArray[ii]   <<   "     ";
            }
  }
void bsort(int iArray[], int n)
// Алгоритм бульбашкового сортування - BubbleSort
  {
    int i,j,k,t;
    for(i=0; i < n; i++)
            {
                j=i;
                for(k=j+l; k < П; k++)
                    {
                        if(iArray[k]   <=   iArray[j])
                            {
                               j=k;
                            }
                    }
                if (i  <>  j)
                    {
```

```
                    t=iArray[j];
                    iArray [j] =iArray [i] ;
                    iArray[i]  =  t;
                }
            }
        }
```

Виправте помилки та досягніть її повного і правильноговиконання.

Зробіть висновки про дану програму: що є вхідними даними, результатами, від чого можуть виникнути помилки, що вона робить, які алгоритмічні структури застосовуються, які типи задіяні.

Лабораторна работа №2
Розробка і виконання програм простої структури

Мета і зміст роботи:

Створення, налагодження та виконання простої програми, що містить введення/виведення інформації і найпростіші обчислення, робота з операціями та стандартними функціями С++.

В результаті виконання роботи необхідно:

- знати загальну структуру програми;
- вміти створювати, компілювати та виконувати налагодження своїх програм простої структури на консолі;
- знати операції в С++, правила їх виконання і пріоритет виконання;
- оперувати типами величин;
- знати і використовувати стандартні функції в С++.

Методичні вказівки

1. Для введення і виведення даних використовувати операції >> і << і стандартні потоки

cin і **cout.**

2. Для обчислень використовувати стандартні функції з бібліотечного файлу **math.h.**

3. При виконанні завдань використовувати допоміжні змінні для зберігання проміжних

результатів.

Зміст звіту

Звіт оформляється в окремому зошиті або в надрукованому вигляді. Він повинен містити:

1. Номер роботи, тему і постановку задачі.

2. Програму вирішення завдання 1. Змінні *a* і *b* вводяться користувачем, на екран виводиться значення виразу.

3. Результати роботи програми для трьох різних наборів даних різних типів **int, float, double** оформлені у вигляді таблиці.

4. Порівняння результатів при використанні різних типів і висновки про використання різних типів даних.

5. Програму вирішення завдання 2. Змінні *m* і *n* вводяться користувачем. На екран виводиться результат розрахунку і значення змінних до розрахунку і після розрахунку.

6. Результати роботи програми для трьох різних наборів даних, серед яких є набір *m=5 і n=8*.

7. Записати висновки про порядок дій і пояснення отриманих результатів.

8. Математичну модель завдання 3. Перерахувати вхідні, проміжні та вихідні змінні.

9. Програму розв'язання завдання 3.

10. Результати роботи програми для п'яти різних наборів даних.

11. Прийом завдання проводиться в наступному порядку:

- написаний або надрукований звіт по роботі (див. вище);
- працюють, повністю налагоджені програми, що містять докладні коментарі та збережені на флеш-карту;
- усні відповіді на 5-7 питань лекції, задані викладачем під час захисту.

Постановка задачі

1. Обчислити значення виразу при різних речових типах даних (**float і double**).

Обчислення слід виконувати з використанням проміжних змінних. Порівняти і пояснити отримані результати.

2. Обчислити невідому величину і вивести результати, використовуючи потоки введення-виведення даних. Обов'язково організувати виведення на екран монітора необхідних пояснень і коментарів.

3. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично.

Завдання по варіантам

№ варианта	Задание 1	Задание 2	Задание 3
1	$\frac{(a+b)^2-(a^2+2ab)}{b^2}$	n+++m	Знайти площу трикутника за трьома заданими сторонами.
2	$\frac{(a-b)^2-(a^2-2ab)}{b^2}$	++n*++m	Знайти радіус окружності, що описана навколо рівнобедреного трикутника за двома його заданими сторонами.
3	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$	n---m	Знайти радіус окружності, вписаної в різносторонній трикутник.
4	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$	n++*m	Знайти косинуси кутів трикутника за трьома заданими сторонами.
5	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$	--m-++n	Знайти площу трикутника за двома заданими сторонами та заданим кутом між ними.
6	$\frac{(a-b)^3-(a^3-3ab^2)}{b^3-3a^2b}$	m-++n	Знайти скалярний добуток векторів, що задані своїми координатами.
7	$\frac{(a-b)^3-(a^3)}{b^3-3ab^2-3a^2b}$	m+--n	Знайти площу трикутника, який заданий координатами своїх вершин в прямокутній системі координат.
8	$\frac{(a+b)^4-(a^4+4a^3b+6a^2b^2)}{4ab^3+b^4}$	n++-m	Знайти площу ромба по заданим діагоналям.
9	$\frac{(a+b)^4-(a^4+4a^3b)}{6a^2b^2+4ab^3+b^4}$	++n*--m	Знайти суму нескінченної геометричної прогресії по заданому першому члену і знаменнику.

10	$\frac{(a-b)^4-(a^4-4a^3b+6a^2b^2)}{b^4-4ab^3}$	n---++m	Знайти площу трапеції за заданими основами і висотою.
11	$\frac{(a-b)^4-(a^4-4a^3b)}{6a^2b^2-4ab^3+b^4}$	n--*m	Знайти висоту ромба по заданим діагоналям.
12	$\frac{(a+b)^2-(a^2+2ab)}{b^2}$	--m-++n	Знайти суму перших 20 членів арифметичної прогресії по заданому першому члену і різниці.
13	$\frac{(a-b)^2-(a^2-2ab)}{b^2}$	m++-++n	Обчислити периметр ромба по заданим діагоналям.
14	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$	++m+--n	Обчислити площу круга і його довжину по заданому радіусу.
15	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$	++n+++m	Обчислити п’яту степінь суми двох зааних чисел.
16	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$	n--*++m	Обчислити висоту трикутника, знаючи його площу і одну зі сторін.
17	$\frac{(a-b)^3-(a^3-3ab^2)}{b^3-3a^2b}$	n---m--	Обчислити різницю шостих степеней двох заданих чисел.
18	$\frac{(a-b)^3-(a^3)}{b^3-3ab^2-3a^2b}$	n--*--m	Обчислити об’єм циліндра за заданим радіусом основи і висотою.
19	$\frac{(a+b)^4-(a^4+4a^3b+6a^2b^2)}{4ab^3+b^4}$	+m-++n	Обчислити об’єм паралелепіпеда за трьома заданими його вимірами.
20	$\frac{(a+b)^4-(a^4+4a^3b)}{6a^2b^2+4ab^3+b^4}$	--m-n--	Обчислити площу повної поверхні прямокутного паралелепіпеда по трьом заданим його вимірам.
21	$\frac{(a-b)^4-(a^4-4a^3b+6a^2b^2)}{b^4-4ab^3}$	++-m++	Обчислити середнє геометричне трьох заданих чисел.

22	$\frac{(a-b)^4-(a^4-4a^3b)}{6a^2b^2-4ab^3+b^4}$	++n+m++	Обчислити об'єм піраміди, в основі якої лежить трикутник, відомо три його сторони і висота піраміди.
23	$\frac{(a+b)^3-(a^3+3a^2b)}{3ab^2+b^3}$	++n*m--	Обчислити площу повної поверхні конуса за заданими площею основи і висотою конуса.
24	$\frac{(a+b)^3-(a^3)}{3ab^2+b^3+3a^2b}$	n++*--m	Обчислити об'єм конуса за заданими радіусом і твірною.
25	$\frac{(a-b)^3-(a^3-3a^2b)}{b^3-3ab^2}$	--m*++n	Очислити площу повної поверхні циліндра за заданими довжиною кола основи та твірною.

Лабораторна робота №3
Розробка та виконання програм з операторими розгалуження

Мета і зміст роботи:

Створення, налагодження та виконання програм, що містять розгалуження та вибір, робота з вкладеними розгалуженнями, вивчення різних форм запису операторів вибору.

В результаті виконання роботи необхідно:

- вміти застосовувати оператор розгалуження if в повній та скороченій формах;
- вміти створювати, компілювати та виконувати налагодження своїх програм;
- знати оператори вибору в С++, правила їх виконання;
- оперувати типами величин і розуміти правила перетворення типів;
- знати логічні операції та правила їх виконання.

Методичні вказівки

1. У програмах використовувати стандартну та альтернативну форми запису оператора розгалуження **if.**
2. У задачах, де потрібно розглянути більше трьох умов, використовувати оператор множинного вибору **switch.**
3. При виконанні завдань використовувати детальні коментарі.

Зміст звіту

1. Номер роботи, тему і постановку задачі.
2. Програму вирішення завдання 1. На екран вивести відповідні коментарі з поясненням
   дій, початкові і кінцеві значення змінних.
3. Результати роботи програми для трьох різних наборів даних.
4. Програму вирішення завдання 2, використовуючи структуру **switch.**
5. Результати роботи програми для трьох наборів даних.
6. Програму розв'язання завдання 3.

7. Результати роботи програми для п'яти різних наборів даних.
8. Прийом завдання проводиться в наступному порядку:
   - написаний або надрукований звіт по роботі (див. вище);
   - працюють, повністю налагоджені програми, що містять докладні коментарі та збережені на flash-карту;
   - відповіді на 5-7 питань лекції 3.

Постановка задачі

Створити додаток консолі і в ньому набрати і виконати програму. Дані описувати відповідно до вказаних у варіанті типів. Слідкуйте за типами отриманих результатів. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті. Програму тестувати на різних наборах даних і робити висновки про отримані результати. Для тестування програми вибирати варіанти, що відповідають різним ситуаціям.

ЗАВДАННЯ 0

Завдання виконується в зошиті (або на листочку) і є допуском до виконання лабораторної роботи. Студент виконує ВСІ задачі даного завдання. У розв’язку записувати лише оператор розгалуження (не потрібно писати повний текст програми)

1. Виведіть на екран більше з двох чисел.
2. Знайдіть менше з двох чисел в окрему змінну.
3. Перевірте задане число на парність.
4. Перевірте чи є задане число кратним номеру Вашого варіанту.
5. Перевірте чи є задане число меншим від поточного року.
6. Перевірте чи є задане число двозначним.
7. Перевірте чи є заданий символ літерою латинського алфавіту.
8. Перевірте чи є задане число розв’язком нерівності $x^2+5x-6>0$.
9. Чи належить задане число одному з проміжків $(-\infty,-10]\cup[5, 100]\cup(1000,+\infty)$.
10. Чи належить число проміжку $(-100,10]$.

ЗАВДАННЯ 1 (по варіантам)

1. Задано два довільних дійсних числа. Перевірити, чи є одне число квадратом іншого. Якщо так, то вивести повідомлення, яке з чисел відповідає умові. В іншому випадку менше з них замінити їх сумою, а більше – їх різницею. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

2. Задано два довільних цілих числа. Якщо вони не рівні між собою, то менше з чисел замінити половиною їх суми, а більше – подвоєним добутком. Якщо вони рівні одне одному, то обчислити значення натуральних логарифмів цих чисел. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

3. Задано три дійсних числа. Якщо серед них є такі, що дорівнюють один одному, замінити ці числа їх квадратами, інші – синусами цих величин. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

4. Задано три цілих числа. Знайти середнє арифметичне найбільшого та найменшого з цих чисел. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

5. Задано три дійсних числа. Знайти квадрат добутку найбільшого та найменшого за модулями з цих чисел. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

6. Задано три дійсних числа. Якщо добуток цих чисел не перевищує 200, замінити найбільше число на суму двох інших, у іншому випадку найменше число замінити на різницю двох інших. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

7. Задано три цілих числа. Якщо сума цих чисел менша 50, вивести ці числа, у порядку зростання, у противному разі – у порядку спадання. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

8. Задано три дійсних числа. Якщо числа введені в порядку зростання, замінити кожне з них на суму двох інших, у іншому разі вивести суму та добуток всіх трьох чисел. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

9. Задано три цілих числа. Знайти серед них те, що не є ні найбільшим, ні найменшим, перевірити чи є воно точним квадратом, найбільше число перевірити на парність, а найменше замінити сумою двох інших. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

10. Задано три цілих числа. З'ясувати, чи утворюють ці числа, розташовані в порядку спадання, арифметичну прогресію? Якщо це так, то знайти її різницю і суму 20 перших її членів, а якщо ні, то найбільше число перевірити на парність і якщо воно парне, то найменше з них замінити на квадрат третього, а якщо ні, то середнє за значенням замінити на його квадратний корінь. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

11. Задано три цілих числа. З'ясувати, скільки серед них пар, що дають в сумі парне число і яке з чисел є найменшим, а яке найбільшим. На екран вивести відповідні коментарі з поясненням дій.

12. Порівняти різницю модулів найбільшого та найменшого з трьох цілих чисел з їх середнім арифметичним та середньому з них за значенням присвоїти суму середнього двох інших. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

13. Задано три цілих числа. Знайти серед них парні числа і замінити їх нулями, а непарні числа замінити вдвічі більшими і всі результати вивести в порядку спадання. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

14. Створити програму визначення, чи є задане ціле число парним числом і яким – двозначним, тризначним чи іншим. Якщо парне двозначне, то зменшити його вдвічі, якщо парне тризначне – збільшити на 11, а якщо інше, то знайти його квадрат. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

15. Створити програму, за допомогою якої можна з'ясувати, чи утвориться із заданих чотирьох чисел арифметична прогресія, геометрична прогресія або зовсім не утворюється. Якщо якась утвориться, то знайти її різницю або знаменник та суму перших десяти членів, а якщо не утвориться ніяка, то замінити всі введені числа на вдвічі менші. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

16. Для чотирьох цілих чисел знайти їх середнє арифметичне та середнє геометричне модулів цих чисел. Якщо середнє арифметичне непарне і двозначне, то всі задані числа збільшити втричі. Якщо середнє геометричне більше від 10, то всі задані числа зменшити вдвічі. В будь-якому іншому випадку замінити всі задані числа на суму трьох інших. На екран вивести відповідні коментарі з поясненням дій, початкові та кінцеві значення змінних.

ЗАВДАННЯ 2 (по варіантам)

1. По введеному номеру місяця від 1 до 12 вивести на екран назву відповідної пори року (1,2,12 – зима, 3,4,5 – весна, …).

2. По введеному числу від 1 до 7 вивести на екран відповідний день тижня (1 – понеділок, 2 – вівторок, …).

3. По введеному числу від 0 до 9 вивести на екран назву відповідного факультету (інституту) (0 – біологічний, 1 – філологічний, 2 – механіко-математичний, …).

4. Вивести на екран інформацію залежно від порядкового номера місяця, що виражається цілим числом від 1 до 12, про кількість днів цього місяця.

5. Вивести на екран інформацію залежно від порядкового номера місяця, що виражається цілим числом від 1 до 12, про його назву.

6. За введеним цілим числом від 1 до 5 спрогнозувати характер подій для користувача на сьогодні (1 – день буде вдалим, 2 – передбачається подорож …). Якщо введене число не відповідає заданому діапазону, виводиться пропозиція повторити арифметику.

7. За введеним цілим числом від 1 до 6, яке позначає вік дитини, вивести на екран назву групи дитячого садка, у яку ця дитина потрапить (1-3 – ясельна, 3-4 – молодша, 4-5 – середня, 5-6 – старша). Якщо введене число не відповідає заданому діапазону, то виводиться відповідна інформація – ця дитина або зовсім мала, або вже доросла.

8. За введеним цілим числом від 1 до 4 вивести на екран прізвище, ім'я, по батькові відповідного президента сучасної України, а від 5 до 10 – прізвище, ім'я, по батькові прем'єр-міністрів в хронологічному порядку. Якщо введене число не відповідає заданому діапазону, виводиться інформація що такого не існує.

9. За введеним цілим числом вивести на екран літеру свого прізвища. Якщо введене число не відповідає ніякій літері, виводиться інформація що такої літери не існує.

10. За введеним цілим числом від 1 до 5 вивести елементну базу комп'ютера відповідного покоління. Якщо введене число не відповідає ніякому поколінню, виводиться інформація що такого покоління не існує.

11. За введеним цілим числом від 1 до 4 вивести відповідну пору року. Якщо введене число не відповідає ніякій порі, виводиться інформація що такої пори не існує.

12. За введеним цілим числом від 1 до 7 вивести відповідний рівень навчання (бакалавр, спеціаліст, магістр). Якщо введене число не відповідає ніякому рівню, виводиться інформація що такого покоління не існує.

13. За введеним цілим числом від 1 до 8 вивести назву планети за їх віддаленістю від сонця (1 – Меркурій, 2 – Венера, …). Якщо введене число не відповідає ніякій планеті, виводиться інформація що такого покоління не існує.

14. Визначити рівень успішності учня, в залежності від введеного значення середнього балу з усіх предметів цього учня, за такою схемою: якщо середній бал не менше 10 але не більше 12 – рівень «відмінний», якщо середній бал не менше 7 але не більше 9 – «добрий», якщо середній бал не менше 4 але не більше 6 – «задовільний», якщо середній бал не менше 1 але не більше 3 – «незадовільний».

15. Вивести на екран власну словесну оцінку погоди (холодно, прохолодно, тепло, спекотно) залежно від температури, що виражається цілим числом градусів.

16. Визначити агрегатний стан води за температурою, що виражається цілим числом.

17. Відповідно до віку клієнта дозволити чи заборонити йому придбати вироби з відповідними поясненнями: тютюнові (18 років та старший – дозволити, молодший 18 років – заборонити); алкогольні (21 рік або старший – дозволити, молодший 21 року – заборонити).

18. За віком людини, що виражається цілим числом, дати назву періоду її життя (дитинство – до 14 років, юність – до 21, зрілість – до 65, похилий вік – після 65).

19. Залік з інформатики містить 36 задач. При розв’язанні всіх задач ставлять оцінку «5», при розв’язанні 35, 34 або 33 задач – «4», при розв’язанні не менше 18 і не більше 32 задач – «3», а при розв’язанні менше 18 задач – оцінку «2». По введеному значенню кількості правильно розв’язаних задач виставити відповідну оцінку.

20. По введеному року від 1950 до 2020 вивести на екран назву відповідної назви року по східному календарю (1 – миша, 2 – бик, 3 – тигр, 4 – кіт, 5 – дракон, 6 – змія, 7 – кінь, 8 – вівця, 9 – мавпа, 10 – півень, 11 – собака, 12 – кабан).

ЗАВДАННЯ 3 (по варіантам)

Обчислити значення функції F залежно від значень сталих a, b, c, та змінної x.

1. $F = \begin{cases} ax^2 + b & \text{, де } x < 0 \text{ та } b \neq 0 \\ \dfrac{x-a}{x-c} & \text{, де } x > 0 \text{ та } b = 0 \\ \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

2. $F = \begin{cases} \dfrac{1}{ax} - b & \text{, де } x + 5 < 0 \text{ та } c = 0 \\ \dfrac{x-a}{x} & \text{, де } x + 5 > 0 \text{ та } c \neq 0 \\ \dfrac{10x}{c-4} & \text{, інші випадки} \end{cases}$

3. $F = \begin{cases} ax^2 + b + c, & \text{де } a < 0 \text{ та } c \neq 0 \\ \dfrac{-a}{x-c} & \text{, де } a > 0 \text{ та } c = 0 \\ a(x+c), & \text{інші випадки} \end{cases}$

4. $F = \begin{cases} -ax - c & \text{, де } c < 0 \text{ та } x \neq 0 \\ \dfrac{x-a}{-c} & \text{, де } c > 0 \text{ та } x = 0 \\ \dfrac{bx}{c-a} & \text{, інші випадки} \end{cases}$

5. $F = \begin{cases} a - \dfrac{x}{10+b} & \text{, де } x < 0 \text{ та } b \neq 0 \\ \dfrac{x-a}{x-c} & \text{, де } x > 0 \text{ та } b = 0 \\ 3x + \dfrac{2}{c} & \text{, інші випадки} \end{cases}$

6. $F = \begin{cases} ax^2 + b^2x & \text{, де } c < 0 \text{ та } b \neq 0 \\ \dfrac{x+a}{x+c} & \text{, де } c > 0 \text{ та } b = 0 \\ \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

7. $F = \begin{cases} -ax^2 - b & \text{, де } x < 5 \text{ та } c \neq 0 \\ \dfrac{x-a}{x} & \text{, де } x > 5 \text{ та } c = 0 \\ -\dfrac{x}{c} & \text{, інші випадки} \end{cases}$

8. $F = \begin{cases} -ax^2 & \text{, де } c < 0 \text{ та } a \neq 0 \\ \dfrac{a-x}{cx} & \text{, де } c > 0 \text{ та } a = 0 \\ \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

9. $F = \begin{cases} ax^2 + b^2x & \text{, де } a < 0 \text{ та } x \neq 0 \\ x - \dfrac{a}{x-c} & \text{, де } a > 0 \text{ та } x = 0 \\ 1 + \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

10. $F = \begin{cases} ax^2 - bx + c & \text{, де } x < 3 \text{ та } b \neq 0 \\ \dfrac{x-a}{x-c} & \text{, де } x > 3 \text{ та } b = 0 \\ \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

11. $F = \begin{cases} ax^2 + \dfrac{b}{c} & \text{, де } x < 1 \text{ та } x \neq 0 \\ \dfrac{x-a}{(x-c)^2} & \text{, де } x > 1{,}5 \text{ та } c = 0 \\ \dfrac{x^2}{c^2} & \text{, інші випадки} \end{cases}$

12. $F = \begin{cases} ax^3 + b^2 + c & \text{, де } x < 0{,}6 \text{ та } b + c \neq 0 \\ \dfrac{x-a}{x-c} & \text{, де } x > 0{,}6 \text{ та } b + c = 0 \\ \dfrac{x}{c} + \dfrac{x}{a} & \text{, інші випадки} \end{cases}$

13. $F = \begin{cases} ax^2 + b & \text{, де } x - 1 < 0 \text{ та } b - x \neq 0 \\ \dfrac{x-a}{x} & \text{, де } x - 1 > 0 \text{ та } b + x = 0 \\ \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

14. $F = \begin{cases} -ax^3 - b & \text{, де } x + c < 0 \text{ та } a \neq 0 \\ \dfrac{x-a}{x-c} & \text{, де } x + c > 0 \text{ та } a = 0 \\ \dfrac{x}{c} + \dfrac{c}{x} & \text{, інші випадки} \end{cases}$

15. $F = \begin{cases} a - x^2 + b & \text{, де } x < 0 \text{ та } b \neq 0 \\ \dfrac{x}{x - c} + 5{,}5 & \text{, де } x > 0 \text{ та } b = 0 \\ \dfrac{x}{-c} & \text{, інші випадки} \end{cases}$

16. $F = \begin{cases} a(x + c)^2 - b & \text{, де } x = 0 \text{ та } b \neq 0 \\ \dfrac{x - a}{-c} & \text{, де } x = 0 \text{ та } b = 0 \\ a + \dfrac{x}{c} & \text{, інші випадки} \end{cases}$

Лабораторна робота №4
Розробка та виконання програм з оператором умови та операторами циклів

Мета та зміст роботи:

Створення та виконання програм, що містять оператори розгалуження та циклів.

В результаті виконання роботи необхідно:

- вміти застосовувати оператор умови **if** в повній та скороченій формах;
- вміти створювати, компілювати та виконувати відладку своїх програм;
- знать оператори циклів в С++ та правила їх використання;
- оперувати типами величин і розуміти правила перетворення типів;

Постановка задачі

Створити консольний додаток і в ньому створити і виконати програму. Дані описувати у відповідності до вказаних у варіанті типах. Слідкуйте за типами отриманих результатів. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені у звіті. Програму тестувати на різних наборах даних і зробити висновки про отримані результати. Для тестування програми обирати варіанти, які відповідають різним ситуаціям.

ЗАВДАННЯ 1 (по варіантам)

У всіх варіантах потрібно вивести логічне значення True, якщо приведене висловлення для запропонованих вихідних даних є істиною, і значення False у протилежному випадку.

Всі числа, для яких зазначено кількість цифр (двозначне число, тризначне число і т.д.), вважати цілими додатніми.

1. Задано два цілих числа: A, B. Чи є кожне з чисел A і B непарним
2. Задано ціле число A. Чи є A невід'ємним або непарним.
3. Задано два цілих числа: A, B. Чи справедливо, що $A \geq 0$ або $B < -2$.
4. Задано два цілих числа: A, B. Чи справедливо, що $A > 2$ та $B \geq 3$.
5. Задано три цілих числа: A, B, C. Чи відповідають вони умові $A > B > C$.
6. Задано три цілих числа: A, B, C. Чи є хоча б одне з чисел A, B, C від'ємним.
7. Задано два цілих числа: A, B. Перевірити істинність висловлення: «Тільки одне з чисел A і B непарне».
8. Чи є серед трьох заданих цілих чисел хоча б одна пара рівних.
9. Задано ціле число A. Перевірити істинність висловлення: «Число A є тризначним та парним».
10. Задано три цілих числа: A, B, C. Чи є тільки одне з них непарним.
11. Задано ціле число. Чи є воно непарним двозначним.
12. Задано ціле число. Чи є воно тризначним або непарним.
13. Задано два цілих числа: A, B. Чи мають числа A і B однакову парність.
14. Перевірити істинність висловлення: «Серед трьох заданих цілих чисел є хоча б одна пара взаємно протилежних».
15. Задано три цілих числа: A, B, C. Перевірити істинність висловлювання: «Кожне з чисел A, B, C парне».
16. Задано два цілих числа: A, B. Перевірити істинність висловлення: «Хоча б одне з чисел A і B непарне».
17. Задано три цілих числа: A, B, C. Чи знаходиться число B між числами A і C.
18. Задано три цілих числа: A, B, C. Перевірити істинність висловлювання: «Тільки два з чисел A, B, C є додатніми.
19. Задано числа x, y. Перевірити чи належить точка з координатами (x, y) другій або третій координатній чверті.

ЗАВДАННЯ 2 (по варіантам)

1. Дано два цілих числа A і B (A < B). Вивести в порядку зростання всі цілі числа, розташовані між A и B (включаючи самі числа A и B).
2. Дано два цілих числа A и B (A < B). Вивести на екран всі непарні числа, розташовані між A и B (включаючи самі числа A и B).
3. Вивести на екран куби чисел от A до B.
4. Даны два целых числа A и B (A < B). Вывести в порядке убывания все целые числа, расположенные между A и B (не включая сами числа A и B), а также количество N этих чисел.
5. Даны два целых числа A и B (A < B). Вывести все круглые числа (которые заканчиваются на 0) , расположенные между A и B (включая при необходимости сами числа A и B).
6. Дано цілі числа K и N (N > 0). Вивести N раз число K.
7. Вывести на экран квадраты чисел от A до B.
8. Вывести на экран все числа от A до B, кратные заданому N.
9. Напечатать все двухзначные числа, сумма цифр которых равна заданному n (n<15).
10. Дано натуральное число *n*. Вывести на экран все делители этого числа.
11. Даны два целых числа A и B (A < B). Вывести все четные числа, расположенные между A и B (включая при необходимости сами числа A и B).
12. Вивести на екран K степенів числа N.
13. Вивести на екран всі тризначні числа, добуток цифр яких дорівнює заданому n (n<50).
14. Вивести на екран всі тризначні числа кратні заданому n (n<70).
15. Вивести на екран всі числа від A до B, які є точними квадратами (включаючи самі числа A і B) .

ЗАВДАННЯ 3 (по варіантам)

Вивести на екран таблицю значень заданої функції для діапазону аргументу від a до b з кроком h. Таблиця повинна містити заголовок та значення аргументу і значення функції. На екрані таблиця має виглядати естетично.

1. $Y=\sin x + \lvert x\rvert + 2^x$
2. $Y=\sin x^{1/2} + e^x - 3$
3. $Y=ab + \sin^2 x – x1/2$
4. $Y=x^3 + x^{1/2} – 3c$
5. $Y=tgx^2 + x - 3$
6. $Y=x^{1/2} + \cos x - 3$
7. $Y=\ln x^2 + x^2 + 2$
8. $Y=\cos x^2 + \sin^2 x + 2$
9. $Y=\cos x + \ln x - e^x$
10. $Y=e^x + \lvert x\rvert + x^2$
11. $Y=x^3 + \ln\lvert x\rvert - 3$
12. $Y=tgx + x^{1/2} + 2$
13. $Y=x^5 + 2x^2 - 3$
14. $Y=x^{1/2} + 3\lvert x\rvert + x^2$
15. $Y= \cos^2 x + \ln x + 2$
16. $Y= x^3 + 2\ln\lvert x\rvert + 3$
17. $Y=\sin^2 x + x^3 + \lvert x\rvert$

Лабораторна робота №5
Розробка і виконання програм, що містять оператори циклу

Мета і зміст роботи:

Створення, відладка і виконання програм, що містять різні види циклів, робота з вкладеними циклами, вивчення різних форм запису операторів циклу.

В результаті виконання роботи необхідно:

- Вміти застосовувати оператори циклів;
- Вчитися обирати найбільш оптимальний варіант з трьох операторів циклів;
- Знати оператори циклів в С++ і правила їх виконання;
- Вчитися правильно будувати вирази-умови, використовуючи логічні операції;
- Вміти виконувати трасування програми і пояснювати принцип роботи програми.

Методичні вказівки

4. В трьох програмах до трьох задач використати всі три види циклів.
5. В третій задачі доцільно прорахувати скільки варіантів можливо при кожному розподілі, а потім вирішувати які вкладені цикли ви будете використовувати і яким чином. Обов’язково виводити на екран кожний можливий варіант і підрахувати загальну кількість можливих варіантів.
6. Всі програми мають містити коментарі для користувача та дані про розробника.
7. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті.

Зміст звіту.

1. Номер роботи, тему і постановку задачі.
2. Програму розв'язку завдання 1.
3. Результати роботи програми для трьох різних наборів даних.
4. Програму розв'язку завдання 2.
5. Результати роботи програми.
6. Програму розв'язку завдання 3.
7. Результати роботи програми для різних наборів даних, включаючи те дане, що наведено в умові
8. Зробити висновок про зручність та переваги використання різних видів циклів.

Завдання по варіантам

Варіант 1.

1. Обчислити суму парних натуральних чисел від *n* до 105.
2. Обчислити: $\frac{(\sqrt{3}+1)^2}{4}+\frac{(\sqrt{3}+2)^2}{5}+\frac{(\sqrt{3}+3)^2}{6}+...+\frac{(\sqrt{3}+n)^2}{n+3}$.
3. q (1000) шт. цегли можна перевозити візками місткістю 100, 300, 400 і 500 шт. цегли. Отримати всі можливі варіанти перевезень.

Варіант 2.

1. Знайти тризначні числа від *n* до 800, що діляться одночасно на 16 і 24.
2. Дано дійсне *a*. Обчислити: $y=\frac{3}{a}+\frac{4}{a^2}+\frac{5}{a^3}+\frac{6}{a^4}+...+\frac{n+2}{a^n}$
3. Футбольний м'яч коштує s(65) грн. Отримати всі можливі варіанти оплати, якщо у покупця є 5-, 10- і 20-гривневі купюри.

Варіант 3.

1. Обчислити суму *n* доданків виду $\frac{i}{i+1}$, де *i* набуває значень натуральних чисел від 1 до *n*.
2. Обчислити: $y=\sin 1(\sin 1+\sin 2)(\sin 1+\sin 2+\sin 3)...(\sin 1+\sin 2+\sin 3+...+\sin n)$
3. Садівникові потрібно z(18) кг мінеральних добрив. Отримати всі можливі варіанти купівлі добрива, якщо в магазині продаються розфасовки по 5, 4 і 2 кг.

Варіант 4.

1. Обчислити суму квадратів непарних натуральних чисел від *i* до *n*, де *i* задане натуральне число, *i*<*n*-1.
2. Дано натуральне n. Обчислити: $\frac{1}{2}+\frac{2}{3}+\frac{3}{4}+...+\frac{n}{n+1}$.
3. Повітроплавцеві потрібно заповнити воднем повітряну кулю місткістю x(17) куб. м балончиками по 1, 2 і 5 куб. м водню. Отримати всі можливі варіанти наповнення.

Варіант 5.

1. Знайти тризначні числа від 100 до *n*, що дорівнюють сумі кубів своїх цифр.
2. Дано дійсне x. Обчислити: y=sin*x*+sin*2x*+sin*3x*+…+sin*nx*
3. Шляховим майстрам потрібно прокласти r(190) м залізниці рейками по 8 і 10 м. Отримати всі можливі варіанти прокладання.

Варіант 6.

1. Обчислити суму натуральних чисел, кратних 3, що належать інтервалу від 1 до *n*.
2. Дано натуральне n. Обчислити:
$$y=\frac{\cos 1}{\sin 1}\cdot\frac{\cos 1+\cos 2}{\sin 1+\sin 2}\cdot...\cdot\frac{\cos 1+\cos 2+...+\cos n}{\sin 1+\sin 2+...+\sin n}$$
3. d(36) кг яблук потрібно розфасувати у пакети по 2, 4, 5 і 10 кг. Отримати всі можливі варіанти розфасування.

Варіант 7.

1. Обчислити суму натуральних чисел, кратних 5, що належать інтервалу від *i* до *n*, де *i* - деяке натуральне число, менше від *n*.
2. Дано дійсне x і натуральне n.. Обчислити: y=sin(x)+sin(x^2)+sin(x^3)+….+sin(x^n).
3. f(240) екскурсантів можна розсадити в автобуси ЛАЗ (місткість – 40 осіб) і ПАЗ (місткість – 30 осіб). Отримати всі можливі варіанти замовлень автобусів для перевезення пасажирів.

Варіант 8.

1. Ввести з клавіатури *n* довільних цілих чисел і обчислити кількість додатних непарних з них.
2. Дано натуральне n. Обчислити:

$$y = \frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + ... + \frac{1}{\sin 1 + \sin 2 + ... + \sin n}.$$

3. Друкарці можна друкувати книги обсягом 30, 40 і 60 аркушів за її вибором. Усього вона надрукувала n(1200) аркушів. Визначити скільки і яких книг вона могла надрукувати. Отримати всі можливі варіанти.

Варіант 9.

1. Ввести з клавіатури *n* довільних цілих чисел і обчислити суму додатних парних з них.
2. Дано дійсне x і натуральне n. Обчислити:

$$y = \frac{x-2}{x+3} + \frac{x-3}{x+4} + \frac{x-4}{x+5} + ... + \frac{x-n}{x+n+1}$$

3. Для ремонту дороги потрібно завезти p(24) т щебеню. В автопарку є самоскиди вантажопідйомністю 3, 4 і 6 т. Отримати всі можливі варіанти перевезення щебеню.

Варіант 10.

1. Ввести з клавіатури *n* довільних цілих чисел і обчислити середнє арифметичне від'ємних чисел.
2. Дано дійсне x і натуральне n. Обчислити:

$$y = \frac{2}{x+3} + \frac{3}{x+4} + \frac{4}{x+5} + ... + \frac{n}{x+n+1}.$$

3. t(200) л бензину потрібно розлити у баки місткістю 60, 45 і 25 літрів. Отримати всі можливі варіанти розливу.

Варіант 11.

1. Задано послідовність натуральних чисел від 1 до *n*. Знайти суму чисел, некратних трьом.
2. Дано дійсне a і натуральне n. Обчислити:

$$y=\left(1+\frac{1}{a+2}\right)+\left(1+\frac{1}{a+4}\right)+\left(1+\frac{1}{a+6}\right)+...+\left(1+\frac{1}{a+2n}\right)$$

3. p(120) осіб потрібно посадити за 4- та 6-містні столики. Отримати всі можливі варіанти поєднання столиків.

Варіант 12.

1. Серед довільного ряду цілих чисел, які вводяться з клавіатури, визначити кількість натуральних, кратних 9.
2. Дано натуральне число n. Обчислити: $y=\frac{3+6}{4+2}+\frac{3+7}{5+2}+\frac{3+8}{6+2}+...+\frac{3+n}{n-2+2}$.
3. Вантаж s(200) т можна перевозити вантажівками по 3, 4 і 5 т. Отримати всі можливі плани перевезень.

Варіант 13.

1. Обчислити суму довільних чисел, що вводяться з клавіатури і кількість яких наперед невідома. Розрахунок припиняється, якщо введене число дорівнює одиниці або нулю.
2. Дано дійсне x і натуральне n. Обчислити: $1+\frac{x^1}{1}+\frac{x^2}{2}+\frac{x^3}{3}+...+\frac{x^n}{n}$.
3. z(120) нафтовиків потрібно перевезти на вахту вертольотами МІ-2 і МІ-8. Місткість МІ-2 становить – 8 осіб, а МІ-8 – 28 осіб. Отримати всі можливі варіанти поєднання МІ-2 і МІ-8 для перевезення нафтовиків.

Варіант 14.

1. Обчислити добуток натуральних парних чисел від 1 до введеного тризначного числа, кратних 3, але некратних 9.
2. Дано дійсне x. Обчислити: (x-1)(x-2)(x-3)…(x-n).
3. x(50) тюльпанів потрібно розфасувати у подарункові набори по 3, 5 і 7 квіток. Вивести на екран всі можливі варіанти букетів.

Варіант 15.

1. Обчислити добуток натуральних чисел, кратних 5, від *i* до *n*.
2. Дано натуральне n. Обчислити: $y = \cos 1 \cos 2 \cos 3 \cos 4 \ldots \cos n$
3. У магазині для пересилання поштою підготовлено n(60) книг. Посилки комплектують по 10, 15 і 20 книг. Отримати всі можливі варіанти комплектів.

Варіант 16.

1. Знайти добуток усіх натуральних парних чисел від 20 до *n*.
2. Дано дійсне a. Обчислити: y=a(a+1)(a+2)…(a+n-1)
3. y(30) л соку потрібно розлити в 2- і 3-літрові банки. Вивести на екран всі можливі варіанти розливу.

Варіант 17.

1. Знайти двозначні числа від *n* до 99, для яких число дорівнює сумі подвоєного квадрата першої цифри і квадрата другої цифри.
2. Дано дійсне a. Обчислити:

$$y = \frac{1}{a} + \frac{1}{a(a+1)} + \frac{1}{a(a+1)(a+2)} + \ldots + \frac{1}{a(a+1)(a+2)\ldots(a+n)}.$$

3. t(350) саджанців помідорів для продажу потрібно розфасувати в пакети по 30, 50 і 100 шт. Отримати всі можливі варіанти розфасування.

Варіант 18.

1. Знайти суму квадратів двозначних непарних чисел від 10 до *n*, які діляться на 3, та вказати їх кількість.
2. Дано дійсне x і натуральне n. Обчислити:

$$y = \frac{\sin(x)}{2} + \frac{\sin(x)}{3} + \frac{\sin(x)}{4} + \ldots + \frac{\sin(x)}{n}.$$

3. На перевезення a(800) кг овочів з бази підготовлено ящики. В них можна завантажити по 8, 10 і 15 кілограмів. Отримати всі можливі варіанти завантаження.

Варіант 19.

1. Знайти суму квадратів двозначних чисел від *n* до *m*, що діляться одночасно на 4 і 6.
2. Дано натуральне n. Обчислити: $n+(n-1)+(n-2)+...+1$
3. m(400) т зерна можна перевозити вантажівками по 2, 4 і 6 т. Отримати всі можливі варіанти перевезень.

Варіант 20.

1. Знайти добуток двозначних чисел від 20 до *n*, що діляться одночасно на 5 і 3 та вивести їх на екран.
2. Дано дійсне a і натуральне n. Обчислити: y=a(a-n)(a-2n)(a-3n)…(a-n^2).
3. Потрібно викласти k(100) м бордюру брусками по 1, 2 і 3 м. Отримати всі можливі варіанти прокладання.

Лабораторна робота №6
Одновимірні масиви.

Мета і зміст роботи:

Отримати навички опрацювання одновимірних масивів.

В результаті виконання роботи необхідно:

- вміти створювати одновимірний масив різними способами і виводити його на екран в різних видах;
- вчитися обирати найбільш оптимальний варіант заповнення масиву;
- вчитися використовувати функцію **rand()**;
- вчитися виконувати опрацювання масивів;
- користуватися вказівниками при звертанні до елементів масиву.

Методичні вказівки

1. При виконанні роботи використовувати статичні масиви. Для організації статичних масивів зі змінними розмірами необхідно оголосити масив достатньо великої довжини (наприкладр, 100). Реальну довжинумасива вводить користувач. Останні елементи масива, хоча пам’ять під них і виділена, не будуть розглядатись. При зміні довжини масива необхідно змінювати його реальну довжину.
2. Формувати одновимірний масив, використовуючи датчик випадкових чисел, а отриманий масив виводити на екран. Якщо є обґрунтована необхідність, то масив можно вводити вручну.
3. Всі программі мають містити детальні пояснювальні коментарі.
4. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті.

Зміст звіта

1. Номер роботи, тему и постановку задачі.
2. Програму розв’язування кожної задачі.
3. Результати роботи кожної програми.
4. Пояснення результатів.
5. Прийом завдань відбувається в наступному порядку:
   - написаний або надрукований звіт по роботі (див. вище);
   - працюючі, повністю налагоджені програми, які містять детальні коментарі і збережені на *flash*-карту;
   - усні відповіді на 5-7 питань по лекції.

Завдання по варіантам

Варіант 1.

1) Дано лінійний масив чисел. Знайти всі від’ємні елементи даного масиву та обчислити їх добуток.
2) Дано лінійний масив цілих чисел. Обчислити добуток мінімального і максимального елементів масиву.

Варіант 2.

1) Дано лінійний масив дійсних чисел. Знайти елементи масиву, у яких значення співпадає з порядковим номером та підрахувати їх кількість.
2) Дано лінійний масив дійсних чисел. Знайти добуток третього і максимального елементів.

Варіант 3.

1) Дано лінійний масив цілих чисел. Знайти всі парні елементи масиву та обчислити їх суму.
2) Дано лінійний масив дійсних чисел. Обчислити суму максимального елементу масиву і кількості нульових елементів масиву.

Варіант 4.

1) Дано лінійний масив дійсних чисел. Знайти елементи масиву, які не більше 3 та підрахувати їх кількість.
2) Дано лінійний масив дійсних чисел. Обчислити середнє арифметичне кількості додатних елементів масиву і максимального елементу.

Варіант 5.

1) Дано лінійний масив дійсних чисел. Вивести на екран елементи масиву, що стоять на непарних місцях та обчислити їх суму.
2) Дано лінійний масив дійсних чисел. Обчислити добуток максимального і п’ятого елементу.

Варіант 6.

1) Дано лінійний масив дійсних чисел. Знайти всі додатні елементи даного масиву та обчислити їх суму.
2) Дано лінійний масив дійсних чисел. Підрахувати кількість елементів масиву, які більші за мінімальний елемент на 5.

Варіант 7.

1) Дано лінійний масив дійсних чисел. Знайти всі елементи даного масиву, які є точними квадратами та обчислити їх добуток.
2) Дано лінійний масив дійсних чисел. Обчислити суму останнього і максимального елементів масиву.

Варіант 8.

1) Дано лінійний масив цілих чисел. Знайти всі елементи даного масиву, які не кратні 10 та обчислити їх суму.
2) Дано лінійний масив дійсних чисел. Обчислити добуток першого і мінімального елементів масиву.

Варіант 9.

1) Дано лінійний масив дійсних чисел. Знайти всі додатні елементи даного масиву та підрахувати їх кількість.
2) Дано лінійний масив дійсних чисел. Обчислити середнє арифметичне мінімального і максимального елементів даного масиву.

Варіант 10.

1) Дано лінійний масив дійсних чисел. Вивести на екран елементи, які не є точними квадратами та обчислити їх кількість.
2) Дано лінійний масив дійсних чисел. Підрахувати, скільки в масиві зустрічається максимальне число.

Варіант 11.

1) Дано лінійний масив дійсних чисел. Знайти всі додатні елементи даного масиву та підрахувати їх суму і кількість нульових елементів масиву.
2) Дано лінійний масив дійсних чисел. Обчислити середнє арифметичне всіх елементів масиву і визначити різницю між максимальним елементом і знайденим значенням.

Варіант 12.

1) Дано лінійний масив дійсних чисел. Знайти всі непарні елементи масиву та підрахувати скільки разів зустрічається в масиві модуль числа 7.
2) Дано лінійний масив дійсних чисел. Обчислити суму останнього і мінімального елементів масиву.

Варіант 13.

1) Дано лінійний масив цілих чисел. Знайти всі від’ємні елементи даного масиву та обчислити їх середнє арифметичне.
2) Дано лінійний масив дійсних чисел. Обчислити суму елементів, що стоять на непарних місцях і порівняти її з мінімальним елементом.

Варіант 14.

1) Дано лінійний масив дійсних чисел. Знайти всі від'ємні елементи даного масиву та обчислити суму їх квадратів.
2) Дано лінійний масив дійсних чисел. Обчислити суму елементів, що стоять на парних місцях і порівняти її з максимальним елементом.

Варіант 15.

1) Дано лінійний масив дійсних чисел. Вивести на екран елементи масиву, що стоять на парних місцях та обчислити їх добуток.
2) Дано лінійний масив дійсних чисел. Обчислити різницю між найбільшим і найменшим елементами масиву.

Варіант 16.

1) Дано лінійний масив цілих чисел. Знайти всі непарні елементи масиву та обчислити їх добуток.
2) Дано лінійний масив дійсних чисел. Підрахувати, скільки разів в масиві зустрічається мінімальне число.

Варіант 17.

1) Дано лінійний масив цілих чисел. Знайти всі елементи одновимірного масиву, що кратні 5 та обчислити їх суму.
2) Дано лінійний масив дійсних чисел. Обчислити добуток між максимальним елементом масиву та сумою його додатних елементів.

Лабораторна робота №7
Багатовимірні масиви

Мета і зміст роботи:

Отримати навички роботи з багатовимірними масивами.

В результаті виконання роботи необхідно:

- вміти заповнювати багатовимірні масиви різними способами і виводити його на екран у вигляді таблиці;
- вчитися розглядати масив порядково і по стовпчиках;
- користуватися вказівниками при звертанні до елементів масиву.

Методичні вказівки

1. При виконанні роботи використовувати статичні масиви. Для організації статичних масивів зі змінними розмірами необхідно оголоситимасив достатньо великої довжини (наприкладр, 100). Реальну довжинумасива вводить користувач. Останні елементи масива, хоча пам’ять під них і виділена, не будуть розглядатись. При зміні довжини масива необхідно змінювати його реальну довжину.
2. Формувати одновимірний масив, використовуючи датчик випадкових чисел, а отриманий масив виводити на екран. Якщо є обґрунтована необхідність, то масив можно вводити вручну.
3. Всі программі мають містити детальні пояснювальні коментарі.
4. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті.

Завдання по варіантам

Варіант 1.

1) Дано двовимірний масив дійсних чисел. Обчислити суму всіх додатних елементів масиву і помножити її на перший елемент масиву.

2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких є додатні елементи.

Варіант 2.

1) Дано двовимірний масив цілих чисел. Обчислити добуток всіх елементів, значення яких парні.
2) Дано двовимірний масив цілих чисел. Знайти суму добутків всіх рядків.

Варіант 3.

1) Дано двовимірний масив дійсних чисел. Обчислити добуток першого елементу масиву і кількості від'ємних.
2) Дано двовимірний масив цілих чисел. Знайти номери рядків, в яких немає парних елементів.

Варіант 4.

1) Дано двовимірний масив дійсних чисел. Обчислити добуток всіх від'ємних елементів масиву і додати його до третього елементу масиву в другому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких є тільки один додатний елемент.

Варіант 5.

1) Дано двовимірний масив дійсних чисел. Обчислити середнє арифметичне всіх елементів масиву і визначити різницю між знайденим значенням і другим елементом в третьому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких є тільки один непарний елемент.

Варіант 6.

1) Дано двовимірний масив дійсних чисел. Обчислити середнє арифметичне додатних елементів масиву.
2) Дано двовимірний масив цілих чисел. Знайти суму добутків всіх стовпчиків.

Варіант 7.

1) Дано двовимірний масив цілих чисел. Обчислити кількість всіх елементів, значення яких непарні і помножити на кількість елементів третього рядка.
2) Дано двовимірний масив цілих чисел. Знайти кількість рядків, в яких є два парних елемента.

Варіант 8.

1) Дано двовимірний масив дійсних чисел. Обчислити суму кількості додатних елементів першого стовпчика і суми всіх елементів масиву.
2) Дано двовимірний масив цілих чисел. Знайти кількість стовпчиків, в яких немає нульових елементів.

Варіант 9.

1) Дано двовимірний масив дійсних чисел. Обчислити середнє арифметичне всіх від'ємних елементів масиву.
2) Дано двовимірний масив цілих чисел. Знайти номери рядків, в яких немає від'ємних елементів.

Варіант 10.

1) Дано двовимірний масив дійсних чисел. Підрахувати кількість від'ємних елементів масиву і додати до неї елемент, що стоїть на перетині другого рядка і першого стовпця.
2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких є нульові елементи.

Варіант 11.

1) Дано двовимірний масив дійсних чисел. Обчислити суму передостаннього елементу масиву в передостанньому рядку і кількості додатних елементів масиву.
2) Дано двовимірний масив цілих чисел. Знайти номери рядків, в яких є тільки один нульовий елемент.

Варіант 12.

1) Дано двовимірний масив дійсних чисел. Обчислити суму всіх елементів, значення яких є точними квадратами.
2) Дано двовимірний масив цілих чисел. Знайти кількість стовпчиків, в яких є тільки один від'ємний елемент.

Варіант 13.

1) Дано двовимірний масив дійсних чисел. Підрахувати кількість парних елементів масиву і помножити її на суму перших трьох елементів головної діагоналі.
2) Дано двовимірний масив цілих чисел. Знайти кількість рядків, в яких є лише по два додатні елементи.

Варіант 14.

1) Дано двовимірний масив дійсних чисел. Підрахувати кількість нульових елементів масиву і додати до неї елемент масиву, що стоїть в другому рядку і третьому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти добуток сум усіх рядків.

Варіант 15.

1) Дано двовимірний масив дійсних чисел. Обчислити кількість всіх непарних елементів, і помножити її на перший елемент третього рядка.
2) Дано двовимірний масив цілих чисел. Знайти номери рядків, в яких є від'ємні елементи.

Варіант 16.

1) Дано двовимірний масив дійсних чисел. Обчислити середнє арифметичне всіх парних елементів масиву і визначити добуток між знайденим значенням і третім елементом в другому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти добуток сум усіх стовпчиків.

Варіант 17.

1) Дано двовимірний масив цілих чисел. Обчислити середнє арифметичне всіх непарних елементів масиву і визначити суму між знайденим значенням і третім елементом в першому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти кількість рядків, в яких є нульові елементи.

Варіант 18.

1) Дано двовимірний масив дійсних чисел. Обчислити суму всіх додатних елементів масиву, значення яких не більші 5 і знайти суму між знайденим значенням і другим елементом в першому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти добуток номерів рядків, в яких немає нульових елементів.

Варіант 19.

1) Дано двовимірний масив дійсних чисел. Обчислити добуток елементів, які стоять на головній діагоналі і визначити суму між знайденим значенням і другим елементом в першому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких немає додатних елементів.

Варіант 20.

1) Дано двовимірний масив дійсних чисел. Знайти мінімальне значення серед всіх елементів масиву, обчислити середнє арифметичне додатних елементів масиву, але не більших 7 і визначити суму цих значень.
2) Дано двовимірний масив цілих чисел. Знайти кількість стовпчиків, в яких сума більше 50.

Варіант 21.

1) Дано двовимірний масив дійсних чисел. Знайти кількість точних квадратів і додати до неї значення третього елементом в другому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти номери рядків, в яких добуток елементів менше 40.

Варіант 22.

1) Дано двовимірний масив дійсних чисел. Обчислити суму всіх елементів, значення яких від'ємні, але не менші -3 і знайти різницю між знайденим значенням і п'ятим елементом в шостому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти кількість рядків, в яких кількість парних елементів більше 3.

Варіант 23.

1) Дано двовимірний масив дійсних чисел. Знайти суму не точних квадратів і відняти від неї значення другого елемента в першому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти кількість стовпчиків, в яких є хоча б один непарний елемент.

Варіант 24.

1) Дано двовимірний масив дійсних чисел. Знайти максимальне значення серед всіх елементів масиву, обчислити середнє арифметичне від'ємних елементів масиву, але не менших -5 і визначити добуток цих значень.
2) Дано двовимірний масив цілих чисел. Знайти номери стовпчиків, в яких немає точних квадратів.

Варіант 25.

1) Дано двовимірний масив дійсних чисел. Обчислити середнє арифметичне елементів, які стоять на головній діагоналі і визначити добуток між знайденим значенням і першим елементом в четвертому стовпчику.
2) Дано двовимірний масив цілих чисел. Знайти суму номерів рядків, в яких є додатні елементи.

Лабораторна робота №8
Динамічні масиви.

Мета і зміст роботи:

Получить навыки работы с двумерными динамическими массивами в С++.

В результате выполнения работы необходимо:

- учиться заполнять и обрабатывать двумерные динамические массивы;
- учиться выполнять обращение к функциям через указатели;
- обработка строк двумерного динамического массива с использованием указателей и функций.

Методичні вказівки

1. Двумерный массив задавать как динамический.
2. Размерность массива задает пользователь.
3. Массив заполняется случайными числами.
4. Обработку строк выполнять через функцию.
5. Программа должна содержать выводы промежуточных результатов.
6. Все программы должны содержать подробные пояснительные комментарии.
7. Данные, результаты и пояснительные тексты на экране должны располагаться эстетически. Все математические расчеты и преобразования должны быть изложены в отчете.

Завдання по варіантам

Варіант 1.

Дано двовимірний масив цілих чисел. Знайти суму добутків тих рядків, в яких є два парних елементів.

Варіант 2.

Дано двовимірний масив цілих чисел. Знайти номер рядка, в якому сума від’ємних непарних елементів найбільша.

Варіант 3.

Дан двумерный массив действительных чисел. Определить сумму элементов строки, в которой расположен элемент с наименьшим значением.

Варіант 4.

Дан двумерный массив действительных чисел. Составьте программу подсчёта количества строк, произведение элементов которых больше 520.

Варіант 5.

Дан двумерный массив действительных чисел. Составьте программу, определяющую минимальное значение среди максимальных элементов строк.

Варіант 6.

Дан двумерный массив действительных чисел. Составьте программу поиска номера строки, в которой сумма элементов, стоящих на четных местах, минимальна.

Варіант 7.

Дан двумерный массив действительных чисел. Составьте программу поиска номера строки, в которой произведение квадратов элементов максимально.

Варіант 8.

Дан двумерный массив действительных чисел. Определить, есть ли в массиве хотя бы одна строка, для которой сумма минимального и максимального значения отрицательна.

Варіант 9.

Дан двумерный массив действительных чисел. Составьте программу поиска номера строки, в которой сумма элементов максимальна.

Варіант 10.

Дан двумерный массив действительных чисел. Составьте программу, определяющую минимальные элементы строк, в которых максимальный элемент отрицателен.

Варіант 11.

Дан двумерный массив действительных чисел. Составьте программу, которая определяет, есть ли в таблице строка, произведение элементов которой положительно.

Варіант 12.

Дано двовимірний масив цілих чисел. Знайти добуток сум елементів тих рядків, в яких є додатні елементи.

Варіант 13.

Дано двовимірний масив цілих чисел. Знайти суму добутків рядків.

Варіант 14.

Дано двовимірний масив цілих чисел. Знайти добуток непарних елементів тих рядків, в яких немає парних елементів.

Варіант 15.

Дано двовимірний масив цілих чисел. Знайти суму квадратів сум елементів тих рядків, в яких є тільки один додатний елемент.

Варіант 16.

Дано двовимірний масив цілих чисел. Знайти номер рядка, сума додатних елементів якого найбільша.

Варіант 17.

Дано двовимірний масив цілих чисел. Знайти кількість рядків, в яких немає нульових елементів.

Варіант 18.

Дано двовимірний масив цілих чисел. Знайти добуток сум тих рядків, в яких є тільки один нульовий елемент.

Варіант 19.

Дано двовимірний масив цілих чисел. Знайти тільки ті рядки, в яких є лише по два додатних елемента і загальну суму саме цих елементів.

Варіант 20.

Дано двовимірний масив цілих чисел. Знайти суму парних елементів тих рядків, в яких є додатні елементи.

Варіант 21.

Дано двовимірний масив цілих чисел. Знайти номер рядка, в якому кількість від'ємних елементів найбільша.

Варіант 22.

Дано двовимірний масив цілих чисел. Знайти добуток додатних мінімальних елементів рядків.

Варіант 23.

Дано двовимірний масив цілих чисел. Знайти найменше середнє арифметичне значення мінімального та максимального елементів рядка.

Варіант 24.

Дано двовимірний масив цілих чисел. Знайти добуток сум квадратів парних елементів рядків.

Варіант 25.

Дан двумерный массив действительных чисел. Составьте программу, определяющую сумму максимальных элементов строк.

Лабораторна робота №9
Разробка і виконання програм, які містять оператори циклу. Табуляція функції.

Мета і зміст роботи:

Создание, отладка и выполнение программ, содержащих различные виды циклов, работа с вложенными циклами, изучение разных форм записи операторов цикла.

В результате выполнения работы необходимо:

- уметь применять операторы циклов;
- учиться выбирать наиболее оптимальный вариант из трех операторов цикла;
- учиться использовать вложенные циклы;
- выполнять табулирование функций как одной, так и нескольких переменных;
- уметь выполнять трассировку программы и объяснять принцип работы программы.

Методичні вказівки

1. В первой задаче предпочтительнее рассматривать несколько вложенных циклов, чем внутри одного цикла разбивать число на цифры, что, однако, ошибкой не является.
2. Во время табуляции функции на экран выводить таблицу в виде «таблицы Пифагора» со значениями аргументов.
3. В третьей задаче полностью повторить вторую, добавив к ней строки, необходимые для решения поставленной задачи.
4. Все программы должны содержать подробные пояснительные комментарии.
5. Данные, результаты и пояснительные тексты на экране должны располагаться эстетически. Все математические расчеты и преобразования должны быть изложены в отчете.

Завдання по варіантам

Варіант 1.

1. Знайти всі двозначні числа, для яких модуль різниці його цифр дорівнює 5.
2. Протабулювати функцію $f(x,y) = -2y^2 + 2\cos(15x)$, якщо $x \in (-1;4)$ з кроком 0,65, а $y \in (-4;1)$ з кроком 0,45.
3. Знайти кількість нульових значень та максимальне значення протабульованої функції.

Варіант 2.

1. Знайти всі двозначні числа, в яких сума подвоєного квадрата першої цифри і квадрата другої цифри дорівнює самому числу.
2. Протабулювати функцію $f(x,y) = \sin(x-y)$, якщо $x \in (0;0{,}2)$, а $y \in (1;1{,}2)$ з кроком обох змінних 0,02.
3. Обчислити добуток значень аргументу, для яких досягається максимальне та мінімальне значення протабульованої функції.

Варіант 3.

1. Знайти всі тризначні числа, що діляться на суму своїх цифр.
2. Протабулювати функцію $f(x,y) = \sin x + \cos 2y$, якщо $x \in (1;2{,}5)$, а $y \in (0;1{,}8)$ з кроком обох змінних 0,25.
3. Обчислити добуток усіх додатних та кількість від'ємних значень протабульованої функції.

Варіант 4.

1. Знайти всі тризначні числа, що діляться одночасно на 16 і 24.
2. Протабулювати функцію $f(x,y) = e^{(-(x-y)^2}$, якщо $x \in (1;1{,}3)$, а $y \in (0;0{,}3)$ з кроком обох змінних 0,05.
3. Обчислити добуток усіх значень протабульованої функції, які належать проміжку $(-1;1)$, а також максимальне та мінімальне значення функції на цьому проміжку.

Варіант 5.

1. Знайти всі тризначні числа, що дорівнюють сумі потроєних квадратів своїх цифр.
2. Протабулювати функцію $f(x,y)=\ln(y+\sqrt{|x-y|})$, якщо $x\in(2;3)$, а $y\in(0;1{,}3)$ з кроком обох змінних 0,1.
3. Обчислити окремо кількість від'ємних та додатних значень протабульованої функції.

Варіант 6.

1. Знайти всі чотиризначні «щасливі» числа, в яких сума першої та третьої цифри дорівнює сумі другої та четвертої.
2. Протабулювати функцію $f(x,y)=1/(x+\sqrt{|y|})$, якщо $x\in(1;2)$, а $y\in(0;3)$ з кроком обох змінних 0,4.
3. Обчислити кількість значень протабульованої функції, які належать проміжкам $(-\infty;-3)$ і $(0;+\infty)$.

Варіант 7.

1. Знайти всі трьохзначні числа, що діляться одночасно на 12 або 15.
2. Протабулювати функцію $f(x,y)=\ln(1+\sqrt{x+y})$, якщо $x\in(0;2)$, а $y\in(1;2)$ з кроком обох змінних 0,2.
3. Обчислити кількість мінімальних та максимальних значень протабульованої функції.

Варіант 8.

1. Знайти всі двозначні натуральні числа, в яких сума цифр ділиться на 5 або на 10.
2. Протабулювати функцію $f(x,y)=tg(1/(x^2+y^2))$, якщо $x\in(0;0{,}35)$, а $y\in(0;0{,}5)$ з кроком обох змінних 0,05.
3. Обчислити кількість та добуток тих значень протабульованої функції, для яких виконується нерівність $1.3<f<5$.

Варіант 9.

1. Знайти всі тризначні натуральні числа, в яких сума цифр ділиться на 3 і на 5.
2. Протабулювати функцію $f(x,y)=e^{-x+\sqrt{|y|}}$, якщо $x\in(1;2)$, а $y\in(3;3{,}5)$ з кроком обох змінних 0,1.
3. Обчислити кількість та добуток усіх від'ємних значень протабульованої функції.

Варіант 10.

1. Знайти всі двозначні числа, що діляться на добуток своїх цифр.
2. Протабулювати функцію $f(x,y)=3x^2+2\sin 3y$, якщо $x\in(-2;3)$, а $y\in(0;3)$ з кроком обох змінних 0,2.
3. Обчислити кількість та суму тих значень протабульованої функції, для яких виконується нерівність $0<f<1$.

Варіант 11.

1. Знайти всі чотиризначні непарні числа, що є точними квадратами.
2. Протабулювати функцію $f(x,y)=\cos x+\sin 2y$, якщо $x\in(0;0{,}4)$, а $y\in(0;0{,}6)$ з кроком обох змінних 0,05.
3. Обчислити максимальне значення протабульованої функції, а також визначити значення аргументу, для якого воно досягається.

Варіант 12.

1. Знайти кількість всіх тризначних чисел, що діляться одночасно на 15 і 18.
2. Протабулювати функцію $f(x,y)=\sqrt{x^2+y^2+1}$, якщо $x\in(1;5)$, а $y\in(0;4)$ з кроком обох змінних 0,5.
3. Обчислити мінімальне значення протабульованої функції, а також визначити значення аргументу, для якого воно досягається.

Варіант 13.

1. Знайти всі двозначні непарні числа, які діляться одночасно на 3 і на 7.
2. Протабулювати функцію $f(x,y)=e^{-x^2-y^2}$, якщо $x\in(0;2)$, а $y\in(1;2)$ з кроком обох змінних 0,2.
3. Обчислити модуль суми максимального та мінімального значень протабульованої функції.

Варіант 14.

1. Знайти всі тризначні числа, що дорівнюють добутку своїх цифр.
2. Протабулювати функцію $f(x,y)=\cos(x+y)$, якщо $x\in(1;1{,}2)$, а $y\in(2;2{,}25)$ з кроком обох змінних 0,02.
3. Обчислити суму кубів додатних значень протабульованої функції та їхню кількість.

Варіант 15.

1. Знайти всі тризначні парні числа, що є точними квадратами.
2. Протабулювати функцію $f(x,y)=e^{1+x-y}$, якщо $x\in(1;1{,}4)$, а $y\in(0;0{,}5)$ з кроком обох змінних 0,05.
3. Обчислити суму усіх від'ємних та кількість додатних значень протабульованої функції.

Варіант 16.

1. Знайти всі двозначні числа, що є точними квадратами.
2. Протабулювати функцію $f(x,y)=\sin x\cdot\cos 2y$, якщо $x\in(0;0{,}4)$, а $y\in(0;0{,}6)$ з кроком обох змінних 0,04.
3. Обчислити суму усіх значень протабульованої функції, для яких виконуються нерівності $f<-0{,}2$ або $f>0{,}5$ та їх кількість.

Варіант 17.

1. Знайти всі двозначні парні числа, які при діленні на суму своїх цифр дають в остачі 1.
2. Протабулювати функцію $f(x,y)=\ln(x+\sqrt{x^2+y^2})$, якщо $x\in(1;4)$, а $y\in(1;3{,}5)$ з кроком обох змінних 0,4.
3. Яких значень протабульованої функції більше: додатних чи від'ємних.

Варіант 18.

1. Знайти всі двозначні прості числа.
2. Протабулювати функцію $f(x,y)=x^3+\sqrt{y}$, якщо $x\in(-2;2)$, а $y\in(0;3{,}5)$ з кроком обох змінних 0,35.
3. Обчислити добуток та кількість усіх значень протабульованої функції, для яких виконується нерівність $-4<f<3$ або та їх кількість.

Варіант 19.

1. Знайти всі тризначні числа, добуток цифр яких ділиться на 8.
2. Протабулювати функцію $f(x,y)=\ln\left|y^2-x^3\right|$, якщо $x\in(-1;3)$, а $y\in(-3;1{,}3)$ з кроком обох змінних 0,3.
3. Обчислити добуток тих значень протабульованої функції, для яких виконуються нерівності $f<-4$ або $f>0{,}5$.

Варіант 20.

1. Знайти всі двозначні числа, в яких перша цифра ділиться націло на другу.
2. Протабулювати функцію $f(x,y)=\sqrt{\left|\ln x^2+e^y\right|}$, якщо $x\in(-1;3{,}5)$, а $y\in(1;4{,}5)$ з кроком обох змінних 0,25.
3. Обчислити середнє арифметичне всіх значень протабульованої функції.

Варіант 21.

1. Знайти всі тризначні числа, в яких сума перших двох цифр ділиться на третю.
2. Протабулювати функцію $f(x, y) = 3^{x+y} - y^2$, якщо $x \in (1; 4)$, а $y \in (-2; 0)$ з кроком обох змінних 0,2.
3. Обчислити максимальне значення серед від'ємних значень протабульованої функції, а також визначити значення аргументу, для якого воно досягається.

Лабораторна робота №10
Робота з рядками.

Мета і зміст роботи :

Отримати навики в роботі з рядками в С++.

В результаті виконання роботи необхідно:

- Навчитися використовувати стандартні функції для роботи з рядками;
- Навчитися виконувати обробку рядків;
- Вміти користуватися вказівниками при зверненні до рядків.

Методичні вказівки

1. Задано рядок, що складається із символів. Символи об’єднуються у слова. Слова один від одного відокремлюються одним або декількома пропусками. В кінці тексту ставиться крапка.Текст містить у собі не більше, ніж 255 символів. Виконати введення і обробку рядка відповідно до свого варіанту.
2. Всі програми повинні містити у собі коментарі з докладними поясненнями.
3. Данні, результати і пояснювальні тексти повинні розташовуватися на екрані естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті.

Завдання по варіантам.

Варіант 1.

Складіть програму, яка у заданому користувачем реченні X знаходить найдовше слово, враховуючи, що слова відділені один від одного одним або декількома пропусками.

Варіант 2.

Складіть програму, яка для заданого користувачем речення з'ясовує, які символи хоча б один раз зустрічаються у ньому.

Варіант 3.

Складіть програму, яка для заданого користувачем речення підраховує, скільки в ньому голосних букв і розділових знаків.

Варіант 4.

Дані слова X і Y. Складіть програму, яка з'ясовує, чи є слово Y перестановкою букв слова X.

Варіант 5.

Задано речення , в якому слова розділені одним або декількома пропусками. Складіть програму, яка буде підраховувати кількість слів у реченні.

Варіант 6.

Задано речення X. Складіть програму, яка з'ясовує, чи можна з його букв скласти слово Y. Кожну букву із X можна вживати тільки один раз.

Варіант 7.

Дано два рядки. Складіть програму, яка з першого заданого рядка видаляє кожен символ, що належить другому заданому рядку.

Варіант 8.

Дано речення. Складіть програму, що знаходить всі слова, в яких зустрічається максимальна кількість букв «а».

Варіант 9.

Дано речення. Складіть програму, що знаходить всі слова, в яких зустрічається буква «а» пристня не менше двох разів.

Варіант 10.

Дано рядок символів. Визначити найдовше слово в цьому рядку.

Варіант 11.

Дано два рядка символів. Визначити в якій із них кількість слів найбільша.

Варіант 12.

Дано рядок символів. Складіть програму, яка визначає довжину максимальної серії символів, що складається з цифр.

Варіант 13.

Задано речення, що складається зі слів, розділених будь-якою кількістю пробілів. Складіть програму, яка редагує речення залишаючи між словами тільки по одному пробілу.

Варіант 14.

Задано рядок, що складається зі слів. Складіть програму, яка в словах що, закінчуються на «ing», змінює закінчення на «ed».

Варіант 15.

Дано речення. Напишіть програму, яка перевіряє, чи виконується в даному рядку баланс дужок.

Варіант 16.

Виключити із заданого рядка символів всі групи символів, розташовані між «(» і «)». Дужки також виключити. Передбачається, що усередині кожної групи дужок немає інших дужок.

Варіант 17.

Дано речення. Напишіть програму, яка редагує це речення, видаляючи символ, що зустрічається підряд, більш ніж один раз.

Варіант 18.

В заданому реченні знайти слова, що починаються з однієї з трьох літер, заданих користувачем.

Варіант 19.

Задано речення. Скласти програму виведення на екран слів, в яких перша і остання літери співпадають.

Варіант 20.

В заданому реченні підрахувати кількість слів, що закінчуються на одну з трьох літер, заданих користувачем.

Варіант 21.

В заданому реченні замінити задану користувачем літеру на сполучення трьох літер, заданих користувачем.

Варіант 22.

У заданому рядку символів подвоїти символ, заданий користувачем (всі такі символи), а всі пробіли видалити.

Варіант 23.

У заданому рядку символів є одна крапка. Визначити кількість символів до крапки і після неї. Вивести ту частину рядка, в якому більше символів.

Варіант 24.

В заданому рядку символів визначити кількість крапок, пробілів і тире. Якого з цих символів найбільше.

Варіант 25.

Задано рядок символів. Користувач вводить два символи. Визначити який з них в цьому рядку зустрічається рідше і на скільки.

Контрольні питання:

1. Що таке рядок в С++ ?
2. Що містить у собі рядок у С ++ ?
3. Яка функція копіює рядки в мові С++?
4. Наведіть приклад ініціювання рядків при оголошені ?
5. Яка функція визначає довжину рядка ?
6. Які операції визначено для рядків типу string ?
7. Яка функція використовується для порівняння рядків між собою ?
8. Як відбувається виділення комірки пам’яті для змінної типу string ?

Назвати функції вводу/виводу, символьного масиву, з клавіатури на екран.

Лабораторна робота №11
Опрацювання текстів.

Мета та зміст роботи:

Отримати навички роботи з рядками в С ++.

В результаті виконання роботи необхідно:

- вчитися використовувати стандартні функції для роботи з рядками;
- вчитися виконувати обробку рядків;
- користуватися покажчиками при зверненні до рядків.

Методичні вказівки

8. Задано рядок, що складається з символів. Символи об'єднуються в слова. Слова один від одного відокремлюються одним або декількома пропусками. В кінці тексту ставиться крапка. Текст містить не більше 255 символів. Виконати введення рядка і обробку рядка відповідно до свого варіанта.
9. Всі програми повинні містити докладні пояснювальні коментарі.

3. Дані, результати і пояснювальні тексти на екрані повинні розташовуватися естетично. Всі математичні розрахунки і перетворення повинні бути викладені в звіті.

Зміст звіту

9. Номер роботы, тему і постановку задачі.
10. Детальний словесний опис кожної програми з описом вхідних даних і вихідних результатів.
11. Програму виконання кожного завдання.
12. Результати роботи кожної програми.
13. Пояснення результатів.
14. Прийом завдання проводиться в такому порядку:
    - написаний або надрукований звіт по роботі (див. Вище);
    - працюючі, повністю налагоджені програми, що містять детальні коментарі і збережені на flash-карту;
    - відповіді на 5-7 питань по лекції 9.

Завдання по варіантах

Обробку рядків організувати використовуючи тільки стандартні функції.

№п.п.	Завдання
1.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. При введенні російського тексту, користувач забув переключити розкладку клавіатури. Напишіть програму, яка перетворює такий рядок в російський текст.
2.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка виводить на екран чотири і більше наступних один за одним без урахування пробілів голосних букв.
3.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка змінює порядок букв в кожному слові на зворотній
4.	Дано два слова A і B. Напишіть програму, яка з'ясовує, чи є слово A перестановкою букв слова B.
5.	Напишіть програму, яка визначає довжину максимальної серії символів, що складається з цифр, в заданій користувачем рядку і виводить їх на екран.
6.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка виводить на екран всі слова що містять певний символ на певній позиції.
7.	Напишіть програму, яка підраховує кількість "щасливих" квитків в рулоні і виводить їх номери на екран. Номер квитка - шестизначне число. Заданий початковий та кінцевий номери.
8.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка знаходить всі слова, в яких буква "a" зустрічається більш ніж один раз.

№п.п.	Завдання
9.	Опишіть функцію, яка повертає **True**, якщо аргументом є голосна буква, і **False** в іншому випадку. Напишіть програму, яка в заданому рядку символів підраховує кількість голосних букв, використовуючи допоміжну функцію.
10.	Напишіть програму, яка перевіряє, чи виконується в заданому рядку баланс дужок.
11.	Напишіть програму, яка перетворює рядок, що складається з прізвища, імені та по батькові співробітника, в рядок що складається з прізвища та ініціалів.
12.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка редагує цей рядок таким чином, щоб кожне слово починалося з великої літери (інші - рядкові).
13.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка змінює порядок букв в кожному слові на зворотній не змінюючи порядку слів в рядку.
14.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка підраховує кількість слів у реченні і виводить їх на екран.
15.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка редагує цю пропозицію, видаляючи символ, що зустрічається поспіль більш ніж один раз.
16.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка редагує цей рядок, видаляючи всі цифри.
17.	Дано рядок, що складається з слів, розділених будь-якою кількістю пробілів. Напишіть програму, яка редагує цей рядок, видаляючи всі символи, крім цифр і крапки.
18.	Напишіть програму, яка в заданому рядку символів підраховує кількість букв.

№п.п.	Завдання
19.	Напишіть програму, яка в заданому рядку символів підраховує кількість цифр.

КОНТРОЛЬНІ ПИТАННЯ:

1. Оголошення рядкових змінних.
2. Функції роботи з рядками.
3. Методи обробки рядків.
4. Ввід та вивід рядків на екран.
5. Метод string.
6. Використання функції gets().
7. Конкатенація рядків.
8. Копіювання рядків.
9. Порівняння рядків.
10. Логічні операції.

Лабораторна робота № 12
Використання стандартних функцій.

Мета та зміст роботи:

Отримати навички роботи з функціями в С ++.

В результаті виконання роботи необхідно:

- вчитися використовувати стандартні функції для роботи;
- вчитися створювати функції користувача;
- користуватися вказівниками при зверненні до параметрів функцій.

Завдання 1:

Складіть програму-меню для розв'язування кількох задач. Користувач вибирає варіанти розв'язування: 1 – розв'язування першої задачі, 2 – розв'язування другої задачі. Розв'язання кожної з них оформити у вигляді підпрограми. Результати розв'язування кожної задачі виведіть у центрі екрана будь-яким кольором на будь-якому фоні.

Завдання (по варіантах):

1. Комп'ютер зчитує чотири різних числа й обчислює:

1) різницю між найбільшим і найменшим числами;
2) добуток першого та третього чисел.

2. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) добуток цих чисел;
2) квадрат різниці двох середніх за значенням чисел.

3. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) відношення найбільшого числа до найменшого;
2) квадрат суми двох менших за значенням чисел.

4. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) відношення найменшого числа до найбільшого;
2) суму квадратів цих чотирьох чисел.

5. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) добуток між найбільшим і найменшим числами;

2) квадрат суми двох більших за значенням чисел.

6. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) квадрат суми найменшого та найбільшого числа;
2) різницю сум квадратів першого і третього та другого і четвертого чисел.

7. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) добуток парних чисел;
2) відношення першого числа до найбільшого.

8. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) добуток непарних чисел;
2) відношення другого числа до різниці найбільшого і найменшого.

9. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) суму другого та четвертого чисел;
2) квадрат різниці двох менших за значенням чисел.

10. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) квадрат найменшого числа;
2) відношення першого числа до найменшого.

11. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) куб найбільшого числа;
2) квадрат різниці двох більших за значенням чисел.

12. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) відношення першого числа до суми найбільшого і найменшого;
2) різницю коренів квадратних першого і найменшого чисел.

13. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) куб найменшого числа;
2) різницю між квадратом найбільшого і квадратом найменшого чисел.

14. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) різницю квадрата найбільшого числа та кореня з найменшого числа;
2) середнє арифметичне чисел.

15. Комп'ютер зчитує чотири натуральних числа й обчислює:

1) різницю кубів найбільшого і найменшого чисел;
2) суму коренів квадратних першого і другого чисел.

16. Комп’ютер зчитує чотири натуральних числа й обчислює:

1) суму кубів двох середніх за значенням чисел;
2) середнє арифметичне першого і третього чисел.

Лабораторна робота № 13
Функції користувача.

Мета та зміст роботи:

Отримати навички роботи з функціями в С ++.

В результаті виконання роботи необхідно:

- вчитися використовувати стандартні функції для роботи;
- вчитися створювати функції користувача;
- користуватися вказівниками при зверненні до параметрів функцій.

№п.п.	Завдання
1.	Опишіть функцію, яка виводить на екран рядок з N зірочок і переводить курсор на новий рядок. Складіть програму,для отримання зображення в якому в першому рядку 1 зірочка, в другому - 2, у третьому -3, в рядку з номером m - m зірочок. Кількість рядків задає користувач.
2.	Опишіть функцію, для знаходження ступеня числа. Обчисліть значення виразу $\mathbf{x^2 + x^4 + x^6 + \ldots + x^{2*n}}$.
3.	Визначте функцію, яка повертає скалярний добуток двох векторів на площині $\overline{AB} = A_x B_x + A_y B_y$, где A_x, A_y - координати вектора Визначте функцію, яка повертає модуль заданого вектора на площині. $\lvert A\rvert = \sqrt{A_x^2 + A_y^2}$, де A_x, A_y, B_x, B_y - координати векторів A і B відповідно Визначте функцію, яка повертає кут між двома векторами на площині, використовуючи певні функції. $\varphi = \arccos\left(\frac{\overline{AB}}{\lvert A\rVert B\rvert}\right)$ Обчислити кут між двома векторами, заданими користувачем, використовуючи певну функцію.

№п.п.	Завдання
4.	Описати дві функції, що обчислюють за стороною a рівностороннього трикутника його периметр **P = 3a** і площу(a - вхідний параметр і є цілим числом). $S = \frac{a^2\sqrt{3}}{4}$ За допомогою них знайти периметри і площі восьми рівносторонніх трикутників зі сторонами, які вводить користувач.
5.	Визначте функцію, яка повертає відстань між двома точками на площині $\boldsymbol{d} = \sqrt{(\boldsymbol{x}_1 - \boldsymbol{x}_2)^2 + (\boldsymbol{y}_1 - \boldsymbol{y}_2)^2}$, де $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{y}_1, \boldsymbol{y}_2$ - координати першої і другої точки відповідно. Визначте функцію, яка повертає площу трикутника з заданими вершинами на площині, використовуючи формулу Герона і раніше визначену функцію $\boldsymbol{S} = \sqrt{\boldsymbol{p}(\boldsymbol{p}-\boldsymbol{a})(\boldsymbol{p}-\boldsymbol{b})(\boldsymbol{p}-\boldsymbol{c})}$, де $\boldsymbol{p} = (\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c})/2$ - півпериметр трикутника, $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$ - довжини сторін трикутника Обчислити площу трикутника з заданими користувачем вершинами, використовуючи визначену функцію.
6.	Описати функції, що знаходять кількість цифр цілого додатнього числа K, і їх суму. За допомогою цієї функції знайти кількість і суму цифр для кожного з п'яти даних цілих чисел, заданих користувачем.
7.	Описати функцію, яка міняє порядок проходження цифр цілого додатнього числа K на зворотний . За допомогою цієї функції поміняти порядок проходження цифр на зворотний для кожного з п'яти даних цілих чисел.

№п.п.	Завдання
8.	Визначити функцію, яка повертає скалярний добуток двох векторів на площині. $\overline{AB} = A_x B_x + A_y B_y$ Обчислити скалярний добуток двох векторів, заданих користувачем, використовуючи визначену функцію.
9.	Описати функцію, яка знаходить максимальну цифру в запису даного натурального числа. За допомогою цієї функції знайти максимальну цифру для кожного з семи заданих користувачем цілих чисел.
10.	Описати функцію, яка знаходить найбільший дільник заданого числа. Знайти найбільші дільники шести заданих чисел.
11.	Визначити функцію, яка повертає суму чисел натурального числа. Знайти теософську суму числа, введеного користувачем, використовуючи визначену функцію (обчислювати суму цифр числа, до тих пір, поки не вийде цифра).
12.	Визначити функції знаходження площі правильних многокутників за відомою довжиною сторони і кількості його сторін. Знайти площу семи різних многокутників. Кількість сторін і довжину задає користувач.
13.	Визначити функцію, яка повертає **True**, якщо аргумент є простим числом і **False** в іншому випадку. Вивести всі прості числа від N1 до N2, використовуючи визначену функцію.
14.	Визначити функцію, яка повертає суму зазначеного числа непарних натуральних чисел. Використовуючи визначену функцію показати, що сума непарних натуральних чисел дорівнює квадрату цілого числа. Кількість непарних натуральних чисел задає користувач.

№п.п.	Завдання
15.	Визначити функцію, яка повертає натуральне число, якщо аргумент функції є квадратом цього числа і нуль в іншому випадку. Складіть програму, яка знаходить всі натуральні числа, менші ніж N, для яких виконується співвідношення $a^2 + b^2 = c^2$.
16.	Визначити функцію, яка повертає натуральне число, якщо аргумент функції є квадратом цього числа і нуль в іншому випадку. Знайдіть всі можливі цілі значення довжин сторін прямокутного трикутника в діапазоні від 1 до N (N задає користувач), використовуючи визначену функцію
17.	Визначити функцію перевірки числа на простоту. Вивести всі прості числа на проміжку від n до m.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Караванова Т.П. Інформатика. Методи побудови алгоритмів та їх аналіз. - К.:Генеза. 2007. 216 с.
2. Ашарина, И.В. Основы программирования на языках С и С++ / И.В. Ашарина. - М.: ГЛТ, 2012. - 208 с.
3. Дорогов, В.Г. Основы программирования на языке С: Учебное пособие / В.Г. Дорогов, Е.Г. Дорогова; Под общ. ред. проф. Л.Г. Гагарина. - М.: ИД ФОРУМ, НИЦ ИНФРА-М, 2013. - 224 с.
4. Страуступ, Б. Язык программирования С++. Специальное издание / Б. Страуступ. - М.: Бином, 2015. - 1136 с.
5. Кениг, Э. Эффективное программирование на С++. Практическое программирование на примерах. Т. 2 / Э. Кениг, Б.Э. Му. - М.: Вильямс, 2016. - 368 с.
6. МакГрат, М. Программирование на С для начинающих / М. МакГрат. - М.: Эксмо, 2015. - 192 с.
7. Мартынов, Н.Н. Программирование для Windows на С / Н.Н. Мартынов. - М.: БИНОМ, 2013. - 528 с.
8. Перри, Г. Программирование на С для начинающих / Г. Перри, Д. Миллер. - М.: Эксмо, 2015. - 368 с.
9. 9. Фленов, М.Е. Программирование на С++ глазами хакера. / М.Е. Фленов. - СПб.: BHV, 2012. - 352 с.
10. Хенкеманс, Д. Программирование на С++ / Д. Хенкеманс, М. Ли. - СПб.: Символ-плюс, 2015. - 416 с.
11. Вирт Н. Алгоритмы и структуры данных. - М.: Мир, 1989. - 360 с.
12. Бочков С.О. Субботин Д.М. Язык программирования С для персонального компьютера. - М.: Радио и связь, 1990.
13. М. Эллис, Б. Страуструп. Справочное руководство по языку С++ с комментариями: Пер. с англ. - Москва: Мир, 1992. 445с.
14. Стенли Б. Липпман. С++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.

15. Бруно Бабэ. Просто и ясно о Borland C++: Пер. с англ. - Москва: БИНОМ, 1994. 400с.
16. В.В. Подбельский. Язык С++: Учебное пособие. - Москва: Финансы и статистика, 1995. 560с.
17. Г. Шилдт. Самоучитель С++: Пер. с англ. - Санкт-Петербург: BHV-Санкт-Петербург, 1998. 620с.
18. У. Сэвитч. С++ в примерах: Пер. с англ. - Москва: ЭКОМ, 1997. 736с.
19. К. Джамса. Учимся программировать на языке С++: Пер. с англ. - Москва: Мир, 1997. 320с.
20. Х. Дейтел, П. Дейтел. Как программировать на С++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с.
21. Керниган Б., Ритчи Д. Язык программирования С. - М.: Финансы и статистика, 1985.
22. Керниган Б., Ритчи Д., Фьюэр А. Язык программирования С. Задачи по языку С. - М.: Финансы и статистика, 1985.
23. Кормен Т.,Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, 2-е издание – М.: Вильямс, 2006.
24. Абельсон Х., Сассман Д. Структура и интерпретация компьютерных программ. М.: Добросвет, 2006
25. Уильям Топп, Уильям Форд. Структуры данных в С++: Пер. сильям Топп, Уильям Форд. Структуры данных в С++: Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2000. – 816 с.
26. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. М.: Вильямс, 2000
27. В. Липский Комбинаторика для программистов.-1988. 200 с.
28. С.Окулов Программирование в алгоритмах.- 2006. 220с
29. Серджвик Р. Фундаментальные алгоритмы на С.Часть 1-3 СПб: Диасофт, 2003.
30. http://www.cplusplus.com/
31. http://www.e-olymp.com/
32. http://www.firststeps.ru/cbuilder/r.php?1

33. http://teacher.ucoz.net/index/lekcii_po_s/0-4
34. http://object.newmail.ru/obj4.html
35. http://www.rsdn.ru/article/cpp/fastdelegate.xml
36. http://www.cyberguru.ru/programming/cpp/cpp-velvet-way-page76.html