МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Основи алгоритмізації та програмування»
для студентів спеціальності

173 – «Авіоніка»

всіх форм навчання

2020

Конспект лекцій з дисципліни «Основи алгоритмізації та програмування» для студентів спеціальності 173 – «Авіоніка» всіх форм навчання. / Укл. С.І.Арсеньєва. – Запоріжжя: НУ «Запорізька політехніка», 2020. -110 с.

Укладач: С.І.Арсеньєва, доцент, к.ф-м.н.

Рецензент: В.В. Корольков, доцент, к.т.н.

Відповідальний

за випуск: зав. нав. лаб. каф. ЕПА К.І.Пилипенко

Затверджено на засіданні кафедри

“Електропривод та автоматизація

промислових установок ”

Протокол № 5 від 26.10.2020 р.

Рекомендовано до видання НМКЕТФ

Протокол № 3 від 19.11.2020 р

ЗМІСТ

1 АЛГОРИТМ ТА ЙОГО ВЛАСТИВОСТІ

1.1 Основні поняття

Саме слово "алгоритм" походить від прізвища персидського математика Аль Хорезми, який в IX столітті розробив правила чотирьох арифметичних дій (сьогодні б сказали алгоритми арифметичних дій).

Алгоритм – це опис деякої послідовності дій, що приводить до розв'язання задачі, яка була поставлена.

Алгоритм – система чітких однозначних вказівок, яка визначає послідовність дій над деякими об'єктами та після певного числа кроків призводить до отримання потрібного результату.

Алгоритми бувають чисельними та логічними[1].

Алгоритми, згідно яким розв'язання задач може бути зведене до арифметичних дій називаються *чисельними алгоритмами*.

Алгоритми, згідно яким розв'язання задач може бути зведене до логічних дій, називаються *логічними алгоритмами* (алгоритми пошуку мінімального числа, пошук шляху в лабіринті).

Провідними властивостями алгоритму є:

а) *Дискретність* – розподілення виконання розв'язання задачі на окремі операції.

Під *дискретністю* розуміється те, що алгоритм складається з опису послідовності кроків обробки, організованих таким чином, що у початковий момент задається вихідна ситуація, а після кожного наступного кроку ситуація перетворюється на підставі даних, отриманих на попередніх кроках обробки. Дискретність алгоритму означає, що він виконується по кроках: кожна дія, передбачена алгоритмом,

виконується тільки після того, як закінчилась попередня, тобто перетворення вихідних даних відбувається у часі дискретно.

б) *Детермінованість (визначеність)* – кожна команда алгоритму повинна однозначно визначати дії виконавця.

Ця властивість означає , що на кожному кроці алгоритму однозначно визначається перетворення даних, що були отримані на попередніх кроках алгоритму, тобто на однакових вихідних даних алгоритм повинен завжди давати однакові результати.

в) *Результативність (кінцевість)* – завершення роботи алгоритму за конечне число кроків (при цьому кількість кроків може бути заздалегідь не відомою та різною для різних вихідних даних).

г) *Масовість (універсальність)* - алгоритм рішення задачі розробляється в загальному вигляді, тобто можливість вирішення класу задач, що розрізняються лише вихідними даними. При цьому вихідні дані вибираються з деякої області, званої областю застосовності алгоритму.

д*) Зрозумілість* - зміст допустимого набору команд, зрозумілого конкретному виконавцю. Кожен крок алгоритму повинен обов'язково представляти собою якусь допустиму дію, тобто алгоритм будується для конкретного виконавця автором і повинен бути їм обом зрозумілий. Це полегшує перевірку і модифікацію алгоритму при необхідності.

1.2. Форми запису алгоритмів

Процес складання алгоритмів називають алгоритмізацією.

Алгоритм, який реалізує рішення задачі, можна уявити різними способами - за допомогою графічного або текстового опису.

Графічний спосіб представлення алгоритмів має ряд переваг завдяки візуальності і явного відображення процесу виконання завдання. Алгоритми, представлені графічними засобами, отримали назву блок-схем (рис.1.1).

Текстовий опис алгоритму є досить компактним і може бути реалізований на природній мові або спеціальною (алгоритмічною) мовою у вигляді програми.

Всі способи подання алгоритмів можна вважати взаємодоповнюючими один одного. На етапі проектування алгоритмів найкращим способом є графічне представлення, а на етапах перевірки і застосування алгоритму - текстовий запис у вигляді програми.

Правила виконання блок-схем:

Блок-схемою називається наочне зображення алгоритму, коли окремі дії (етапи алгоритму) зображуються за допомогою різних геометричних фігур (блоків), а зв'язки між етапами (послідовність виконання етапів) вказуються за допомогою стрілок, що з'єднують ці фігури.

Виконання блок-схем здійснюється згідно відповідного стандарту.

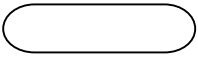
При виконанні блок-схем всередині кожного блоку вказується пояснююча інформація, яка характеризує дії, що виконуються цим блоком. Потоки даних в схемах показуються лініями. Напрямок потоку зліва направо і зверху вниз вважається стандартним. У випадках, коли необхідно внести велику ясність в схему або потік має напрямок відмінний від стандартного, на лініях використовуються стрілки, що вказують цей напрямок.

У схемах слід уникати перетину ліній. Пересічні лінії не мають логічного зв'язку між собою, тому зміни напрямку в

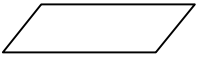
точках перетину не допускаються. Якщо дві або більше вхідних лінії об'єднуються в одну вихідну лінію, то місце об'єднання ліній зміщується.

Кількість вхідних ліній не обмежена,але лінія, що виходить з блоку, повинна бути одна, за винятком логічного блоку.

Основними елементами блок-схем є:

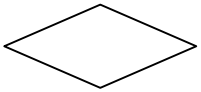


- початок (кінець) алгоритму;



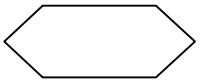
- блок вводу-виводу даних;

- блок обчислень;



- логічний блок, в якому напрямок потоку інформації обирається в залежності від деякої умови;

- процес (підпрограма);



- блок модифікації, в якому функція виконує дії, що змінюють деякі назви (наприклад заголовок циклу);

- з’єднувач, використовується для вказівки зв’язку між потоками інформації;

- між сторінковий з’єднувач, тобто вказівник зв’язку між інформацією на різних листах.

Рисунок 1.1 - Елементи блок – схем

1.3. Дані та їх типи

Алгоритм, який реалізує рішення задачі, завжди працює з даними.

Дані - це будь-яка інформація, подана у формалізованому вигляді і придатна для обробки алгоритмом.

По відношенню до програми дані діляться на вхідні, проміжні та вихідні (рис. 1.2).

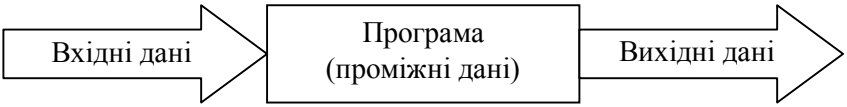


Рисунок 1.2. - Схема розподілу даних

Дані, відомі перед виконанням алгоритму, є *початковими, вхідними даними*. Дані, що використовуються в процесі виконання програми, є *проміжними даними*. Результат розв'язання задачі - це *кінцеві, вихідні дані*.

Дані поділяються на змінні і константи.

Змінні - це такі дані, значення яких можуть змінюватися в процесі виконання алгоритму.

Константи - це дані, значення яких не змінюються в процесі виконання алгоритму.

Будь-яка величина має 3 *основні властивості*:

 ім'я, яке задається ідентифікатором, що представляє собою послідовність літер і цифр, що починаються з літери;

 значення;

 тип даних - це така характеристика даних, яка задає безліч допустимих значень і визначає безліч операцій, які можна до цих даних застосувати.

Типи даних ділять на 2 групи:

а) *прості (скалярні)* типи - містять одне єдине значення. До них відносяться:

 цілий тип - визначає підмножина допустимих значень з безлічі цілих чисел (наприклад: 23, -12);

 дійсний тип - визначає підмножина допустимих значень з безлічі цілих і дробових чисел в деякому діапазоні (наприклад: 2,5; -0,01; 3,6109);

 логічний тип - змінна приймає тільки два значення: істина (true) і брехня (false);

 символьний тип - будь-які символи комп'ютерного алфавіту (наприклад: 'a', '5', '+').

б) *структуровані* типи - описують набори однотипних або різнотипних даних (тобто містять кілька значень), з якими алгоритм повинен працювати як з однієї іменованою змінною. До них відносяться: масиви, рядки, множини і т.д

1.4 Основні структури алгоритмів

Основні структури алгоритмів (ОСА) - це певний набір блоків і стандартних способів їх з'єднання для виконання типових послідовностей дій.

ОСА використовуються при структурному підході до розробки алгоритмів і програм, який передбачає використання декількох основних структур, комбінація яких дає все різноманіття алгоритмів і програм.

До основних алгоритмічних структур належать лінійні,ті,що розгалужуються і циклічні структури.

1.5 Лінійні алгоритми

Лінійними називаються алгоритми, в яких дії здійснюються послідовно один за одним(рисунки 1.3-1.4).

Приклад: Розробити блок-схему алгоритму обчислення площі та периметра прямокутника за двома заданими сторонам *а* і *в*.

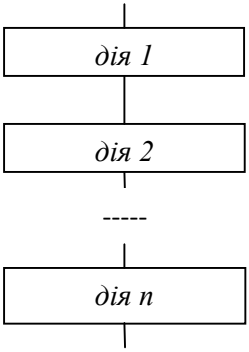


Рисунок 1.3. – Блок – схема алгоритму лінійного процесу

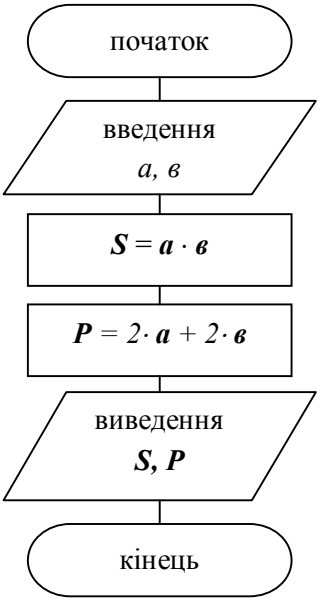
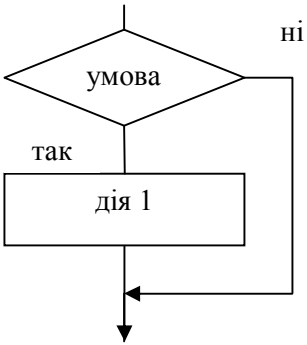


Рисунок 1.4. - Блок-схема обчислення площі та периметра

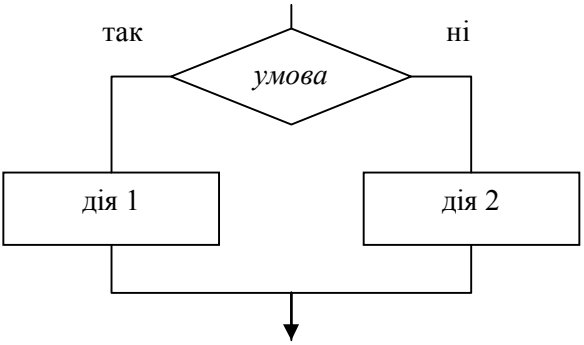
1.6 Алгоритми, що розгалужуються.

Алгоритмом, що розгалужуються, називається такий алгоритм, в якому дія виконується за однією з можливих гілок рішення задачі, в залежності від виконання умов (рис.1.5).

Як умова в алгоритмів, що розгалужуються, може бути використано будь-зрозуміле виконавцеві твердження, виражене як словами, так і формулою. Воно може виконуватись (бути істинним) або не виконуватись (бути помилковим). Таким чином, алгоритм розгалуження складається з умови і двох послідовних дій.



а) для однієї дії;



б) для двох дій;

Рисунок 1.5. – Блок-схеми алгоритму розгалуження а) б)

Приклад: Розробити блок-схему алгоритму обчислення z, якщо дані два дійсних числа *x* і *y*.

$$Z = \begin{cases} X - Y, \text{якщо } X > Y \\ Y - X + 1, \text{якщо } X <= Y \end{cases}$$

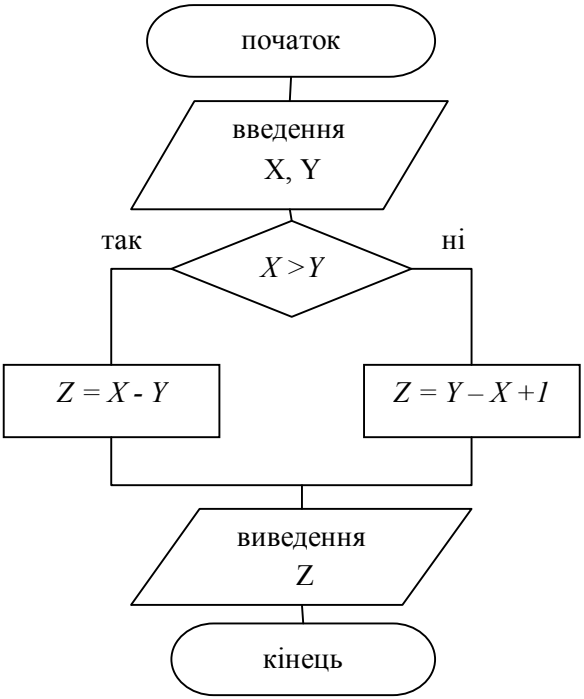


Рисунок1.6. - Блок-схему алгоритму обчислення за прикладом**.**

1.7 Циклічні алгоритми

Циклічним називається алгоритм, в якому деяка частина операцій виконується багаторазово.

Цикл - послідовність дій, що виконуються багаторазово, кожен раз при нових значеннях параметра.

Для організації циклу необхідно:

- задати перед циклом початкове значення змінної, що змінюється в циклі;

-змінювати змінну перед кожним новим повторенням циклу;

-перевіряти умова закінчення або повторення циклу;

- управляти циклом, тобто переходити до його початку, якщо він незакінчений, або виходити з нього після закінчення.

Останні три функції виконуються багаторазово.

Змінна, що змінюється в циклі, називається *параметром*.

У цикл входять в якості базових наступні структури: блок перевірки умови і блок, званий тілом циклу.

Залежно від способу організації числа повторень розрізняють 3 типи циклів:

- цикл з передумовою (цикл-ПОКИ);
- цикл з наступною умовою (цикл-ДО);
- цикл з параметром (цикл з лічильником).

Якщо умова істинна, то тіло циклу виконується і управління передається знову на обчислення умови, якщо ж умова помилкова, то тіло циклу не виконується і відбувається вихід з циклу.

Цикл з передумовою має наступний вигляд:

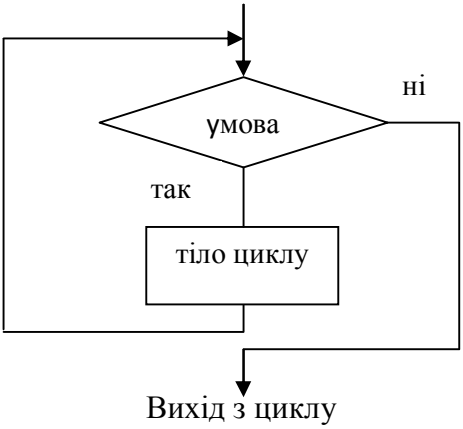


Рисунок1.7. - Блок – схема циклу з передумовою

Якщо умова помилкова, то знову виконуються оператори тіла циклу, якщо ж умова істинна, то цикл закінчується.

Цикл з наступною умовою має наступний вигляд:

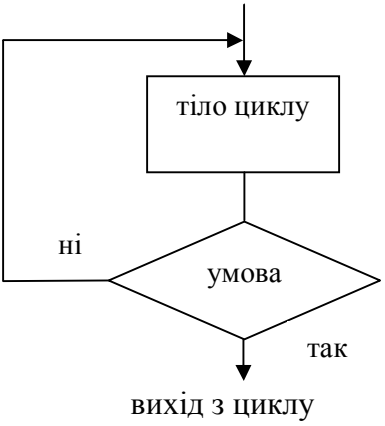


Рисунок 1.8. - Блок–схема циклу з наступною умовою

Циклічні алгоритми, в яких тіло циклу виконується задане число раз, реалізуються за допомогою циклу з параметром. У цьому випадку передбачається повторне виконання тіла циклу з одночасною зміною за правилом арифметичної прогресії значення, що присвоюється параметру циклу.

Цикл з параметром має наступний вигляд:

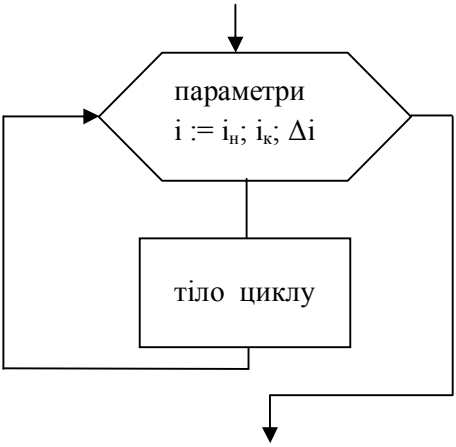


Рисунок 1.9. – Блок–схема циклу з параметром

При обчисленні кінцевої суми в циклічному алгоритмі попередньо необхідно початкову суму прирівняти нулю (S = 0), а при обчисленні кінцевого додатку - початковий додаток прирівняти одиниці (P = 1).

Приклад: Розробити блок-схему алгоритму обчислення факторіала (F) натурального числа N. Факторіал числа (!) - це добуток всіх натуральних чисел від 1 до N.

$$N! = 1*2*3*...*N, (0! =1)$$

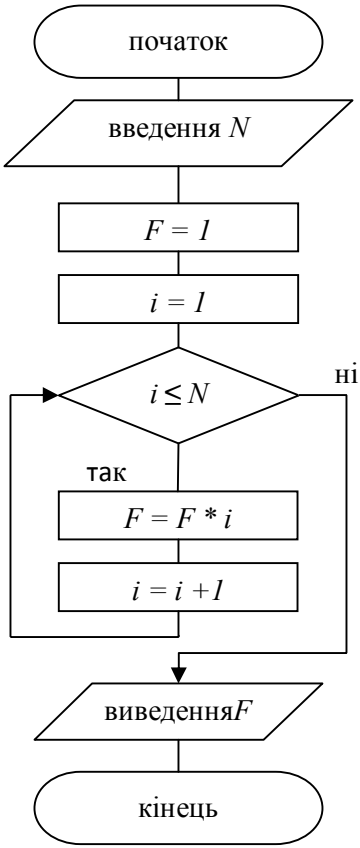


Рисунок 1.10 – Блок–схема алгоритму обчислення факторіалу через цикл з попередньою умовою;

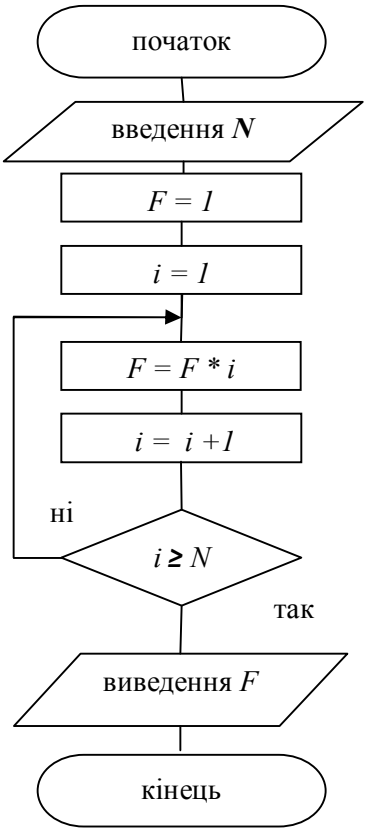


Рисунок 1.11. – Блок–схема алгоритму обчислення факторіалу через цикл з подальшою умовою;

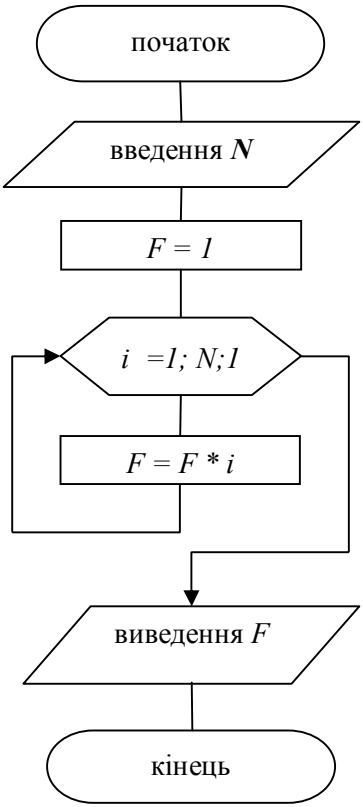


Рисунок 1.12. – Блок–схема алгоритму обчислення факторіалу через цикл з параметром.

2 ОСНОВИ ПРОГРАМУВАННЯ МОВОЮ С/С++ [2]

2.1. Історія та сучасність

Як відомо, мова С була створена Денісом Рітчі на початку 70-х років в Bell Laboratory американської корпорації AT&T. Протягом тривалого періоду ця мова була однією з основних універсальних мов програмування. На початку мова С призначалася для системного програмування, однак у подальшому вона виявилася виключно ефективною також і для створення прикладних програм. Серед переваг мови С відзначають переносність програм на комп'ютери різної архітектури та з однієї операційної системи в іншу, лаконічність запису алгоритмів, логічну стрункість і читабельність програм, можливість одержати ефективний код програми, порівняний по швидкості із програмами, написаними на мові асемблера. Широка область використання мови С обумовлена ще й тим, що вона є мовою високого рівня, яка підтримує модульність, блокову структуру програм, високий рівень абстракцій, широкі можливості розширення системи програмування та її адаптації до будь-якої предметної області, можливість роздільної компіляції модулів. У той же час, мова має набір низькорівневих засобів, що дозволяють програмувати доступ до апаратних засобів комп'ютера і зовнішніх пристроїв, зокрема, добратися до будь-якого біта в просторі оперативної пам’яті або в просторі введення/виведення.

Мова С++ з'явилася у відповідь на більш високі вимоги до мови програмування з боку тих, хто створює програми та системи програм високої складності і, у подальшому, займається їхньою модернізацією. До теперішнього часу мова С++ стала найбільш вживаною мовою як в області системного,

так і проблемного програмування. У даному посібнику представлена мова, яку ми позначаємо як C/C++, маючи на увазі те, що називають С-підмножиною мови С++. По суті це та частина мови С++, яка реалізує сферу компетенції класичної мови С але з певними елементами модернізації та деякими розширеннями.

У наш час для створення складних програмних продуктів мало хто користується тільки лише засобами класичної мови С. І, в той же час, далеко не завжди для створення якісних прикладних програм потрібно використовувати весь величезний спектр можливостей об'єктно-орієнтованого програмування (ООП) на С++. Метою вивчення дисципліни є допомогти Вам засвоїти основи програмування на C/C++, а також ті можливості мови С++, які не пов'язанні прямо з технологією ООП. У зв'язку з тим, що ми будемо вивчати підмножину мови С++, протягом всієї книги ми будемо вести мову саме про мову С++. Успішне засвоєння викладеного в посібнику матеріалу дозволить Вам надалі, якщо виникне потреба, легко й безболісно засвоїти основи об'єктно-орієнтованого програмування мовою С++. Всі описи, приклади і вправи, що приводяться у книзі, відповідають стандарту мови С++ ISO/IEC 14882 (1998).

Приклади і вправи, які розглядаються у курсі лекцій , орієнтовані на так звані консольні прикладні програми [ ]. Це обумовлено двома причинами. З одного боку, саме консольні програми є базовим видом прикладних програм. З іншого, таке обмеження дозволяє у процесі занять краще зосередитися саме на питаннях мови й власне програмування, не обтяжуючи себе необхідністю вникати в деталі програмування елементів графічного інтерфейсу, бо останнє є досить специфічним і складає предмет окремої дисципліни. Такого ж роду методичним

міркуваннями виправдане й активне використання в прикладах і вправах бібліотечного модуля **syst.h** [2], який було розроблено у Запорізькому національному технічному університеті (автор Пінчук В.П.). Відзначимо, що хоча вказаний модуль і не входить до складу стандартних пакетів систем С++ - програмування, його використання робить тексти програм, що наводяться, більш лаконічними і прийнятними та дозволяє більше уваги приділяти власне програмуванню, використанню мовних засобів та алгоритмам. У той же час відсутність бібліотечного модулю syst.h у стандартних пакетах програмування не складає проблеми для тих, хто користується даним посібником: вказаний модуль та детальний опис щодо його застосування міститься на компакт-диску, що додається до посібника.

2.2. Загальна структура програми. Два простих приклади

Формально, текст будь-якої С++-програми будується з елементів наступних видів:

- директиви;
- оголошення;
- функції;
- коментарі.

Директиви

Процес компіляції програми складається із двох фаз: препроцесорна обробка тексту програми й власне компіляція. При виконанні першої фази текст вихідної програми піддається певним змінам, залишаючись при цьому текстом мови С++. Результатом виконання другої фази є об'єктний файл, що має розширення .obj. Обидві фази виконуються однією програмою - компілятором. Директиви дозволяють керувати препроцесорною

обробкою тексту програми й самим процесом компіляції. Запис директиви починається символом # і розміщується в окремому рядку. Прикладом часто використовуваної директиви є директива включення:

#include <stdlib.h>

Замість трикутних дужок можна записати лапки:

#include "stdlib.h"

В процесі компіляції програми такий рядок буде замінено текстом, що міститься у файлі **stdlib.h**. Трикутні дужки вказують на те, що файл **stdlib.h** буде відшукуватися у системному каталозі, шлях до якого прописано при інсталяції С++-пакету (звичайно це папка **..\INCLUDE**). Якщо ж ім'я файлу включено у лапки, то його пошук спочатку буде здійснено у поточному (робочому) каталозі і, якщо його не буде знайдено, пошук буде продовжено у системному каталозі.

Оголошення

Оголошення являє собою запис, що містить опис деяких об'єктів. У мові С++ оголошення є різновидом операторів і повинні завершуватися символом «**;**» . Оголошення, також як і оператори, виконуються в процесі роботи програми і результатом виконання оголошення в більшості випадків є створення імен об'єктів та самих об'єктів з відповідними властивостями.

Оголошення, що не являється частиною тіла якої-небудь функції, є глобальним оголошенням. Об'єкт, створений таким оголошенням також називається глобальним, він може бути як простим, так і динамічним. Простором імен, створених глобальними оголошеннями, є програма в цілому.

Приклад оголошення змінних дійсного типу без ініціалізації:

double x, y, z;

Оголошення двох простих константних об'єктів цілого типу з ініціалізацією:

const int N=10, M=25;

Функції

Функції виконують роль головних будівельних блоків, з яких будується будь-яка програма. Саме функції містять опис алгоритмів, що використовуються в програмі. У програмі обов'язково повинна бути функція з ім'ям **main** (головна функція). Вона є головною у тому сенсі, що виконання програми починається з виконання цієї функції і робота програми завершується при завершенні її роботи.

Коментарі

Коментарі використовуються для того, щоб зробити текст програми більше зрозумілим і зручним для читання. У процесі компіляції коментар ігнорується й, отже, може розташовуватися в будь-якому місці програми. Можна використовувати однорядкові і багаторядкові коментарі. Для запису однорядкових коментарів застосовується пара символів "//". Приклад запису однорядкового коментарю:

// *текст, що може продовжуватися до кінця рядка*

Для запису багаторядкового коментарю використовуються комбінації символів "/*" та "*/". Вони діють як своєрідні дужки, відокремлюючи певну частину тексту програми. Приклад запису багаторядкового коментарю:

/* *текст,*
який може розташовуватися
***у декількох рядках* */**

Наступні приклади двох простих програм дадуть перше

уявлення щодо загальної структури програми.

Приклад 1

Програма по заданих розмірах катетів **a**,**b** прямокутного трикутника знаходить величину його гіпотенузи **c** і площу **S**. Застосовуються такі формули для розрахунків:

$$c = \sqrt{a^2 + b^2} \ ,$$

$$S = \frac{1}{2} a \cdot b \ .$$

```
// Задача про прямокутний трикутник
#include <syst.h>       // підключення універсального
                        // бібліотечного модуля
  void main()
{ double a,b,c,S;       // оголошення змінних
  printf("a,b = "); scanf("%lf %lf",&a,&b);
                        // введення даних
  c=sqrt(a*a+b*b);      // розрахунок довжини гіпотенузи
  S=a*b/2;              // розрахунок площі трикутника
  printf("c = %lf  S = %lf \n",c,S);
                        // виведення результату
  pause;                // пауза до натискання клавіші
}
```

Розглянемо програму більш докладно.

Перший рядок є коментарем. Другий рядок повідомляє компілятор про підключення універсального бібліотечного модуля **<syst.h>**. У третьому рядку записаний заголовок головної (і єдиної у цьому прикладі) функції програми, що має ім'я **main**. У наступних рядках записане тіло функції **main**, яке, так як і тіло будь-якої іншої функції, заключається у фігурні

дужки. Тіло функції починається з оголошення змінних дійсного типу **a**, **b**, **c**, **S**. У мові C++ будь-яка змінна повинна бути оголошена перш ніж вона буде використана. Наступний рядок

printf("a,b = "); scanf("%lf %lf",&a,&b);

містить два оператори. Перший оператор є викликом стандартної бібліотечної функції виведення **printf**, вона виводить рядок підказки "a,b = ". Другий оператор – виклик стандартної функції **scanf** для введення із клавіатури вихідних даних - довжин катетів трикутника. Першим аргументом цієї функції є рядок "%lf %lf", він вказує, що будуть вводитися два дійсних числа подвійної точності (тип double). Наступні аргументи – адреси змінних, котрим повинні привласнитися значення, що вводяться. Для одержання адреси змінних **a** і **b** використовується унарна операція із символом **&** (амперсанд).

Наступний рядок програми містить обчислення довжини катету **c** та операцію присвоєння:

c=sqrt(a*a+b*b);

Правий операнд операції присвоєння містить виклик бібліотечної функції **sqrt**, що обчислює квадратний корінь.

У рядку

S=a*b/2;

обчислюється площа трикутника **S**.

Рядок

printf("c = %lf S = %lf \n",c,S);

служить для виведення результатів обчислень на екран комп'ютера. Тут викликається функція **printf**, перший аргумент якої називається керуючим рядком (або рядком форматів), а наступні аргументи - це змінні, значення яких повинні виводитися на екран. Основне призначення керуючого рядка - управління форматом інформації, що виводиться. Для управління форматами використовуються спеціальні сполучення

символів - специфікації формату. Кожна специфікація починається з символу **%**. Специфікація формату **%lf** вказує, що буде виводитися дійсне число в десятковому записі. Символи керуючого рядка, які не є специфікацією формату, просто виводяться у поточний рядок на екран. Комбінація символів **\n** позначає керуючий байт (керуючий символ), виведення якого викликає переміщення текстового курсору у початок нового рядка. Символ **\n** називають символом завершення рядка.

Далі в програмі записано оператор

pause;

який насправді є макросом, опис якого міститься в заголовковому файлі **<syst.h>**. Цей оператор призупиняє виконання програми до натискання якої-небудь клавіші. Така пауза часто застосовується для того, щоб втримати робоче вікно редактора програми на екрані при запуску її з оболонки типу Borland C++ або Microsoft Visual C++. Оператор **pause;** іноді можна замінити викликом стандартної бібліотечної функції **getch()** або деякими іншими функціями.

Приклад 2

У наступному прикладі програми розв'язується квадратне рівняння виду $ax^2 + bx + c = 0$. У програмі передбачений автоматичний вибір форми виводу результату в залежності від властивостей квадратного рівняння: якщо корені є комплексними, вони виводяться в такій формі: $x_1 = u+vi$, $x_2 = u-vi$, де u,v - дійсна й уявна частини комплексних коренів.

Алгоритм розрахунку.

Спочатку обчислюється дискримінант рівняння:

$$D = b^2 - 4ac \cdot$$

Далі, якщо D>0, тоді існують дійсні корені і вони обчислюються за формулами:

$$x_1 = \frac{1}{2a}(-b+\sqrt{D}),$$

$$x_2 = \frac{1}{2a}(-b-\sqrt{D}).$$

Якщо ж D<0, тоді обчислюються комплексні корені:

$$x_1 = -\frac{b}{2a}+\frac{\sqrt{-D}}{2a}\cdot i,$$

$$x_2 = -\frac{b}{2a}-\frac{\sqrt{-D}}{2a}\cdot i.$$

```
// Розв'язування квадратного рівняння
#include <syst.h>
  void main()
{ double a,b,c,D,x1,x2,re,im; // об'ява змінних
          // введення коефіцієнтів квадратного рівняння
  printf("a,b,c = "); scanf("%lf %lf %lf",&a,&b,&c);
  D=sqrt(b)-4*a*c;      // обчислення дискримінанта
                        // перевірка знака дискримінанта
  if (D>=0) {     // обчислення дійсних коренів рівняння
            x1=(-b+sqrt(D))/(2*a);
            x2=(-b-sqrt(D))/(2*a);
            printf("The roots are real:\n");

printf("x1= %lf \n",x1);
            printf("x1= %lf \n",x2);
          }
  else {      // обчислення комплексних коренів рівняння
            re= -b/(2*a); im= abs(sqrt(-D)/(2*a));
            printf("The roots are complex:\n");
            printf("%lf + %lf*i \n",re,im);
```

```
            printf("%lf - %lf*i \n",re,im);
        }
}
```

Тут функція **abs(x)** обчислює абсолютну величину аргументу. Викликається не стандартна бібліотечна функція **abs(x)**, а функція з універсального модуля **syst.h** [2]. Докладніше про цю та інші функції з модулю syst.h буде сказано нижче.

2.3 Обєкти та ідентифікатори

Об'єкти і їхні атрибути
Алфавіт мови та лексеми
Ідентифікатори

2.3.1. Об'єкти та їхні атрибути

Одним з базових понять сучасного програмування є поняття об'єкта. Для того, щоб пояснити його, згадаємо, що комп'ютерну програму можна уявляти собі як деякий механізм, основним призначенням якого є обробка інформації. Існує такий закон: оброблювана комп'ютерами інформація завжди структурована. Тобто вона являє собою набір (або потік) інформаційних одиниць певних типів. Тип інформаційної одиниці є комплексною характеристикою, яка визначає її семантику й основні атрибути: розмір, спосіб бінарного кодування та ін. Об'єктом називають область оперативної пам'яті, яку виділено для збереження інформаційної одиниці певного типу. Значенням об’єкту є саме інформаційна одиниця, що розташована в об'єкті.

Важливим моментом тут є те, що об'єкт - це область

пам'яті, що асоціюється з деяким наперед визначеним типом значень - інформаційних одиниць. Сучасні мови програмування дозволяють мати справу з об'єктами, що мають часом досить складну структуру. В складних програмах об'єкт іноді можна уявляти собі як інформаційну модель деякої сутності. Поміщена в такий об'єкт інформаційна одиниця визначає стан (а можливо й поведінку) цієї самої сутності.

З об'єктом зв'язують 5 атрибутів: адреса, розмір, ідентифікатор, значення і тип. Адресою об'єкта є адреса його першого байта. Розміром об'єкта вважають кількість байтів оперативної пам’яті, які він займає. Ідентифікатор являє собою ім'я об'єкта. Значення об'єкта - це поміщена в нього інформаційна одиниця. Значення заноситься в об'єкт шляхом виконання таких операцій, як присвоєння, ініціалізація, введення значення із зовнішнього середовища.

Тип об'єкта є комплексним атрибутом, він визначає:

- множину (або діапазон) можливих значень об'єкта;
- спосіб двійкового подання значення;
- набір припустимих операцій над значенням об'єкта й правила (алгоритми) їх виконання;
- розмір об'єкта.

Об’єкти можуть бути глобальними та локальними.

Оголошення глобального об’єкта записується поза межами будь-якої функції. Створюються глобальні об’єкти до початку роботи функції **main**, а знищуються після завершення роботи функції **main**. Розміщуються вони у сегментах даних виконуваного коду програми.

Об’єкти, оголошення яких записано у тілі будь-якої функції, належать до категорії локальних об’єктів. Локальний об’єкт створюється при виконанні його оголошення і знищується при виході з поточного блоку. Розташовуються

локальні об'єкти як правило у сегменті стека коду програми.

Операції створення і вилучення глобальних і локальних об'єктів виконуються програмою автоматично шляхом виклику спеціальних функцій - конструкторів та деструкторів.

Як глобальні, так і локальні об’єкти можуть бути динамічними і не динамічними. У попередніх абзацах мова йшла про не динамічні об’єкти. Динамічні об'єкти створюються спеціальними операціями new і new[] та знищуються при виконанні операцій delete і delete[]. Розміщуються вони в купі (в області оперативної пам'яті, яка розподіляється динамічно або heap-області пам’яті).

В об’єктно-орієнтованому програмуванні об'єкт є структурованим, він може включати деякий набір різнотипних одиниць даних та функцій. Компонентом об’єкту може бути теж деякий об’єкт. Об’єкти - компоненти об'єкту можуть мати різні (і будь-які) типи. Призначенням функцій, які входять до складу об’єкту, є виконання певних дій з компонентами-даними об'єкта.

2.3.2. Алфавіт мови та лексеми

Будь-яка мова програмування являє собою деяку формально-логічну систему. Всі записи і конструкції програми будуються на основі алфавіту мови. Символи цього алфавіту - найменші будівельні елементи, з яких складається вихідний текст програми. Алфавіт мови С++ включає наступні символи:

- прописні й рядкові латинські літери і символ підкреслення;
- арабські цифри від 0 до 9;
- набір спеціальних знаків;
- пробільні символи: пробіл, символ табуляції, символ завершення рядка.

Із символів алфавіту будуються наступні елементи тексту програми - так звані лексеми. Лексема є осмислена у термінах даної мови програмування найкоротша послідовність символів. У мові С/С++ застосовуються такі лексеми:

- ідентифікатори;
- ключові (службові, зарезервовані) слова;
- символи операцій;
- константи;
- роздільники.

Стандарт С++ передбачає 63 ключові слова. Повний список ключових слів можна одержати, працюючи в оболонці С++ і використовуючи систему контекстної допомоги, задавши для одержання довідки слово **keyword**.

2.3.3 Ідентифікатори

Ідентифікатор являє собою ім'я деякої сутності, яка використовується в програмі. Ідентифікаторами забезпечуються об'єкти, функції, константи, типи, мітки. Кожна з мов програмування має свої правила вибору ідентифікаторів. За правилами мови С++ ідентифікатор є послідовність латинських літер, цифр і деяких спеціальних символів, причому послідовність повинна починатися з літери або символу підкреслення. Довжина ідентифікатора довільна, але значущими є тільки перші 32 символи. Важливо відзначити, що в С/С++ при розрізненні ідентифікаторів ураховується регістр. Це означає, що такі імена як **Alfa** і **alfa** вважаються різними. Ніякий ідентифікатор не повинен збігатися із ключовим словом мови. Ключові слова мають спеціальні значення для компілятора. Вживання ключових слів строго регламентовано і ці слова не

можуть використовуватися інакше. Список ключових слів мови С++ наведено нижче.

asm	**auto**	**break**	**case**	**catch**
char	**class**	**const**	**default**	**delete**
do	**double**	**else**	**enum**	**extern**
float	**friend**	**goto**	**if**	**inline**
int	**long new**	**operator**	**protected**	**public**
register	**return**	**short**	**signed**	**sizeof**
static	**switch**	**template**	**this**	**throw**
try	**typedef**	**union**	**unsigned**	**void**
volatile	**while**	**continue**	**for**	**private**
struct	**virtual**			

Ідентифікатор об'єкта має свій набір атрибутів:

- тип;
- сфера дії;
- область видимості;
- клас пам'яті;
- простір імен.

Сферою дії ідентифікатора є область програми, у межах якої зберігається зв'язок між ідентифікатором і відповідним об'єктом. Категоріями сфери дії є: блок, функція, клас, файл, програма.

Областю видимості ідентифікатора є область програми, з якої можливий нормальний доступ за його допомогою до відповідного об'єкта. Категорії видимості: блок, функція, клас, файл.

Ім'я об'єкта в зовнішньому блоці може "екрануватися" ім'ям об'єкта внутрішнього блоку. Такий об’єкт у межах внутрішнього блоку є недоступним. При виході із внутрішнього

блоку доступ відновлюється. Для доступу із внутрішнього блоку до однойменного об'єкту зовнішнього блоку можна використати операцію дозволу доступу із символом "**::**".

Клас пам'яті ідентифікатора визначає спосіб розміщення об'єкта в пам'яті. Категорії класу пам'яті: регістри, сегмент даних, стек, купа.

Простором імен є та частина програми, у межах якого будь-який ідентифікатор повинен бути унікальним. Наприклад: для міток простором імен є функція, для імен компонентів класу - клас, для змінної - сфера дії ідентифікатора.

Нижче наведений приклад, що ілюструє поняття області видимості, сфери дії й простору імен. У програмі формуються 4 простори імен: одне глобальне і ще 3 вкладених. У кожному з них оголошується змінна з ім'ям R. Незважаючи на збіг імен, програма працює нормально.

```
// Приклад 1
#include <syst.h>
int R=0;                  // змінна в глобальному просторі
  void main()             // (рівень 0)
{ int R=1;                // змінна в просторі рівня 1
  { int R=2;              // змінна в просторі рівня 2
     { int R=3;           // змінна в просторі рівня 3
       printf("Level 3: R=%d \n", R);
     }
   printf("Level 2: R=%d \n", R);
  }
  printf("Level 1: R=%d \n", R);
                          // застосовано операцію дозволу
                          // доступу до глобального простору
```

```
  printf("Level 0: R=%d \n", ::R);
}
```

Протокол роботи програми:

Level 3: R=3
Level 2: R=2
Level 1: R=1
Level 0: R=0

Ідентифікатор не повинен збігатися зі службовим словом.

2.3.4. Вправи

Вправа 1

Придумайте слушне ім'я для кожної з наступних змінних:

а) змінна, значенням якої є швидкість автомобіля;

б) змінна, у якій зберігається інформація про деяку людину;

в) змінна, у якій зберігається найбільш висока оцінка, яку можна одержати на іспиті.

Вправа 2

Які з наведених нижче ідентифікаторів є правильними:

alfa , #abc , _AB_ , abc! , Alfa12 , $100 , 2005y , M$, sWp , Beta@ , aa__aa , gamma& , _X_ , delta? , ab.c .

Вправа 3

Скільки з наступних ідентифікаторів позначають різні об'єкти:

alfa , Alfa , _BETA_ , _beta_ , a , A , A_ , a1_b1 , A1_B1.

Вправа 4

Які з наведених нижче рядків не можна застосувати як ідентифікатор об’єкту:

Friend , friend , DO , do , _case_ , long+ , Union , integer , if_else ,

1_2_3_4 , _1_2_3_ , oper , type , try , del , alfa# , _ASM_ , Const .

2.4 Вирази

Поняття виразу. Вирази Lvalue та Rvalue
Операції. Пріоритети та асоціативність

2.4.1. Поняття виразу. Вирази Lvalue та Rvalue

До складу операторів і оголошень можуть входити вирази. Виразом називають фрагмент тексту програми (коду), виконання якого формує (обчислює) деяке значення, або адресу, або посилання на об’єкт. Можна сказати так, що головною особливістю виразу є те, що його можна обчислити. У термінах мови C/C++ можна також казати, що вирази виконуються. Особливістю нашої мови є те, що виконання виразу може призводити до зміни значень змінних, які в нього входять. По завершенню обчислення виразу для збереження отриманого значення створюється тимчасовий об'єкт. Ім’я у такого об’єкту відсутнє. Після завершення поточного оператора тимчасовий об'єкт знищується. Тимчасовий об’єкт, як і будь-який інший, належить до певного типу. Цей тип ототожнюють з типом самого виразу.

Вираз є запис, який будується з наступних елементів: константи, змінні, функції, символи операцій і дужки. Вираз може мати один із двох видів: Lvalue, Rvalue. До виду Rvalue відносяться вирази, результатом виконання яких є значення, до виду Lvalue - вирази, результатом виконання яких є посилання на деякий об'єкт. Rvalue може перебувати тільки праворуч від знака присвоювання "=", Lvalue може перебувати як ліворуч, так і праворуч від нього. Наведемо декілька прикладів:

```
a = x + y + z;                  // тут маємо Lvalue = Rvalue;
*(p + 3) = sin(x) - 3*cos(2*x); // Lvalue = Rvalue;
*(p + i) = *(p + k - 1);        // Lvalue = Lvalue;
```

2.4.2. Операції. Пріоритети та асоціативність

Операції мови C/C++ діляться на операції-функції та операції-процедури. Операція-функція не змінює значення своїх операндів. Вона має єдиний ефект: обчислює та повертає знайдене значення або посилання. Більшість операцій є саме такими.

До операцій-процедур відносять операції, що змінюють значення своїх операндів. Виконання операції-процедури призводить одразу до двох ефектів: змінюється значення операнду і формується значення, яке повертається як результат виконання операції.

По кількості операндів операції діляться на унарні (з одним операндом), бінарні (із двома операндами) і тернарні (із трьома операндами). Усього є 49 операцій, які можна розділити на 5 груп:

- арифметичні;
- операції порівняння й логічні операції;
- побітові операції;
- операції присвоювання;
- інші операції.

Арифметичні операції

До арифметичних операцій відносяться:

+ - додавання;
- - віднімання та унарний мінус;
* - множення;
/ - ділення;
% - залишок від ділення (ділення по модулю);
++ - інкрементування;
-- - декрементування.

Операції "+", "-", "*" й "/" діють так, як і в більшості інших мов програмування. Вони можуть застосовуватися до будь-яких чисельних типів даних. Якщо операнди мають однаковий тип, то результат арифметичної операції має той же тип. Тому якщо, наприклад, обидва операнди мають цілий тип, то результат також буде мати цілий тип. Відзначимо, що останнє відноситься також і до операції ділення. Так, значення виразу 7/3 буде дорівнює 2, а обчислення виразу 1/3 дасть нуль. Треба бути уважними при застосуванні операції ділення у арифметичних виразах.

Якщо операнди операції +, -, * або / мають не однаковий тип, вони автоматично приводяться до старшого типу. При цьому вважається, що старшинство типу зростає у такій послідовності: char, short, int, long int, float, double, long double . Наприклад, якщо змінна m має тип int, а змінна x - тип double, то обчислення виразу m + x дасть результат типу double.

Операція **%** дає залишок від ділення. Ця операція може бути застосована тільки до цілочисельних даних. У наступному прикладі обчислюється ціла частина й залишок від ділення двох цілих чисел.

```
// Приклад 1
#include <syst.h>
  void main()
{ int a, b;                          // оголошення цілих змінних
  printf("a, b = "); scanf("%d %d",&a,&b);
  printf("a/b = %d \n", a/b);
  printf("a\%b = %d \n", a%b);
}
```

Протокол роботи програми:

```
Введіть два цілих числа -> 7 3
Ціла частина = 2
Залишок від ділення = 1
```

Група арифметичних операцій включає ще дві корисні операції, специфічні саме для C/C++. Це унарні операції інкрементування ++ та декрементування --. Операція інкрементування збільшує значення змінної на одиницю, а операція декрементування - зменшує на одиницю. Відносяться вони до операцій-процедур.

Кожна з операцій інкрементування та декрементування має дві модифікації: префіксну та постфіксну. Відрізняються вони порядком виконання дій. Префіксна операція (**++x** або **--x**) спочатку змінює значення змінної, а після цього повертає нове значення змінної-операнду. Постфіксна операція **x++** або **x--** спочатку повертає поточне значення змінної **x**, а вже далі змінює його відповідним чином.

У прикладі, наведеному нижче, кожен з операторів приводить до одного і того ж кінцевого результату:

a=a+1; ++a; a++;

Відзначимо, що вираз **++a** належить до Lvalue (повертається посилання на модифікований об'єкт **a**), у той час як вираз **a++** є Rvalue (повертається значення **a** до його модифікації).

Особливості виконання операції інкрементування можна побачити на такому прикладі:

```
// Приклад 2
#include <syst.h>
  void main()
{ int a=4, b, c=0;
  b=a++;                                          // b=4
  printf("a=%d  b=%d  c=%d \n", a, b,c);
  c=++a;                                       // c=6
  printf("a=%d  b=%d  c=%d \n", a, b,c);
}
```

Протокол роботи програми:

```
a=5  b=4  c=0
a=6  b=4  c=6
```

Операції порівняння й логічні операції

До цієї групи відносяться такі операції:

==	- дорівнює;
!=	- не дорівнює;
<	- менше;
<=	- менше або дорівнює;
>	- більше;
>=	- більше або дорівнює;

! - логічне заперечення (інверсія);
|| - логічне АБО (диз'юнкція);
&& - логічне І (кон'юнкція).

Операція "!" є унарною, решта - бінарні операції. Кожна з цих операцій повертає значення логічного типу, використовуються вони в логічних виразах. Декілька прикладів логічних виразів:

a == 3, g > 35, h <= t, x>0 || x%10 == 0 .

Значення логічного виразу має тип bool, множина значень якого складається з двох елементів: false (не істина, числовий еквівалент 0) і true (істина, числовий еквівалент 1). Кожне з логічних значень має цілочисловий еквівалент. В програмі, що наведена нижче, значення логічних змінних t і f виводяться як цілі числа.

```
// Приклад 3
#include <syst.h>
  void main()
{ bool t, f;
  t = (200>45);                    // значення виразу "істина"
  f = (200<45);            // значення виразу "неправда"
  printf("t is %d \n", t);
  printf("f is %d \n", f);
}
```

Протокол роботи програми:

```
  t is 1
  f is 0
```

Логічні операції в мові С відповідають операціям математичної логіки: **!** - заперечення (інверсія), || - операція

логічного додавання (або диз'юнкція або "логічне АБО"), **&&** - операція логічного множення (або кон'юнкція або "логічне І"). Нижче наведено таблицю істинності для цих операцій.

Таблиця 2.1. Таблиця істинності

x	y	!x	x || y	x && y
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

Особливістю логічної операції || (АБО) є таке: якщо при обчисленні результату операції (вираз1) || (вираз2) - значення лівого операнду (вираз1) буде **true**, то значення другого операнда на результат операції вже не впливає, він завжди буде дорівнювати **true**. У цьому випадку другий операнд і не обчислюється. Аналогічна ситуація має місце і для операції && (логічне І): якщо лівий операнд дорівнює значенню **false**, то результат буде **false** незалежно від значення правого операнду.

Побітові логічні операції

Побітові логічні операції виконуються над кожною парою відповідних бітів цілочислових операндів. Список побітових операцій:

~ - заперечення (інверсія);
& - операція І;
| - операція АБО;
^ - заперечення рівнозначності (не еквівалентність);
<< - зсув бінарного коду вліво;
>> - зсув бінарного коду вправо.

Операція "~" є унарною, решта операцій - бінарні. Побітові операції виконуються для операндів будь-якого цілочислового

типу. Операнди дійсного типу не допускаються.

Таблиці істинності для операції ~, &, ^, | такі ж самі, як і для відповідних логічних операцій. Тільки в цьому випадку порівнюються не значення виразів, а значення кожної відповідної пари бітів (двійкових розрядів) значень операндів.

Побітові операції дозволяють програмувати операції над окремими бітами інформації. Вони часто використовуються у програмах, що забезпечують роботу пристроїв різного призначення.

Розглянемо приклад застосування побітових операцій. Щоб установити значення старшого розряду однобайтової змінної **ch** рівним одиниці зручно використати побітову операцію АБО. При виконанні наступного оператора

ch = ch | 128;

старший біт символьної змінної **ch** буде встановлено в одиницю.

Якщо ж потрібно встановити в нуль старший біт символьної змінної, то доцільно використати побітову операцію "&":

ch = ch & 127;

Результатом операції зсуву << є зсув всіх бітів лівого операнду на кількість позицій (двійкових розрядів), що задає вираз праворуч, вліво. Результатом операції >> є зсув бітів вправо. Двійкові позиції, що звільняються, заповнюються нулями. Формат запису для цих операцій виглядає так:

змінна << кількість_позицій,

змінна >> кількість_позицій.

Нехай, наприклад двійкове подання числа A є 00001001, тоді після виконання наступних операцій будемо мати:

B=A<<3; // B = 01001000

C=A>>3; // C = 00000001

При використанні операцій зсуву відбувається втрата старших або молодших двійкових розрядів.

Операції присвоювання

Група операцій присвоювання складається з операції простого присвоювання і набору операцій комбінованого присвоювання. Операція простого присвоювання із символом "=" формує (обчислює) значення правого операнду й привласнює його об'єкту, що відповідає лівому операнду. При виконанні операції комбінованого присвоювання спочатку виконується відповідна бінарна операція, а потім отриманий результат привласнюється об'єкту, що відповідає лівому операнду. У мові C/C++ визначені наступні операції комбінованого присвоювання:

+= - додати й привласнити;
-= - відняти й привласнити;
*= - помножити й привласнити;
/= - розділити й привласнити;
%= - розділити націль й привласнити;
>>= - виконати зсув бітів вправо й привласнити;
<<= - виконати зсув бітів вліво й привласнити;
&= - виконати побітову кон’юнкцію й привласнити;
|= - виконати побітову диз’юнкцію й привласнити;
^= - виконати побітову операцію заперечення рівнозначності й привласнити.

На відміну від інших мов програмування у мові C/C++ присвоювання є не оператор, а операція. Операція простого присвоєння "=", також як і решта операцій комбінованого присвоєння є бінарними операціями, результатом їх виконання є посилання на лівий операнд. Лівим операндом може бути вираз виду Lvalue, правим - вираз Rvalue або Lvalue.

У виразі **(z=x+y)>0** спочатку обчислюється величина **x + y**, яка далі привласнюється змінній **z**, отримане значення **z** порівнюється з нулем.

Допускається запис ланцюжка операцій присвоєння:

x=y=z=a/d;

У такому виразі операції присвоювання виконуються справа наліво.

Використання операцій комбінованого присвоювання дозволяє записувати оператори більш лаконічно. Так, замість оператора

k = k+3; (2.1)

можна записати оператор

k+=3; (2.2)

Крім стилістичного моменту тут важливим є те, що комбіноване присвоєння виконується швидше, ніж відповідна бінарна операція й наступне присвоєння. Тут, щоправда, слід зазначити, що при завданні відповідного режиму оптимізації для компілятора, при виконанні операторів (2.1) і (2.2) буде побудована та ж сама послідовність інструкцій. Якщо це так, тоді вибір із двох зазначених вище операторів - усього лише справа стилю запису програми й особистих переваг програміста.

Операції простого та комбінованого присвоювання відносяться до операцій-процедур, бо вони змінюють значення лівого операнду.

Інші операції

До групи інших операцій відносяться операції:

&x - одержати адресу змінної x, унарна операція, символом операції є "&"; *p - операція повертає посилання на об'єкт із адресою p, унарна операція, символом операції є "*";

::x - доступ до змінної x, яку визначено в глобальному просторі імен, унарна операція, символом операції є "::";

S.x - повертається посилання на компонент x структурованого об'єкта S, бінарна операція, символом операції є ".";

S.*p - повертається посилання на компонент структурованого об'єкта S з відносною адресою p, бінарна операція, символом операції є ".*";

pS->x - повертається посилання на компонент x того об'єкта, який має адресу pS, бінарна операція, символом операції є "->";

pS->*p - повертається посилання на компонент з відносною адресою p для об’єкта з адресою pS, бінарна операція, символом операції є "->*";

L ?x : y - повертається посилання на x, якщо значенням логічного виразу L є "істина" або посиланні на y, якщо ні, операція має три операнди, символом операції є "?:";

sizeof(x) або sizeof x - операція повертає розмір об'єкта x у байтах, унарна операція, символом операції є "sizeof";

sizeof(тип) - повертається розмір об'єкта зазначеного типу, унарна операція, символом операції є "sizeof";

(тип)x - перетворити значення x до зазначеного типу, унарна операція, символом операції є "(тип)";

(тип*)p - перетворити значення покажчика p до зазначеного типу, унарна операція, символом операції є "(тип*)";

new тип - створити об'єкт зазначеного типу, повертається адреса створеного об'єкту, повертається адреса створеного масиву, унарна операція, символом операції є "new";

new тип [розмір] - створити масив об'єктів зазначеного типу, повертається адреса створеного масиву, унарна операція, символом операції є "new[]";

delete p - знищити об'єкт з адресою p, це унарна операція яка нічого не повертає, символом операції є "delete";

delete[] p - знищити масив об'єктів з адресою p, унарна операція, символом операції є "delete[]";

A,B - виконати послідовно вирази A, B, повернути значення B;

M[k] - операція індексування, вибрати елемент масиву M з номером k, бінарна операція з символом "[]";

ім'я(параметри) - викликати функцію із зазначеними ім'ям та параметрами, бінарна операція з символом "()".

Розглянемо деякі з перерахованих вище операцій. Про інші мова йтиме у наступних розділах посібника.

Операція вибору - єдина операція, що має три операнди. Вона записується у такий спосіб:

s ? x : y

Тут **x**,**y** - вирази, **s** - логічний вираз. Якщо **s** має значення **true**, результатом виконання операції буде **x**, інакше - **y**. У тому випадку, коли **x**,**y** є виразами виду Lvalue, то операція **?:** повертає посилання на **x** або на **y**. Наведемо два приклади застосування операції вибору.

absa = (a>0) ? a : -a;

У цьому рядку виконується обчислення абсолютної величини числа a.

min = (x<y)?x:y;

Тут визначається мінімальне з двох значень.

За допомогою операції вибору можна дуже лаконічно запрограмувати, наприклад, таку операцію: із двох змінних **x** і **y** тій, що має менше значення, привласнити значення 0:

(x<y)?x:y = 0;

Наведемо ще один приклад використання тернарної операції вибору. У наступному фрагменті програми підраховується кількість елементів у числовому масиві x, які не є кратними 3. Результат заноситься в змінну K.

```
int  i, K = 0;
for  (i=0;i<N;i++)  K+= x[i]%3 ? 1:0 ;
```

Операція sizeof має дві форми: sizeof(вираз) і sizeof (тип). Результатом цієї операції є розмір відповідного об'єкта в байтах. У першому випадку вираз, зазначений в дужках, не обчислюється, замість цього визначається його тип і, потім, розмір об'єкта, що відповідає цьому типу.

Операція із символом "," (кома) має найнижчий пріоритет із всіх операцій. Виконується вона зліва направо, її значенням є значення правого операнда. У виразі

вираз1, вираз2

спочатку виконується **вираз1** потім **вираз2**. Значення **вираз2** і буде результатом операції. Прикладом використання цієї операції є такий оператор циклу:

```
for (i=1,sum=0; i<=10; i++ ) sum+=exp(i);
```

Пара виразів i=1, sum=0 виконує ініціалізацію змінної циклу **i** та обнуляє початкове значення **sum** для підсумовування.

Порядок обчислення виразу, пріоритети і асоціативність операцій

При написанні програми важливо уявляти собі, у якому порядку обчислюється той чи інший вираз. Порядок при обчисленні виразу визначається:

- круглими дужками: у першу чергу обчислюється підвираз у внутрішній парі дужок;
- пріоритетами операцій: спочатку виконуються операції з більш високим пріоритетом;
- асоціативністю операцій: операції з однаковим пріоритетом виконуються або зліва направо, або справа наліво, залежно від асоціативності операції.

Виклик функції має більш високий пріоритет, ніж будь-яка

інша операція. Символи операцій, їх категорії пріоритетів і асоціативності наведені нижче в таблиці 2. Найвищим вважається пріоритет 1, а найнижчим - пріоритет 16.

Таблиця 2.2.Категорії пріоритетів і асоціативність операцій

Пріоритет	О п е р а ц і я	Асоціативність
1	() [] -> :: .	→
2	! ~ -- ++ & * (тип) sizeof new delete	←
3	.* ->*	→
4	* / %	→
5	+ -	→
6	<< >>	→
7	< <= > >=	→
8	== !=	→
9	&	→
10	^	→
11	|	→
12	&&	→
13	||	→
14	?:	→
15	= *= /= %= += - = &= ^= |= <<= >>=	←
16	,	→

Послідовність обчислення виразу можна задати примусово, розставивши належним чином дужки.

Переповнення при виконанні операції

При виході результату виконання цілочисельної операції за

межі інтервалу відповідного типу (при переповненні розрядної сітки) переривання роботи програми стандартними засобами не відбувається. При виконанні операцій з дійсними типами вихід значення за межі припустимого інтервалу генерує сигнал виключної ситуації, у відповідь на яку операційна система завершує програму аварійно, з виведенням відповідного повідомлення.

2.5 Оператори

Види операторів
Стандартні оператори
Оголошення змінних та ініціалізація
Константи й константні об'єкти

2.5.1. Види операторів

Оператори являють собою засоби опису алгоритмів. У мові C++ розрізняють 5 видів операторів:

- оператор-оголошення;
- стандартний оператор;
- виклик процедури;
- оператор-вираз;
- складений оператор.

Будь-який оператор завершується крапкою з комою. Виключеннями є:

- складений оператор, що завершується фігурною дужкою;
- оголошення функції, що також завершується фігурною дужкою.

Оператор-оголошення (або просто оголошення) містить описи атрибутів об'єктів, що об’являються, або опис функції, структури, класу, об'єднання. При виконанні оператора-оголошення для об'єкта можуть виконуватися наступні дії:

а) створення імені об'єкту, яке заноситься в таблицю ідентифікаторів компілятора і, таким чином, воно стає відомим компіляторові;

б) створення об'єкту зазначеного в оголошенні типу;

в) ініціалізація створеного об'єкта.

У певних випадках може виконуватися тільки перший етап, або перший і другий етапи або всі три етапи. Згідно правил С++ оголошення, які породжують об’єкти, можна записувати в будь-якому місці програми.

Стандартні оператори реалізують типові елементи алгоритмів, такі як цикли, розгалуження, умовні й безумовні переходи і т.і. Предметна розмова про стандартні оператори буде нижче.

Процедурою називають функцію, що не повертає значення. Тобто процедура то є функція, для якої тип значення, що повертається, записано як **void**. Виклик процедури завершується символом ";" і , таким чином, він є оператором.

Оператор-вираз - це будь-який вираз, який завершується крапкою з комою. Нижче наведені приклади запису деяких операторів-виразів. Мається на увазі, що a,b,c,z - змінні, k - ціла змінна, p - покажчик.

```
z=(a+b+c)/3;    // обчислити вираз та привласнити його
                // значення змінній z;

a++, b++;       // збільшити значення a на одиницю,
                //повернути b і після цього,
```

```
                    // збільшити b на одиницю;

*(p+k)--;           // адресу p зсунути на k об'єктів уперед,
                    //взяти об'єкт за отриманою адресою,
                    //зменшити його значення на 1;

*(++p)=a++;         // адресу p збільшити на розмір одного
                    // об'єкта, значення змінної a
                    // привласнити об'єкту з отриманою
                    // адресою,після цього значення
                    // змінної a збільшити на одиницю.
```

Складеним оператором називають послідовність операторів, яку включено у фігурні дужки:

{ опер_1; опер_2; ... }

Складений оператор формально є єдиним оператором, його можна писати всюди, де можна використовувати один оператор. Наприклад, як тіло оператора циклу **while** або оператора, що записується в складі оператора **if**.

Якщо складений оператор містить, принаймні, одне оголошення, він називається блоком. Блок утворює власний простір імен. Будь-який ідентифікатор, оголошений у деякому блоці, існує на відрізку від моменту виконання його об’яви до виходу за фігурну дужку, що завершує поточний блок.

2.5.2. Стандартні оператори

Далі у записах формату операторів квадратні дужки позначають необов’язковий елемент оператора.

Оператор безумовного переходу має наступний формат:

goto *мітка*;

Мітка являє собою ідентифікатор, яким можна позначити будь-який оператор. Мітка відокремлюється від оператора двокрапкою:

***мітка*: *оператор*;**

Простором імен мітки є функція, у якій вона визначена.

Оператор goto рекомендується використовувати тільки у виняткових ситуаціях. Інтенсивне використання оператора goto та надлишок міток у програмі, роблять програму незручною для читання і малозрозумілою, воно вважається поганим стилем програмування.

Прикладом виправданого використання мітки є організація виходу із системи вкладених циклів:

```
for (...)
    { for (...)
       { while (...)
          { ........
            goto label;
            ...
          }
       }
    }
label: printf("Вихід із вкладених циклів \n");
```

Особливістю цього оператора в C++ є те, що він не може вказувати перехід уперед, переступаючи через оголошення змінної з ініціалізацією. У такій ситуації компілятор видає повідомлення “Goto bypasses initialization of a local variable”. У той же час запис такого переходу назад допускається.

Оператор вибору має такий формат:

if (ЛВ) *опер_1*; [else *опер_2*;]

Тут через ЛВ позначається логічний або цілочисловий вираз. Якщо значенням ЛВ є "true" або ціле число, що не дорівнює нулю, виконується оператор ***опер_1***, а оператор ***опер_2*** не виконується. І навпаки, якщо ЛВ = false, оператор ***опер_2*** виконується, а оператор ***опер_1*** не виконується.

Подивимось, як можна використати оператор **if** для визначення знака дійсного числа.

```
// Приклад 1
#include  <syst.h>
  void main()
{ double a;
  printf ("a = "); scanf("%lf", &a);
  if (a==0) printf ("a == 0 \n");
  if (a<0)  printf ("a < 0 \n");
  if (a>0)  printf ("a > 0 \n");
}
```

Іноді зручним виявляється використання конструкції виду **if-else-if**:

```
if (ЛВ1) опер_1;
   else if (ЛВ2) опер_2;
       else if опер_3;
            .....
              else опер_N;
```

У такому записі умови перевіряються зверху вниз. Як тільки яке-небудь зі значень ЛВ1, ЛВ2, ... прийме значення “true”, виконається наступний оператор, а вся інша частина конструкції буде зігнорована. Програма з попереднього

прикладу може бути переписана ще й у такому вигляді:

```
// Приклад 2
#include  <syst.h>
  void main()
{ double a;
  printf ("a = "); scanf("%lf", &a);
  if (a<0)      printf ("a<0 \n");
  else if (a==0) printf ("a == 0 \n");
    else       printf ("a > 0 \n");
}
```

Оператор багатоваріантного вибору реалізує точку множинного розгалуження алгоритму. Цей оператор має такий формат:

```
switch (S)        { case C1: операторы; break;
                    case C2: операторы; break;
                    .....................
                  [ default : операторы ]
            };
```

S являє собою вираз, який може мати будь-який цілий тип (char, short, int, long, enum, bool). Константи **C1**, **C2**, повинні мати типи, які відповідають виразу **S**, називаються вони мітками варіанту. Якщо значення виразу **S** збігається з однією з міток варіанта, виконується перехід на мітку цього варіанту. Допоміжний оператор break завершує роботу оператора switch. Якщо значення виразу **S** не збігається з жодною міткою варіанта, виконується перехід на рядок **default**, якщо він присутній. Після цього робота оператора **switch** завершується.

Для того, щоб забезпечити виконання тільки одного з варіантів, передбачених у структурі **switch**, наприкінці кожного зі списків "оператори" необхідно записати оператор **break**. При

відсутності цього оператора виконуються всі наступні варіанти, записані в тілі switch (за винятком варіанта **default)** до наступного оператора **break**.

Нижче наведено приклад використання оператора **switch** для обробки натискання клавіш 'y', 'n', Esc.

```
// Приклад 3
#include  <syst.h>
  void main()
{ char x;
  char *s1 = "Key y was pressed";
  char *s2 = "Key n was pressed";
  char *s3 = "Key Esc was pressed";
  for (;;) {        x=getkey();
          switch (x) { case 'y':  puts(s1);  break;
                       case 'n':  puts(s2);  break;
                       case  27:  puts(s3);  exit(0);
                     }
     }
}
```

На початку функції **main()** оголошується символьна змінна x, значення якої присвоюється оператором **x=getkey()**. Наступні рядки програми містять оголошення символьних рядків **s1** і **s2**, які виводяться на екран за допомогою функції **puts**. Оператор **break** здійснює вихід з тіла оператора **switch** .

Для програмування циклів ми маємо оператори циклу трьох видів. Оператор циклу **while** найбільш простий, він має такий формат:

while (ЛВ) оператор;

Так само, як і в операторі if, ЛВ може бути логічним або

цілочисловим виразом. Робота оператора - тіла циклу наступним разом повторюється, якщо ЛВ має логічне значення true або еквівалентне йому ціле значення. Тіло циклу повинно бути єдиним оператором. Якщо треба, щоб тіло циклу включало більше, ніж один оператор, застосовуємо фігурні дужки (тобто створюємо складений оператор).

Зверніть увагу на таке: якщо початкове значення ЛВ дорівнює false, оператор тіла циклу, не буде виконаний жодного разу.

Нижче наводиться приклад використання оператора while для визначення найбільшого загального дільника двох цілих чисел. Наведений текст є реалізацією відомого алгоритму Евкліда:

```
int  a=125,  b=2500, r;
while  (r = a%b)  { a=b;  b=r; }
printf("НЗД = %d \n", b);
```

Зверніть увагу на те, що перевірка завершення циклу **while** у цьому прикладі реалізована не зовсім звичайним способом. Вираз **r = a%b** у другому рядку має подвійне призначення: з одного боку, він обчислює залишок від ділення (і є однієї з операцій, передбачених алгоритмом Евкліда), з іншого, він же контролює завершення циклу **while**. Цикл завершується, якщо залишок від ділення дорівнює нулю (нагадаємо, що число нуль діє так само, як і логічне значення **false**).

Ще один різновид оператора циклу (цикл **do-while**) має такий формат:

```
do оператор;
while (ЛВ);
```

Оператор, що становить тіло циклу, виконується ще раз,

якщо значення ЛВ є **true**. Різниця між циклами **while** і **do-while** полягає у наступному. Якщо з самого початку ЛВ = **false**, то тіло циклу while не виконується ні разу (цикл з передумовою). Тіло циклу **do-while** у цьому випадку буде виконано 1 раз (цикл з постумовою).

Нижче наведено фрагмент програми, у якому для знаходження найбільшого загального дільника двох цілих чисел використовується оператор **do-while**.

```
int  a=125,  b=2500, r=a%b;
do { a=b;  b=r; }
while  (r = a%b);
printf("НЗД = %d \n", b);
```

Наступний приклад використання циклу **do-while** являє собою програму, що повертає код натиснутої клавіші. При натисканні клавіші з розширеним (двобайтовим) кодом програма виводить на екран другий, не нульовий байт. Завершення роботи програми відбувається при натисканні клавіші Esc. У програмі використовується функція отримання символу натиснутої клавіші **getkey** з модулю **syst.h**.

```
// Приклад 4
#include <syst.h>
  void main()
{ char s;
  do { s=getkey();
       printf("s= %c  code=%d \n", s,s);
     }
  while (s!=27);
}
```

Третім різновидом оператора циклу є оператор **for**, він має такий формат:

for (*ініц*; ЛВ; *модиф*) *оператор*;

У круглих дужках записуються три групи операторів, групи розділяються символом **";"** . Перша група, що позначена як ***ініц,*** використовується для оголошення й ініціалізації (або тільки для ініціалізації) змінних циклу. Можна оголосити і ініціалізувати одразу декілька змінних циклу. Друга група операторів, що позначена як ЛВ, використовується для контролю завершення роботи циклу: якщо значення ЛВ дорівнює **true**, виконання тіла циклу повторюється. Третя група операторів - ***модиф*** використовується для модифікації значень змінних циклу. Можна записувати модифікацію одразу декількох змінних циклу.

Кожний з елементів (або всі відразу) у круглих дужках може бути відсутнім, але символи-роздільники **";"** зберігаються. Наприклад, можна застосовувати такий оператор

for (;;) *оператор*;

Він відповідає безкінечному циклу. У такому випадку ініціалізація змінних циклу та їх модифікація, а також перевірка умови завершення циклу повинні бути передбачені відповідними операторами у тілі циклу.

Оператор

for (int k=0;;) *оператор*;

відповідає нескінченному циклу з оголошенням та ініціалізацією змінної k.

В межах однієї групи у дужках можна записати декілька виразів, розділяються вони комами:

for (i=0,j=N;;i++,j--) *оператор*;

Такий оператор відповідає циклу з ініціалізацією змінних та їх модифікацією, але без перевірки умови.

Оператор

break;

застосовується як допоміжний оператор. Він здійснює достроковий вихід з тіла оператора **do**, **for**, **while**, **switch**. Часто цей оператор застосовується для виходу з циклу у тому випадку, коли перевірка умови завершення циклу у його заголовку не передбачена. У наступному прикладі програми вихід із циклу відбувається, коли значення виразу $\mathbf{i^3}$ перевершує 10000.

```
// Приклад 5
#include <syst.h>
  void main()
{ int i;
  for (i=0;;i++)
    { printf ("%d^3 = %d \n", i, i*i*i);
      if (i*i*i>10000) break;
    }
 }
```

Оператор

continue;

здійснює достроковий перехід на наступну ітерацію циклу, він також є допоміжним і використовується в складі операторів **do**, **for**, **while**. Цей оператор дозволяє не виконувати частину операторів тіла циклу, що залишилася, а відразу перейти до наступної ітерації. Наприклад, для виведення на екран всіх чисел кратних 11, можна скористатися такою програмою:

```
// Приклад 6
   #include <syst.h>
  void main()
```

```
{ int i;
  for (i=0; i<500; i++)
   {  if (i%11) continue;
      printf ("%d ", i);
   }
 }
```

Для того, щоб запрограмувати достроковий вихід з функції застосовується такий оператор:

return *вираз*;

де ***вираз*** визначає значення, що повертається функцією. Функція, яка повертає значення, повинна вміщувати принаймні один такий оператор.

Для дострокового виходу із процедури (функції, яка не повертає значення) вираз після слова return не записується:

return;

При відсутності оператора return у процедурі, вихід з неї здійснюється природним шляхом після завершення останнього оператору тіла процедури.

2.5.3. Оголошення змінних та ініціалізація

Оголошення змінної може вміщувати таку інформацію: ім'я змінної, її тип, значення або вираз для її ініціалізації. Загальний формат оголошення змінної має такий вигляд:

***Тип змінна* [= *початкове значення*];**

В одному оголошенні можна записувати декілька змінних. Приклад:

double a, b, x=2.345;

Значення для ініціалізації можна також записувати в круглих дужках безпосередньо після імені змінної:

int number(125);

Для ініціалізації повинні використовуватись вирази відповідного типу. Вирази, що записуються для ініціалізації, обмежуються тими, що можуть бути обчисленими при виконанні оголошення. Приклад правильно записаних оголошень:

```
int a=3, b=4;
int c=sqrt(a*a+b*b);
```

Оголошення змінної може бути записано у будь-якій точці програми.

При створенні об'єктів, їхньої ініціалізації та знищенні використовуються спеціальні функції - конструктори та деструктори. У випадку вбудованих типів конструктори й деструктори створюються і діють автоматично, явно вони не визначаються. Наприклад, у такому рядку програми:

```
int number(125);
```

викликається конструктор для створення об'єкта типу int з ім’ям number та ініціалізації його значенням 125.

Якщо оголошення починається з службового слова extern, воно називається попереднім. Приклад попереднього оголошення змінної alfa:

```
extern double alfa;
```

Попереднє оголошення не створює об'єкт (не виділяє для нього місце у пам’яті), воно лише робить ім'я об’єкту (у нашому прикладі **alfa**) відомим компіляторові. Попередні оголошення використовуються в багатомодульних (багатофайлових) програмах для розширення сфери дії імен на декілька об’єктних файлів програми.

2.5.4. Константи і константні об'єкти

Константний об'єкт відрізняється тим, що після його початкової ініціалізації значення об’єкту не може бути змінено.

Константи діляться на іменовані й неіменовані (прості).

Прості константи можуть мати один з наступних типів: цілий, дійсний, символьний, рядковий, логічний, тип enum. Тип константи й відповідний діапазон можливих значень визначається компілятором по її зовнішніх ознаках.

Цілі константи можуть мати формат десятковий, восьмеричний, шістнадцятирічний. Запис восьмеричної константи починається із символу 0 (нуль), шістнадцятирічної - із символів 0x. При записі цілих констант можуть використатися суфікси:

L або l - для довгих констант (4 байти),

U або u - для беззнакових констант.

Шістнадцятирічна й восьмеричні константи можуть бути тільки беззнаковими. Цифри шістнадцятирічної константи позначаються символами: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Дійсна константа повинна мати крапку, або символ **e**, або обидва символи. При відсутності суфікса дійсна константа має тип **double**. Суфікс **F** або **f** вказує на тип **float**, а суфікс **L** або **l** на тип **long double**.

Приклад запису деяких числових констант:

2005 - ціла;
2500000000L - довга ціла;
4000000000UL - беззнакова довга ціла;
1024U - беззнакова ціла;
1.609e-19F - дійсна 4-х байтова;
3.141592659 - дійсна 8-ми байтова (подвоєної точності);
0x2F3A28F0 - шістнадцятирічна.

Символьна константа являє собою один або два символи,

взяті в апострофи, наприклад: 'A', 'B' , '$', 'ad' . Для односимвольної константи виділяється один байт, для двосимвольної - два байти. Символ можна вказати восьмеричним або шістнадцятирічним числовим кодом: наприклад, символ '\0xA1' являє собою російську літеру "б".

Спеціальні символи й керуючі байти записуються за допомогою символу "\". Приклад спеціальних символів, які використовуються найбільш часто:

\\ - символ "\";
\' - апостроф;
\" - лапки;
\n - перехід у наступний рядок;
\r - повернення в початок поточного рядка;
\b - повернення на одну позицію назад;
\t - табуляція;
\a - звуковий сигнал;
\0 - нульовий символ (символ із числовим кодом 0).

При застосуванні спеціальних символів, варто розташовувати їх в середині одинарних лапок, якщо ви використовуєте дані символи самі по собі, наприклад '\n', або усередині подвійних лапок, якщо спеціальний символ треба записати в середині рядка. Наприклад: "Добрий день\nСвіт!\n". Для того, щоб наочно побачити, як працюють спеціальні символи, запустіть на виконання наступну програму. В програмі застосовуються спеціальні символи '\a', '\t', '\n' .

```
// Приклад 1
#include <syst.h>
  void main()
{ cout << "Signal\a\tSignal\a\tSignal\a\n";
}
```

Рядковою константою (рядком символів) є послідовність символів у подвійних лапках, наприклад: "IBM PC XT/AT" . У рядкову константу автоматично заноситься завершальний символ '\0' (нуль-байт). Такі рядки називають asciz-рядками.

Логічними константами є слова false та true. Зверніть увагу на те, що ці слова у лапки не беруться:

bool L = false;

Константи типу enum являють собою слова - значення типу enum. Кожне таке слово має цілочисловий еквівалент.

Константні вирази.

Всюди, де використовуються константи, у якості константи можна записати константний вираз. Елементами константного виразу є константи або константні вирази, символи припустимих операцій, функції, а також дужки. Функцію записувати можна за умови, що аргументами її є константи або константні вирази.

Іменовані константні об'єкти оголошуються із застосуванням службового слова const :

const double pi = 3.141592;

При використанні константних об'єктів варто керуватися наступними правилами:

а) оголошення будь-якого константного об'єкта повинне містити ініціалізацію;

б) ніякий оператор не може змінювати значення оголошеного константного об'єкта;

в) константним може бути оголошений об'єкт будь-якого типу.

2.6 Типи мови C++

Типи та їхні різновиди

Службове слово void

Тип-перерахування enum
Перетворення типів

2.6.1Типи та їхні різновиди

Будь-яку програму можна уявляти собі як механізм обробки даних. Інформація, що оброблюється, завжди структурована (явно або не явно, хочемо ми цього, чи ні). Спосіб і засоби структурування даних, що застосовуються, у значній мірі визначають рівень і якість програмної розробки, а часто і її долю в майбутньому. Основним засобом структурування даних є типи. Всі типи, що використовуються в C++, діляться на 3 види:

- вбудовані (базові типи),
- похідні (покажчики й посилання),
- нестандартні.

Короткий опис вбудованих типів наводиться нижче. Параметри вбудованих типів наводяться в таблицях.

Цілі типи

bool - логічний тип. Розмір - 1 байт, значення - true, false. Відноситься до групи цілих типів, значенню true відповідає ціле число 0, значенню false - будь-яке ціле, що не дорівнює нулю.

char - однобайтовий цілий тип, може використовуватися для зберігання як цілих чисел, так і символів. Має дві модифікації: **signed char** та **unsigned char**.

short - двобайтовий цілий тип. Має модифікації: **signed short** та **unsigned short**.

int - чотирьохбайтовий цілий тип. Має модифікації: **signed int** та **unsigned int**.

long - чотири або восьмибайтовий цілий тип. Повне ім'я цього типу **long int**, однак **int** можна опускати. Має модифікації:

signed long [**int**] та **unsigned long** [**int**].

Кожен з цілих типів, крім bool (char, short, int, long) має дві модифікації: signed і unsigned. По умовчанню звичайно працює модифікація signed.

Таблиця 2.3.Характеристики цілих числових типів (платформа Win32)

Т и п	Розмір	Діапазон значень
Signed char	1	$-2^7 .. 2^7-1$
Unsigned char	1	$0 .. 2^8-1$
Signed short	2	$-2^{15} .. 2^{15}-1$
Unsigned short	2	$0 .. 2^{16}-1$
Signed int	4	$-2^{31} .. 2^{31}-1$
Unsigned int	4	$0 .. 2^{32}-1$
Signed long	4	$-2^{31} .. 2^{31}-1$
Unsigned long	4	$0 .. 2^{32}-1$

Для платформ, орієнтованих на 64-бітові процесори, типи **signed long** і **unsigned long** мають такі характеристики:

Таблиця 2.4. Характеристики цілих числових типів (платформа Win64)

Т и п	Розмір	Діапазон значень
Signed long	8	$-2^{63} .. 2^{63}-1$
Unsigned long	8	$0 .. 2^{64}-1$

До дійсних типів відносяться типи **float, double, long double**. Значення дійсних типів, на відміну від цілих, подаються наближено. Точність подання значення залежить від кількості розрядів мантиси числа, інтервал значень - від кількості розрядів, що відведені для збереження порядку. Основні

характеристики дійсних типів наведені в таблиці нижче.

Таблиця 2.5.Характеристики дійсних типів

Т и п	Розмір	Діапазон значень	Точність	Довж. мантиси
float	4	$3\cdot10^{-38}$ .. $3\cdot10^{38}$	$3.0\cdot10^{-08}$	24
double	8	$2\cdot10^{-308}$ .. $2\cdot10^{308}$	$5.6\cdot10^{-17}$	53
long double	10	$3\cdot10^{-4932}$ .. $1\cdot10^{4932}$	$2.7\cdot10^{-20}$	64

Похідні й нестандартні типи

До похідних типів відносяться покажчики та посилання. Використання цих типів буде розглянуто в наступних розділах книги. Нестандартні типи створюються шляхом запису відповідних визначень. Вони належать до однієї з наступних категорій:

- enum (перелічення);
- struct (структура);
- class (клас);
- union (об'єднання).

Останні три категорії належать до класових типів. Вони є основою для реалізації принципів об’єктно-орієнтованого програмування.

2.6.2. Службове слово void

Службове слово **void** застосовується при запису оголошень деяких об'єктів у програмі. І хоча слово **void** синтаксично застосовується як тип об'єкта, насправді типом даних не є. Випадки, коли необхідно використовувати **void** будуть

розглянуті у наступних розділах, присвячених покажчикам та функціям.

2.6.3. Тип-перелічення enum

Тип-перелічення enum створюється за допомогою службового слова **enum**. Такий тип відповідає деякій заданій множині слів-ідентифікаторів. Приклад оголошення типу-перелічення для якого обрано ім'я DAY:

enum DAY { mon, tus, wed, th, fri, sat, sun };

Приклад оголошення та ініціалізації змінних типу DAY:

DAY x=mon, y=tus;

Змінні x,y можна оголосити й безпосередньо, не використовуючи ім'я типу:

enum { mon, tus, wed, th, fri, sat, sun } x=mon, y=tus;

Кожному enum-слову ставиться у відповідність ціле число - його порядковий номер. Компілятор інтерпретує значення типу **enum** як значення числового типу **unsigned char**. Для типу **enum** немає стандартних функцій введення/виведення, однак їх неважко написати самому.

Використовуючи тип **enum** легко створити, наприклад, свій логічний тип **BOOL**:

enum BOOL { false, true };

Цікавою можливістю використання типу enum є застосування enum-значень як іменованих констант:

enum { small=10, large=1000 };

Тепер **small** і **large** можна використовувати як константи, наприклад у оголошенні масивів:

int a[small], A[large];

2.6.4. Перетворення типів

При використанні простих типів операція перетворення типу може записуватися одним із двох еквівалентних способів: **(*тип*)x** або ***тип*(x)** , де **x** - вираз, а ***тип*** - ім'я типу, до якого виконується перетворення.

C++ допускає запис приведення до будь-якого типу. Відповідальність за правильне використання цієї операції покладена на програміста. Наведемо приклад, коли запис явного перетворення типу має сенс:

```
int a=2;
float x=3.5;
a = a + (int)x;
```

Останній оператор буде виконуватися швидше, у порівнянні з більш простим оператором:

```
a = a + x;
```

Це пов'язано з тим, що замість двох операцій перетворення типу (int→float та float→int) виконується тільки одне перетворення виду (float→ int).

Операцію приведення типу доводиться явно записувати, якщо потрібно одержати результат дійсного типу від ділення цілих значень. Наприклад, при виконанні таких рядків програми

```
int i=3, k=4;
float x;
x = i/k ;
```

отримаємо значення x = 0. Якщо ж, все-таки потрібно одержати точне значення x = 0.75, другий рядок необхідно записати так:

```
x = (float)i/(float)k;
```

або хоча б так:

```
x = i/(float)k;
```

2.7 Покажчики і посилання

Покажчики
Посилання

2.7.1. Покажчики

Покажчиком (pointer) називають змінну, яка приймає значення, що є адресою деякого об'єкта. Покажчики можуть бути типізованими й не типізованими. Найчастіше зустрічаються типізовані покажчики. Приклад оголошення типізованого покажчика для роботи з об'єктами типу float:

```
float* pf;
```

Еквівалентний запис цього оголошення виглядає так:

```
float *pf;
```

Останній варіант зручніше, якщо оголошується одразу декілька покажчиків:

```
float  *p, *q, *h;
```

Покажчику **pf** можна привласнити адресу будь-якого об'єкта, що має тип **float**. Відзначимо, що типізованим є не тільки сам покажчик, але і його значення - адреса.

Унарна операція * , що діє на покажчик, повертає посилання на об'єкт (значення Lvalue), адреса якого відповідає значенню покажчика. У наступному прикладі

```
float  a = 5;
float* p = &a;
*p = *p/2;
```

Тут ***p** є посилання на об’єкт, яким є змінна **a**, а останній оператор діє так само, як оператор

```
a = a/2;
```

Значення 0 або NULL для покажчика є спеціальною ознакою, воно означає, що покажчик не вказує на жоден об'єкт. При створенні покажчика при умові, що ініціалізація його буде виконуватись десь далі, настійно рекомендується ініціалізувати його значенням NULL. Це зменшує ризик виникнення помилок у програмі.

Арифметика покажчиків

Арифметичні операції над покажчиками (адресами) виконуються за особливими правилами - правилам адресної арифметики. Нехай **p** - типізований покажчик, **k** - змінна цілого типу. Тоді значенням виразу

p+k

є значення адреси, збільшеної на розмір **k** об'єктів того типу, який відповідає покажчику **p**. Тобто така операція повертає адресу, зсунуту на **k** об'єктів уперед відносно значення **p**. У наступному прикладі

```
double *p, *q = &x;
p= q+5;
```

значенням покажчика **p** буде значення покажчика **q**, збільшене на 5*8=40 байтів.

Операція виду

p+1

повертає адресу того об'єкта, що є наступним після об'єкта з адресою p.

Операція виду **p++** збільшує значення покажчика **p** на розмір одного об'єкта відповідного типу, тобто вона діє також, як оператор **p = p+1**.

Операції **-** (віднімання) і **--** (декрементування) діють аналогічним чином, але адреса змінюється у бік зменшення.

Визначеною є також операція віднімання для покажчиків

або адрес. Нехай p1 та p2 є покажчики на об’єкт деякого типу. Тоді результатом виконання операції

p2 - p1

буде відстань від об’єкту з адресою p1 до об’єкту з адресою p2 у просторі пам’яті, яка повертається як кількість об’єктів. Наприклад, при виконанні такого програмного коду:

```
 void main()
{ double x[100];
 double *p1= &x[10], *p2= &x[20];
 printf("p2-p1 = %d \n",p2-p1);
}
```

отримаємо на екрані наступне:

p2 - p1 = 10

При одержанні нових значень покажчиків за допомогою зазначених вище операцій, наявність у тій області пам'яті, що відповідає новому значенню адреси, об'єкта відповідного типу не перевіряється. Це є джерелом численних і досить неприємних помилок, вони виявляються тільки лише на етапі виконання програми. Локалізація таких помилок з цієї причини може бути досить непростою.

Застосовуються покажчики в більшості випадків для роботи з масивами, для програмування операцій з динамічними об'єктами, а також у якості параметрів функцій. Нижче наведено цікавий приклад використання покажчика на тип char для побудови функції **strcpy**, що здійснює копіювання символів з рядка **s2** у рядок **s1**:

```
 void strcpy(char* s1, char* s2)
{ while (*s1++ = *s2++);
}
```

Зверніть увагу на те, що рядок-джерело і рядок-призначення передаються у функцію за допомогою покажчиків. Це є основний спосіб передачи масиву у функцію.

Крім поняття покажчик на об’єкт, існує й таке поняття, як покажчик на функцію. Робота з такими покажчиками буде розглянута пізніше, у главі, присвяченій функціям.

Типовими помилками при використанні покажчиків є:

- запис операцій з невизначеними (не ініціалізованими) покажчиками;

- втрата адреси динамічного об'єкта при виході з блоку.

У прикладі, наведеному нижче, виконується операція з невизначеним покажчиком, що є грубою помилкою:

```
float* p;
*p = 12.5;
```

Після виконання першого рядка буде створений покажчик, який залишається не ініціалізованим. Це означає, що його значення буде випадковим. Виконання другого рядка приведе до запису числового значення типу double (8 байт) по випадковій адресі. У більшості випадків після запуску програми це приводить до одержання повідомлення про помилку періоду виконання, яке генерується операційною системою. Воно може виглядати так:

```
Thread stopped
C:\users\VP\lab_05.exe:
Fault: access violation at 0x402307
write of address 0x0
```

Ще один приклад з помилкою:

```
char s = "Borland Builder 6.0";
s = "Intel";
```

Після виконання другого рядка область пам'яті, у якій зберігається рядок "**Borland Builder 6.0**" буде загублена

назавжди, бо адреса цього рядка ніяким чином не зберігається.

Нагадаємо ще раз, що покажчик у більшості випадків є типізований. Тип значення покажчика при необхідності можна перетворити за допомогою унарної операції перетворення типу. Роль символу такої операції грає запис виду (***тип****) , де ***тип*** - тип, до якого приводиться значення покажчика. Використання цієї операції вимагає деякої обережності. У більшості випадків, така операція використовується для перетворення типізованої адреси у не типізовану або навпаки.

При оголошенні не типізованого покажчика використовується службове слово **void**. Наприклад:

```
void*  pv;
```

Не типізований покажчик не асоціюється з якимось певним типом. Значення такого покажчика є "чиста" адреса. Йому можна присвоїти адресу об'єкта будь-якого типу, причому присвоювання можна записувати без вказівки операції приведення типу, наприклад:

```
double x,y;
void* px = &x;
```

Однак при виконанні операції з певним об'єктом через не типізований покажчик операцію приведення типу необхідно записувати:

```
y = fabs(*pv);                    // помилка!
y = fabs(*(double*)pv);           // правильно!
```

2.7.2. Посилання

Посилання (reference), на відміну від покажчика, завжди є типізованим, бо воно посилається на певний об’єкт. Воно може

бути іменованим або неіменованим (не явним). Для створення іменованого посилання необхідно записати відповідне оголошення. Формат такого оголошення має вигляд:

***тип & ім'я = Lvalue*;**

Тут ***ім'я*** - іменоване посилання, що оголошується, **Lvalue** - вираз виду **Lvalue**. Приклад оголошення посилання **R** на змінну **A** типу **float**:

```
float  A;
float& R=A;
```

Іменоване посилання являє собою ім'я (можливо додаткове) деякого об'єкта. Не іменовані посилання застосовуються для тимчасового посилання на об’єкт. Приклад використання не іменованого посилання:

```
float* p = new float[100];
*(p+2) = a+b+c;
```

Ліворуч в останньому рядку записане неявне посилання на третій елемент динамічного масиву, адреса масиву надається покажчиком p .

При використанні посилань необхідно враховувати наступні правила.

- Оголошення змінної-посилання повинне включати її ініціалізацію, крім випадків, коли вона є параметром функції або описана як extern або є компонентом класу.
- Значення посилання після ініціалізації не може бути змінено.
- Посилання завжди має певний тип.
- Посилання на покажчик об’являти можна, але покажчик на посилання - ні. Наприклад, запис

```
double*& a = p;
```

є припустимим, якщо покажчик p має тип double*. А такий запис:

double&* x;

є помилковий.

- Не можна створювати масив посилань.
- Посилання не є об'єктом, воно не займає місце у пам'яті.

Основні області застосування посилань:

- параметри функцій;
- значення, що повертає функція;
- робота з динамічними об'єктами.

Використання посилання як параметру функції або значення, що повертається, буде розглянуто пізніше, у главі, присвяченій функціям.

2.8 Функції та процедури

Загальні відомості
Функція main
Функції зі змінною кількістю параметрів
Покажчики на функції
Функції з шаблонами

2.8.1. Загальні відомості

Функція у термінах сучасних мов програмування є те, що у ту епоху, коли електронні обчислювальні машини вже були, але ніяких мов програмування ще не було, називалося підпрограмою. Розумілося це як деякий програмний код (частина програми), записаний таким чином, щоб було можливо програмувати звернення до нього з будь-якої точки програми. У формі підпрограм реалізовувались типові елементи програм, які мали "багаторазове" призначення. Широкого розповсюдження

набула практика створення бібліотек стандартних підпрограм, їх застосування суттєво скорочувало загальний розмір програм, а сам процес програмування робило більш надійним.

У сучасних мовах програмування такий засіб набув подальшого розвитку. Значно змінилася як практика створення і використання підпрограм-функцій, так і засоби обміну інформацією між ними та навколишнім програмним середовищем.

У мові C/C++ функції є основними елементами тієї частини програми, що представляє алгоритми розв'язання проблеми, покладеної на неї. Образно кажучи, функції являють собою головні будівельні блоки, з яких будується виконуваний код будь-якої програми. Будь-яка програма містить у собі не менш однієї функції. Серед всіх функцій кожної програми особливе місце займає функція з фіксованим ім'ям **main** (головна функція). З виконання цієї функції починається робота будь-якої програми і після завершення роботи головної функції завершується й робота програми в цілому. На відміну від інших, функція **main** має свої особливості побудови й роботи, які будуть розглядатися трохи нижче.

Запис визначення функції складається з двох основних елементів: заголовка функції та її тіла. Заголовок представляє інтерфейсну частину функції, крім імені функції він містить інформацію про кількість параметрів і тип кожного з них. Крім того заголовок функції містить запис про тип значення, що вона повертає. Якщо нам відоме призначення функції, то знання її заголовка досить для того, щоб її можна було використовувати для програмування.

Тіло функції заключають у фігурні дужки і, таким чином воно є блоком. Звідси виходить, що тіло функції утворює свій простір імен. Всі оголошення, записані в середині функції, так

само як і оголошення її параметрів у заголовку, є локальними. У більшості випадків оголошення функції має такий формат:

***тип ім'я*(*список параметрів*)**
{ *тіло функції*
}

Після фігурної дужки, що завершує запис тіла функції, крапка з комою не ставиться.

У зв'язку з тим, що функція є досить автономний компонент програми, важливе значення мають наявні механізми обміну інформацією між функцією та зовнішнім кодом програми. Мовою C/C++ передбачено такі засоби обміну інформацією:

- глобальні об'єкти;
- вхідні та вихідні параметри функції;
- значення, що повертається.

Глобальні об'єкти

Глобальні об'єкти є найпростіший спосіб налагодження передавання вхідної та вихідної інформації. Але при застосування глобальних об'єктів треба враховувати такі обмеження:

а) доступ із функції до глобального об'єкту неможливий, якщо його ім'я екранується локальними оголошеннями функції;

б) якщо глобальний об'єкт є константним, його не можна застосувати для передавання вихідної інформації;

в) глобальні об'єкти рекомендується застосовувати тільки для передавання такої інформації, яка має дійсно глобальний характер (наприклад, константи або інформація, яка має загальне значення для програми в цілому). Для передавання поточної інформації краще застосовувати параметри функції.

У наступному прикладі програма розраховує і виводить на екран таблицю функції
y = x lg(x) з кроком табуляції 0.5. Функція func(), яка виконує обчислення, не має параметрів та повертаємого значення, обмін інформацією з зовнішнім кодом виконується через глобальні змінні X,Y.

```
// Приклад 1
#include <syst.h>
double X,Y;                    // X,Y - глобальні змінні
  void func()                  // функція працює з
                               // глобальними об’єктами
{ Y= X*lg(X);
}
  void main()
{ for (int k=2;k<=20;k++)
   { X= 0.5*k;
     func();
     printf("%7.4f  %7.4f \n",X,Y);
   }
}
```

Параметри функції

Параметри функції є локальними об'єктами або локальними посиланнями, які діють у межах тіла функції. Можна застосовувати 3 види параметрів: параметр-змінна, параметр-покажчик та параметр-посилання. Наприклад, у наступній функції застосовано всі 3 види параметрів:

```
  int F(int k, int* p, int& x)
{ .....................
}
```

У тому випадку, коли параметром функції є змінна або

покажчик, в момент її виклику відбувається копіювання значення фактичного параметра у відповідний локальний об'єкт, що відповідає формальному параметру функції. Якщо параметром функції є посилання, тоді на час виконання функції воно зв'язується з тим ім’ям, якого зазначено як фактичний параметр у виклику функції. Це дає змогу створювати функції, які спроможні отримувати вхідну інформацію від діючої програми і передавати вихідну інформацію у програму.

Для отримання вхідної інформації можна застосовувати параметр функції будь-якого з трьох вказаних вище видів: параметр-змінну, параметр-покажчик, параметр-посилання. Параметри останніх двох видів варто застосовувати у тому випадку, коли треба передати об’єкт або масив великого розміру. Використання параметра-покажчика або параметра-посилання призводить до того, що фактично копіюється лише адреса об'єкта (4 байти), а не сам об'єкт. Таким чином при передаванні об’єкту великого розміру істотно розвантажується стек і скорочується час, що затрачається на виклик функції.

Передача інформації із функції до програми, яка її викликає, реалізується шляхом запису деякого значення у зовнішній (по відношенню до функції) об’єкт. Для цього можна застосовувати параметри-покажчики або параметри-посилання.

У наведеному нижче прикладі функція **c_mul** виконує операцію множення двох комплексних чисел: $c = a \cdot b$, де $a = a\_re + a\_im \cdot \mathbf{i}$, $b = b\_re + b\_im \cdot \mathbf{i}$, $c = c\_re + c\_im \cdot \mathbf{i}$, де **i** - уявна одиниця.

```
void c_mul(double a_re, double a_im,
           double b_re, double b_im,
           double& c_re, double& c_im)
{ c_re=a_re*b_re-a_im*b_im;
```

```
  c_im=a_im*b_re+a_re*b_im;
}
```

Зверніть увагу на те, що розраховані значення дійсної та уявної частин комплексної величини c (c_re та c_im) передаються за допомогою параметрів-посилань, вони мають тип double&.

Таким чином, якщо ми створюємо функцію, що повинна змінювати стан деякого зовнішнього об'єкта, ми можемо обрати один із двох варіантів: використати параметр-покажчик або параметр-посилання. Різниця, яка виникає при їх застосуванні ілюструється у наступному прикладі процедурами swap1 і swap2. Призначення обох функцій однаково: обмін значеннями двох зовнішніх об'єктів типу long:

```
  void swap1(long *px, long *py)
{ long R=*px; *px=*py; *py=R;
}
```

Приклад застосування функції swap1:

```
  long a=3, b=4;
  swap1(&a,&b);

  void swap2(long &x, long &y)
{ long R=x; x=y; y=R;
}
```

Застосування функції swap2:

```
  long a=3, b=4;
  swap2(a,b);
```

У першому варіанті застосовано параметри-покажчики, у

другому - параметри-посилання. Зверніть увагу на те, що виклики функцій swap1 і swap2 записується по різному.

Якщо параметром функції є масив, він ніколи не передається по значенню, замість цього передається адреса першого елементу масиву.

Простори імен

Ім'я функції, що не є членом класу, належить глобальному простору імен. Ім'я функції - члена класу належить простору свого класу. Всі імена, що оголошуються всередині функції, належать простору цієї функції. Цьому ж простору належать і імена параметрів, які оголошуються у заголовку функції.

Значення, що повертаються

Якщо тип, що повертається, позначено словом void, то така функція не повертає будь-якого значення. Такі функції називають процедурами. Наведена вище функція swap є такою процедурою. Взагалі функція може повертати: значення будь-якого раніше визначеного типу, або адресу об'єкта, або посилання на зовнішній об'єкт.

Дещо незвичним є випадок, коли функція повертає посилання. У прикладі, наведеному нижче, визначається функція F(i,k), що дозволяє звертатися до одновимірного глобального масиву R[400] як до двовимірної матриці розміром 20x20:

```
// Приклад 2
int R[400];  int& F(int i, int j){ return R[20*i+j];
}
  void main(){ F(10,15) = 7777; // записати елемент матриці
  printf("%i \n", F(10,15));   // прочитати елемент матриці і
}                               // вивести його на екран
```

Ще одним прикладом функції, що повертає посилання, є функція cell такого типу:

char& cell(int x, int y).

Припустимо, що ця функція повертає посилання на той байт відеопам'яті, якому відповідає позиція на текстовому екрані з координатами x,y. Приклади використання такої функції: оператор

cell(45,15) = 'A';

записує в 45 позицію 15 рядка символ 'A' , а оператор

cell(1,20) = cell(50,10);

копіює символ з однієї позиції екрана в іншу.

Прототипи

Прототип є попереднім оголошенням функції. Прототип записується як заголовок функції, що завершується крапкою з комою. Він містить інформацію про тип функції, типах її параметрів та їх кількості. Прототип використовується компілятором для контролю правильності виклику функції, а також для автоматичного приведення аргументів до потрібного типу. Приклад запису прототипу функції:

int fun(int, float*, double&);

Імена параметрів функції вказувати у прототипі не обов’язково. Прототипи є єдиним засобом розв'язування колізій, пов'язаних із записом програм, які містять непрямі рекурсивні виклики функцій.

У прототипі функції можна записувати ініціалізацію аргументів. Можна застосовувати прототипи, які відповідають функціям зі змінною кількістю параметрів.

Статичні змінні

Тіло функції може містити змінні, оголошені із застосуванням службового слова **static**. Такі змінні відрізняються тим, що їх значення (також як і вони самі) зберігаються до наступного виклику функції. Об'єкти, що відповідають змінним типу static, знищуються при завершенні всієї програми, а не при завершенні роботи функції. Статичні змінні фактично є глобальними об'єктами, які відрізняються тим, що їх ім'я інкапсульовано у простір функції.

Нижче наводиться приклад функції, що підраховує кількість своїх викликів і, якщо ця кількість перевищує задану величину, функція видає на екран повідомлення й завершує роботу всієї програми.

```
// Приклад 3
#include <syst>
  void fun(int N)
{ static int num;
  num++;
  if (num > N)
    { printf("К-сть викликів перевищує припустиме
значення");
      exit(0);
    }
}
  void main()
{ for (;;) fun(1000);
}
```

Параметри функції з ініціалізацією

Заголовок функції може містити ініціалізацію деякої частини своїх параметрів. Ініціалізація виконується, якщо при

виклику функції відповідні фактичні параметри не зазначені. Приклад запису такої функції:

```
long F(int a, long x=0, long y=1)
{ .....................................
}
```

Параметри, що ініціалізуються, повинні бути останніми в списку. Вони можуть не задаватися при виклику функції. Всі наступні виклики наведеної вище функції F є правильними: F(n,a,b) , F(n,a) , F(n) .

Модифікатори

У заголовку функції можна записувати спеціальні службові слова-модифікатори **inline**, **extern**, **static**, **volatile**.

Модифікатор **inline** вказує на те, що замість побудови звичайного виклику функції компілятор повинен просто підставляти код тіла функції в кожному місці її виклику. При такому способі використання функції, програма буде виконуватися швидше, однак при цьому розмір файлу, що виконується, може суттєво збільшуватися.

Модифікатор **extern** використовується в багатофайлових програмах. Він вказує на те, що записаний далі заголовок функції являє собою попереднє оголошення функції. При цьому повне визначення функції може перебувати в іншому об’єктному файлі програми. Фактично це означає розширення простору, у якому діє ім'я функції на поточний об’єктний файл програми. Звичайно ім’я функції діє у просторі одного (поточного) об’єктного файлу програми.

Нижче наведено ще кілька корисних прикладів.

Раніше, як ілюстрація використання операторів циклу, наводилася програма визначення найбільшого загального дільника двох заданих цілих чисел. Нижче ці обчислення

реалізовані у вигляді функції **nod**.

```
// Приклад 4
  int nod(int a, int b)
{ int r; while (r=a%b) { a=b; b=r; }
  return b;
}
```

Наведемо цікавий приклад побудови функції парного обміну swap для змінних типу int. Завдяки використанню побітової операції "^" (не еквівалентність), необхідність у застосуванні допоміжної (буферної) змінної відпадає. Для передачі параметрів використовуються посилання.

```
// Приклад 5
  void swap(int& a, int& b)
{ a^=b; b^=a; a^=b;
}
```

Функція binar з наступного прикладу формує символьний рядок, який є текстовим записом бінарного коду значення-аргументу і повертає адресу символьного рядка. Аргументом функції є 4-х байтове ціле значення типу long. Провідні (ті, що стоять попереду) нулі бінарного коду у символьному рядку не відображуються. Зверніть увагу на використання в тілі функції статичного символьного масиву B.

```
// Приклад 6
  char* binar(ulong x)
{ static char B[33];
  char i,s, j=0;
```

```
  for (i=0;i<32;i++) { s= x & 0x80000000 ? '1' : '0' ;
                       B[j++]=s;
                        x<<=1;
                      }
  if (j>0) B[j]=0;
  else { B[0]='0'; B[1]=0; }
  return B;
}
```

2.8.2. Функція main

У порівнянні зі звичайними функціями, функція **main** має кілька відмінностей.

а) Ми не можемо обирати ім'я цієї функції вільно, воно визначено як системне ім’я. У деяких системах програмування (або при створенні програми під інші платформи) ім'я головної функції може відрізнятися від **main**.

б) При запуску будь-якої програми її робота починається з виконання функції **main**, а після завершення роботи функції **main** завершується і програма в цілому. Будь-яка інша функція, що задіяна у програмі, повинна прямо або побічно (через якусь іншу функцію) викликатися із функції **main**. Виключенням з цього є тільки спеціальні функції класів - конструктори і деструктори при умові, якщо вони викликаються для глобальних об’єктів.

в) Параметри функції **main** та їх застосування також визначено наперед. Вона може мати 3, 2 або 1 параметр або не мати параметрів зовсім. Найчастіше функція main застосовується без параметрів. Через параметри функції main передається інформація про параметри командного рядка, за

допомогою якого запускається програма, а також значення параметрів оточення процесора.

г) Тип значення, що повертається функцією **main**, може бути int або void. Якщо зазначено тип повернення int, останнім виконуваним оператором функції **main** повинен бути оператор

return ex;

де ex - вираз цілого типу. При цьому передбачається, що ціле число, яке повертається, буде якимось чином оброблено операційною системою. Обробка цього значення може бути передбачена в тексті керуючого bat-файлу.

Параметри функції main

Прототип функції main з повним списком параметрів виглядає так:

void або int main(int N, char p, char** q);**

Значення всіх параметрів формується і передається операційною систе-мою при запуску програми.

Перший параметр - N одержує значення, що дорівнює кількості параметрів командного рядка. Наприклад, при запуску програми **proba.exe** з такого командного рядка

proba.exe file1.txt file2.txt

параметр N одержить значення 3.

Другий параметр - p є покажчиком на одновимірний масив рядків, які містять самі параметри командного рядка. Параметри командного рядка розділяються пробілами.

Третій параметр - q передає адресу масиву рядків, які містять параметри оточення процесора. Зазначений масив завершується адресою NULL, це значення використовується при побудові циклічного перебору параметрів оточення процесора.

У наведеному нижче прикладі програма виводить на екран параметри командного рядка і параметри оточення процесора.

Зверніть увагу на особливості побудови другого циклу, що здійснює виведення параметрів оточення процесора.

```
// Приклад 1
#include <syst.h>;
  void main(int N, char** p, char** q)
{ int i;
  printf("Кількість параметрів командного рядка N =
%d \n",N);
  puts("Параметри командного рядка:");
  for (i=0;i<N;i++) printf("%s \n",p[i]);
  puts("Параметри оточення процесора:");
  for (i=0;q[i]!=0;i++) printf("%s \n",q[i]);
}
```

2.8.3. Функції зі змінною кількістю параметрів

Можна оголосити функцію зі змінною кількістю параметрів. Список параметрів такої функції розділяється на дві частини: фіксована частина списку і змінна частина. У заголовку функції змінна частина списку вказується трьома крапками, наприклад:

```
  int F(int a, ...)
{ тіло функції
}
```

При побудові такої функції для обробки списку параметрів застосовуються засоби модуля <stdarg.h>. Фіксована частина списку параметрів не повинна бути пустою, змінна частина списку параметрів у заголовку функції позначається трьома

крапками. Всі фактичні параметри змінної частини списку повинні мати однаковий тип. Розглянемо як це робиться на прикладі, у якому функція max(n,...) повертає значення найбільшого аргументу, причому перший параметр вказує кількість фактичних параметрів змінної частини списку, а самі параметри змінної частини списку мають тип int.

```
// Приклад 1
#include <syst.h>
#include <stdarg.h>
  int max(int N, ...)      // три крапки - змінна частина
                           // списку параметрів
{ va_list p;               // створено спеціальний покажчик p
  va_start(p,N);           // ініціалізація покажчика p
  int i,x, M=va_arg(p,int);  // M дорівнює значенню першого
                           // параметра
  for (i=1;i<N;i++)        // змінної частини списку параметрів
    { x=va_arg(p,int);     // x дорівнює значенню наступного
                           // параметра
      if (M<x) M=x;
    }
  return M;
}
  void main()
{ printf("max = %d \n", max(5,35,-35,40,17,-12));
}
```

У цьому прикладі застосовується спеціальний тип va_list і функції va_start та va_arg з модулю <stdarg.h>. Тип va_list визначає спеціальний покажчик, який вказує на один з аргументів змінної частини списку. Виклик функції va_start(p,N) ініціалізує покажчик p адресою першого параметра,

розташованого за останнім параметром фіксованої частини списку параметрів з ім'ям N. При виконанні виклику функції va_arg(p,int) виконуються дві дії:

а) формується значення, що повертається, воно відповідає тому параметру змінної частини списку параметрів, на який вказує покажчик p;

б) значення покажчика p переноситься на наступний параметр змінної частини списку параметрів, у нашому прикладі він має тип int.

2.8.4. Покажчики на функції

Оголошення покажчика на функцію має такий вигляд:

***тип* (*pf)(*параметри*);**

Тепер pf - покажчик на функцію зазначеного типу. Тип функції визначає кількість і типи її параметрів, а також тип значення, що повертається функцією.

Ім'я будь-якої функції F є покажчиком-константою на функцію відповідного типу (згадайте, схожа ситуація має місце у випадку масивів: ім'я не динамічного масиву є покажчиком-константою на масив).

Якщо ми маємо оголошений покажчик на функцію pf, йому можна привласнити адресу функції відповідного типу:

pf = F;

Після такого присвоювання виклик функції F можна записати так:

pf(параметри),

або так:

(*pf)(параметри).

Обидва виклики будуть еквівалентні звичайному виклику

функції F:

F(параметри).

Оголошення покажчиків на функції виглядають громіздко і вони не завжди зручні, особливо у заголовках функцій. Якщо це так, то записи оголошень можна спростити й зробити їх більше зрозумілими, якщо попередньо використати оголошення синоніму типу typedef :

```
typedef float (*FUN)(float,float);
FUN pf;
```

Тут у першому рядку визначається синонім типу FUN, що відповідає покажчику на функцію із двома аргументами типу float та значенням типу float, що повертається. У другому рядку оголошується покажчик pf на функцію зазначеного типу.

Особливо зручним такий прийом виявляється в тих випадках, коли потрібно оголосити масив покажчиків на функцію:

```
FUN pf[10];
```

а також при запису параметрів функції типу покажчик на функцію.

Нижче наведено приклад використання масиву покажчиків на функцію. У цьому прикладі запис pf[i](a,b) означає виклик однієї з трьох функцій залежно від значення індексу i .

```
// Приклад 1
#include <syst.h>
 int (*pf[3])(float x, float y);
                    // pf - ім'я масиву покажчиків на функцію
 int f0(float x, float y)
                    // об'ява функції відповідного типу
{ return x+y;
}
```

```
 int f1(float x, float y)
{ return x-y;
}
 int f2(float x, float y)
{ return x*y;
}

 void main()
{ pf[0]=f0; pf[1]=f1; pf[2]=f2;
                  // ініціалізація масиву покажчиків
 for (int i=0;i<3;i++)
   printf("f[i](x)=%d \n",pf[i](2.8,2.8));      // виклик
}                                                // функцій
```

Гарним прикладом використання покажчика на функцію є побудова функції для обчислення інтегралу по формулі Симпсона, при умові, що підінтегральну функцію потрібно передавати як параметр.

```
// Приклад 2
typedef double (*fun)(double);
  // оголошується синонім типу для покажчика на функцію
 double F(double x)      // підінтегральна функція F(x)
{ return ln(x);
}
 double G(double x)      // підінтегральна функція G(x)
{ return sqr(sin(x));
}
 double simp(fun f, double a, double b, int N)
                         // функція обчислення інтегралу
{ double x, h=(b-a)/N, S=0;
```

```
 for (int i=1;i<N;i+=2) { x=a+i*h;
                          S+= 4*f(x)+2*f(x+h);
                         }
 S+= f(a)-f(b);
 return S*h/3;
}
 void main()
{ printf("Integral for ln(x)    = %11.8f \n",
       simp(F,1,2,32));
 printf("Exact value          = %11.8f \n", 2*ln(2)-1);
 printf("Integral for sqr(sin))= %11.8f \n",
simp(G,0,pi,32));
 printf("Exact value          = %11.8f \n", pi/2);
}
```

2.8.5. Функції з шаблонами

При оголошенні функції з шаблонами використовуються параметризовані типи - шаблони. Застосування такого засобу дає змогу писати більш універсальні коди. Це робить програми більш лаконічними і прийнятними. Водночас це є засіб повторного використання коду.

Наприклад, якщо нам потрібна функція, яка повертає значення меншого із двох параметрів, то є сенс побудувати її з використанням шаблонів:

```
 template <class type>          // type - ім'я абстрактного типу
 type min(type A, type B)
{ return A<B? A:B;
}
```

Тепер цю функцію можна застосовувати з довільним типом

параметрів:

min(10,15) // виклик функції int min(int,int);

min(8.5,4.3) // виклик функції min(double,double);

Компілятор генерує тіло функції (створює конкретизацію функції) у той момент, коли зустрічає її виклик і робить це відповідно до типу фактичних параметрів.

Шаблони типу мають глобальний статус, вони не можуть бути вкладені в клас або бути локальними в деякій функції.

Якщо використовується шаблон, що вказує тип значення, яке повертається, то він повинен збігатися із шаблоном одного з параметрів. Наприклад, наступна конструкція не буде надійно працювати:

```
template <class alfa, class beta)
alfa  fun(beta  x)
{ .....................
}
```

Шаблони не можуть бути застосованими для функцій зі змінною кількістю параметрів. При створенні багатофайлових програм (тобто програм, які складаються з декількох об'єктних модулів) треба ураховувати, що застосування шаблонів може призводити до певних проблем.

Спеціальна реалізація функції

Функція із шаблоном містить деякий універсальний алгоритм, якому належить працювати для будь-яких типів даних. Але іноді потрібно для деяких конкретних типів параметрів використати якийсь спеціальний алгоритм. У цьому випадку можна визначити спеціальну реалізацію функції. Для того, щоб перевірити, як діє такий засіб, введіть та виконайте наведену нижче програму.

// Приклад 1

```
 template <class TYPE>
 void swap(TYPE& a, TYPE& b)
                    // функція обміну значеннями
{ TYPE R;
 R=a; a=b; b=R;
}
 void swap(string& s1, string& s2)
                     // спеціальна реалізація swap для типу string
{ string r=s1; s1=s2; s2=r;
 puts("Special Realization!");
}
 void main()
{ float x=3.3, y=4.4;
 string a="aaaaaaaa", b="bbbbbbbb";
 cout << "x=" << x << "  y=" << y << endl;
 swap (x,y);
 cout << "x=" << x << "  y=" << y << endl;
 cout << "a=" << a << "  b=" << b << endl;
 swap(a,b);
 cout << "a=" << a << "  b=" << b << endl;
}
```

Нижче наведено приклад корисної функції з шаблонами, яка обчислює абсолютну величину для наданого значення довільного числового типу.

```
// Приклад 2
 template <class T>
 inline T abs(T x)
{ if (x<0) return -x; else return x;
}
```

Остання функція заміняє собою цілий набір стандартних

бібліотечних функцій: abs(int x), labs(long x), fabs(float x), fabsl(long double x). Крім того, визначена функція abs правильно працює й для комплексного аргументу. У цьому випадку спрацьовує спеціальна реалізація функції abs(x) для комплексного аргументу з стандартного бібліотечного модулю <math.h>.

У наступному прикладі шаблони застосовано для побудови функцій create і del, які призначені для створення і знищення одновимірних масивів з елементами довільного типу.

```
// Приклад 3
  template <class type>
  void create(type* &A, int N)
{ A= new type[N];
  errhalt(A==NULL,"No memory!");
          // виклик функції обробки виняткової ситуації
}
  template <class type>
  void del(type* &A)
{ delete[] A;
}
```

2.9 Консольне введення/виведення

Засоби бібліотеки C
Використання потоків

2.9.1. Засоби бібліотеки C

Консольним введенням/виведенням називають операції, у яких використовуються стандартні пристрої введення/виведення. Останні, в принципі, можуть

перепризначуватися, однак у більшості випадків стандартним пристроєм введення є клавіатура, а стандартним пристроєм виведення інформації - екран монітора.

Засоби програмування операцій консольного введення/виведення можна класифікувати в такий спосіб. Всі бібліотечні засоби можна розділити на дві групи: функції бібліотечних модулів stdio.h і conio.h (це мова С) та засоби, пов'язані з потоками (а це вже С++). І в першому, і в другому випадках існують засоби консольного й файлового введення/виведення. Крім того, функції введення/виведення розподіляють на функції, які виконують форматні перетворення та функції, які не виконують такого роду перетворення при передаванні даних.

Форматне перетворення виведення - це перехід від бінарного коду значення до його зовнішнього, символьного подання. При форматному перетворенні введення виконується зворотний перехід: від символьного подання до бінарного коду. Форматні перетворення величин дійсного типу у більшості випадків призводять до втрати деякої частини інформації. У той же час форматні перетворення величин цілого типу являють собою "точне" кодування і не супроводжуються втратою інформації.

Нижче приводяться описи деяких найбільше часто вживаних функцій консольного введення/виведення із модулів stdio.h та conio.h.

Для введення даних із клавіатури використовується функція

scanf(*формат*, p1, p2,...) ;

де *формат* - специфікація формату, а p1, p2, ... - адреси об'єктів, значення яких вводяться. У найпростіших випадках специфікація формату для введення має такий вигляд:

"%*символ*" , де *символ* - один із символів перетворення формату. Як виглядає специфікація формату у загальному випадку та які можливості керування форматними перетвореннями ми маємо - це можна подивитись у довідковій системи оболонки, яку ми застосовуємо для побудови програми. У середовищі Borland C++ для цього треба вибрати пункт "Help" головного меню вікна оболонки.

Виведення даних на екран здійснюється функцією

printf(*формат*, e1, e2, ...);

де *формат* також є специфікація формату, e1, e2, ... - довільні вирази (або змінні), значення яких виводяться.

Специфікації формату

Специфікація формату для виведення звичайно має такий вигляд:

" % [*ширина*][.*точність*]*символ*" ,

де *ширина* - ширина поля виведення (кількість позицій рядка екрана),

точність - потрібна кількість цифр після десяткової крапки (остання цифра виводиться з округленням),

символ - символ форматного перетворення, пов'язаний з типом значення, яке передається.

Нижче наведені формати, які використовуються найбільш часто.

%c - введення/виведення символу (тип char);
%s - введення/виведення рядка символів (тип char*);
%d - введення цілого числа (типи short, int),
виведення цілого числа (типи char, short, int);
%ld - введення/виведення цілого числа (тип long);
%u - введення/виведення цілого числа (типи unsigned

char, unsigned short, unsigned int);

%lu - введення/виведення цілого числа (тип unsigned long);

%X - введення/виведення цілого числа в 16-ричному форматі;

%f - введення/виведення дійсного числа (тип float);

%e - введення/виведення дійсного числа у формі з десятковим порядком (тип float);

%lf - введення/виведення дійсного числа (тип double);

%Lf - введення/виведення дійсного числа (тип long double);

%*d - при введенні пропустити значення, тип якого відповідає символу d,

замість d можна записати символ, що відповідає іншому типу.

Передбачено також формати введення за зразком та інші можливості.

Введення одного символу

Для введення одного символу можна використати функції getchar() і getch().

Функція getchar() повертає черговий символ з буфера клавіатури. Якщо буфер порожній, очікується введення символів, яке повинно завершуватися натисканням клавіші Enter.

Для того, щоб подивитися, як працює функція getchar(), запустіть на виконання таку програму:

```
// Приклад 1
#include <syst.h>
  void main()
{ int c=0;
```

```
  while (c!=10) { c=getchar(); printf("%c=%d \n", c,c); }
}
```

Ця програма виводить на екран символи натиснутих клавіш та їх числові коди. Програма дозволяє натиснути одразу декілька клавіш. Після натискання клавіші Enter робота програми завершується (код цієї клавіші є 10).

Функція getch() вводить символ натиснутої клавіші негайно, без відображення на екрані.

Подивитися, як працює ця функція, можна за допомогою такої програми:

```
// Приклад 2
#include <syst.h>
  void main()
{ int c=0;
  while (c!=27) { c=getch(); printf("%c=%d \n", c,c); }
}
```

Робота програми завершується після натискання клавіші Esc (код її дорівнює 27).

Введення/виведення символьного рядка

Для введення рядка можна використати функцію gets(), для виведення - процедуру puts(s), де s - змінна типу char*, що задає рядок. Процедура puts автоматично додає символ завершення рядка '\n'. Це дає можливість записувати операцію переміщення курсору у початок наступного рядка таким чином:

```
  puts("");
```

Розглянемо приклад ще однієї програми, яка виводить символ натиснутої клавіші, а також його числовий код в 16-

ричному та десятковому відображенні. При натисканні клавіші з розширеним (двобайтовим) кодом програма виводить значення другого (ненульового) байта. У програмі використовується цикл типу do-while та функція getkey з модуля <syst.h>, яка повертає код натиснутої клавіші, у тому числі, код клавіші управління. Програма завершує роботу при натисканні клавіші Esc.

```
// Приклад 3
#include <syst.h>
  void main()
{ char s;
  do { s=getkey();
       printf("символ = %c 16-річ.код = %X дес.код = %d \n",
         s,s,s);
}
  while (s!=27);
}
```

2.9.2. Використання потоків

Потоком називають спеціально сконструйований об'єкт, що включає крім даних певний набір функцій, що забезпечує виконання операцій введення/виведення (у тому числі й файлове введення/виведення). Потік може бути асоційовано з деяким пристроєм введення/виведення, з файлом, з певною областю оперативної пам’яті.

Основні операції з потоками мають назву "вставка" й "добування". Вставкою називають запис інформаційної одиниці у потік. Добування – читання чергової інформаційної одиниці з її вилученням із потоку. Для роботи з потоками

використовуються бібліотечні файли **iostream.h** та **fstream.h**, в яких визначена система спеціальних (потокових) класів і об'єктів цих класів - потоків.

Об'єкти-потоки, які призначені для програмування операцій зі стандартними пристроями, називають стандартними потоками. Стандартні потоки є об'єктами класу **iostream**. Нижче наведені імена стандартних потоків та їх призначення.

cin - стандартний пристрій введення (по умовчанню це клавіатура, відповідає імені stdin у бібліотеці функцій введення/виведення мови С);

cout - стандартний пристрій виведення (по умовчанню це екран, відповідає імені stdout у бібліотеці С);

cerr - стандартний пристрій для виведення повідомлень про помилки й виняткові ситуації, при кожній новій вставці буфер, створюваний для цього пристрою, очищується;

clog - пристрій такого ж призначення, як і cerr, однак буфер не очищується, накопичуючи всі повідомлення, які поступають у потік clog.

Виведення у потік

Для виведення в потік (вставки в потік) використовується перевантажена операція із символом "<<". Перевантаження операції означає, що при виконанні зазначеної операції викликається відповідна функція. Операція з символом "<<" виконує вставку у відповідний потік (наприклад, у стандартний потік cout). Перевантажена операція "<<" у цьому випадку є альтернативою функції виведення типу printf. Нижче наведений рядок програми, що виводить на екран повідомлення "Hello, World!" та переводить курсор у початок наступного рядка.

```
cout << "Hello, World! \n";
```

У якості правого операнда операції "<<" можна

використовувати вирази наступних типів: char , char*, int , long, float , double , long double, void*. Крім того, можна виводити значення будь-якого класового типу, для якого операція "<<" перевантажена (наприклад для рядкового типу string).

Операція "<<" повертає посилання на лівий операнд, вона має асоціативність "зліва направо". Це дає змогу записувати ланцюжок операцій виведення:

```
cout << "i=" << i << ",  d=" << d << "\n" ;
```

Введення з потоку

Введення інформації з потоку (добування з потоку) здійснюється перевантаженою операцією ">>". Цю операцію можна використовувати замість функції scanf. Нижче наведений приклад оператора, що вводить значення змінної x із клавіатури.

```
cin >> x;
```

Правий операнд може бути змінною будь-якого припустимого типу. Допускаються ті ж самі типи, як і для операції вставки. Значенням, що повертається, для операції добування ">>" є посилання на лівий операнд. Також, як і операція вставки, операція добування має асоціативність ліворуч. При виконанні оператора

```
cin >> i >> d;
```

з потоку cin добувається перше значення, що привласнюється змінній i, а потім добувається наступне значення для змінної d.

Якщо введення значення завершується невдало (наприклад, через невідповідність типу змінної та значення, що вводиться), операція повертає значення null. Це дає можливість записувати оператори виду:

```
int  x;
while (cin >> x) { ... }
```

Робота такого оператора циклу завершується при введенні неправильного значення (наприклад послідовності символів, які не відповідають цілому числу).

Форматні перетворення

При виконанні операцій введення/виведення за допомогою потоків форматні перетворення виконуються автоматично. Застосовуються формати, які відповідають прийнятому умовчанню. Виконуване перетворення залежить від типу значення, що передається, і від встановлених прапорів стану форматування. Ці прапори - статичні елементи класу ios. Приклади деяких прапорів форматування:

ios::hex - виведення в 16-ричному форматі;

ios::uppercase - при виведенні 16-ричних чисел літери A-F виводити у верхньому регістрі;

ios::showbase - перед 16-ричним числом вставляються символи "0x";

ios::scientific - виведення дійсних чисел у науковій нотації, тобто в експонентній формі;

ios::left - вирівнювання по лівому краю.

Функція setf - член класу потоку, встановлює, а функція unsetf знімає прапор форматування. У наступному прикладі встановлюється виведення в 16-ричному форматі з використанням великих букв A-F :

cout.setf(ios::hex | ios::uppercase);

Форматними перетвореннями можна управляти також і за допомогою спеціальних функцій - маніпуляторів. Маніпулятор setw(n) встановлює ширину поля для виведення поточного значення. Наприклад, при виконанні наступного рядка значення змінної i буде виводитися в поле, шириною 4, а значення змінної j - у поле, шириною 6 позицій:

```
cout << setw(4) << i << setw(6) << j << endl;
```

При використанні маніпуляторів необхідно підключити файл iomanip.h. Нижче наведено список найбільше часто використовуваних маніпуляторів:

dec - виконувати десяткові перетворення;

hex - виконувати 16-ричне перетворення;

endl - вставка символу нового рядка й очищення потоку;

ends - вставка кінцевого нульового символу в рядок;

setprecision(n) - завдання точності для чисел із плаваючої крапкою;

setw(n) - завдання ширини поля введення/виведення.

Ширину поля виведення можна встановлювати також за допомогою функції потоку width(n). Наприклад, послідовність рядків

```
cout.width(4);
cout << i;
cout.width(6);
cout << j;
cout << "\n";
```

еквівалентна наведеному вище прикладу виведення значень змінних i,j.

В якості практичної вправи подивіться, як виконується наступна програма:

```
// Приклад 1
#include <iostream.h>
 void main()
{ cout.setf(ios::hex|ios::uppercase);
 unsigned a  = 65535;
 cout << "Hexagonal a = " << a << endl;
 cout.setf(ios::dec);
```

```
  cout << "Decimal   a = " << a << endl;
  cout << hex << "Hexagonal a = " << a << endl;
}
```

Введення/виведення без форматних перетворень

Використовуючи компонентну функцію **write** класу **iostream** можна здійснити вставку значення об'єкта **x** без форматного перетворення. Прототип цієї функції має такий вигляд:

```
cout.write((char*)&x, sizeof(x));
```

Як видно, функція write просто копіює послідовність байтів заданої довжини в потік cout.

Добування з потоку без форматного перетворення робиться за допомогою компонентної функції read :

```
cin.read((char*)&x , sizeof(x));
```

Параметри цієї функції мають такий же зміст, як і для функції write.

Нижче наводиться програма, що вирішує задачу про паралельні резистори. Для операцій введення/виведення використовуються потоки.

```
// Приклад 2
#include <syst.h>
  void main()
{ int i,N;
  float r[100], R, S=0;
  cout << "Number of the Resistors N = ";  cin >> N;
  for (i=0;i<N;i++)
    { cout << "r[" << i << "] = ";  cin >> r[i]; }
  for (i=0;i<N;i++) S=S+1/r[i];
  R=1/S;
```

```
  cout << "Common resistance R = " << R << "\n";
  cout << "Relative part of the thermal power, % \n";
  for (i=0;i<N;i++)
    cout << "p[" << i << "] = " << R/r[i]*100 << "\n";
}
```

СПИСОК ЛІТЕРАТУРИ

1.Бочарова Т.А. Основы алгоритмизации: Учебное пособие / Т.А. Бочарова, Н.О. Бегункова. - Хабаровск: Изд-во Тихоокеан. гос.ун-та, 2011. - 64 с.

2. Пінчук В.П., Лозовська Л.І. Програмування мовою С/С++ з прикладами та вправами. Навчальний посібник. - Запоріжжя, ЗНТУ, 2008. – 197 с.

3. Подбельский В.В., Фомин С.С. Программирование на языке С. — М.: Финансы и статистика, 2005. — 600 с.

4. Кочан Стефан. Программирование на языке С. — М.: из-во Вильямс, 2007. 496 с.

5. Мартынов Н.Н. Информатика: С для начинающих. — М.: КУДИЦ-ОБРАЗ, 2006. — 304 с.

Електронні ресурси:

1.http://victana.lviv.ua/knyhy/konspekty-lektsii/142-osnovy-alhorytmizatsii-ta-prohramuvannia

2.http://edu.lp.edu.ua/moduli/programuvannya-chastyna-1-osnovy-algorytmizaciyi-ta-programuvannya

3. Лекции по С++. Томский Политех

4. Язык программирования С++. Лекции и упражнения. Стивен Пратта.