**Лекція № 5. Структура програми. Операції порівняння, логічні операції. Оператори вибору.**

*Контрольні запитання*

1. Для чого слугують заголовні файли?
2. Як компілятор шукає розташування заголовних файлів?
3. Які маніпулятори використовуються для завдання довжини поля, дробової частини числа?
4. Який заголовний файл використовується для маніпуляторів завдання довжини поля, дробової частини числа?

**Структура програми**

Основними частинами типової структури програми на С++ є такі:

* директиви препроцесорної обробки;
* опис зовнішніх змінних (вихідних даних і результатів) та функцій;
* функції програми;
* головна функція — програми **main().**

Порядок дій в головній функції повинен відповідати логікі дій. Наприклад для проведених ЛР, структура програми повинна мати вигляд:

1. Оголошення констант та змінних.
2. Введення даних, необхідних для проведення розрахунку.
3. Проведення розрахунку.
4. Виведення результатів розрахунку.

Приклад програми.

**#include <iostream> *// підключення потокового введення-виведення***

**#include <iomanip.h> //для маніпуляторів setw, setprecision**

**#define \_USE\_MATH\_DEFINES // для використання констант С++**

**#include <cmath> // для використання математичних формул**

**#include <Windows.h>// для зміни кольору консолі**

**using namespace std; *// простір імен***

**int main() *// головна функція***

**{const float a =1, b=1; // константи**

**double x, // змінна, що вводиться+**

**y1, y2, y3, y4, y5, y6; // змінні з результатами**

**/\*Отримання дескриптора для зміни кольорів консолі**

**Колір фону білий, колір тексту чорний\*/**

**HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);**

**system("color F0");**

**cout<< "\nx="; cin >> x; // введення даних**

**y1 = a/x + sqrt(x\*x +1);**

**y2 = a/x + sqrt(x\*x + y1);**

**y3 = a/x + sqrt(x\*x + y3);**

**y4 = a/x + сos(M\_PI\*3/4);**

**y5 = a/x + pow (M\_E,x);**

**y6 = a/x + sqrt(x\*x + y3);**

***// виведення повідомлення на екран***

**cout<<"\nResults"<<endl;**

**cout<<" 1 \t 2 \t 3 \t 4 \t 5 \t 6 \n";**

**cout<<setw(7) << setprecision(2)<< y1<<"\t";**

**cout<< setw(7) << setprecision(2)<< y2<<"\t";**

**cout<< setw(7) << setprecision(2)<< y3<<"\t";**

**cout<< setw(7) << setprecision(2)<< y4<<"\t";**

**cout<<setw(7) << setprecision(2)<< y5<< "\t";**

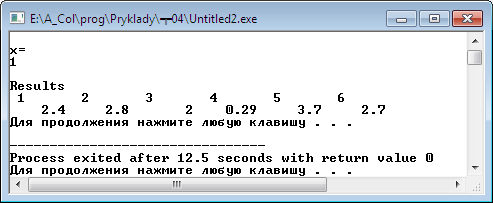
**cout<<setw(7) << setprecision(2)<<y6<<endl;**

**system("pause"); *// пауза***

**return 0; *// повернення результату (вихід із функції)***

**}**

Результати виконання.



**Макровизначення.**

***Макровизначення*** (*макропідстановка* або просто *макрос*)– це вираз, який при компіляції файлу з кодом програми підставляється замість символьного імені, що визначає дану макропідстановку. Макроси дозволяють замінити великий текст коротким ім'ям макросу.

Визначаються макроси директивою препроцесора ***#define***. Формат задання:

***#define ім'я\_макросу текст\_макросу***

Перед компіляцією замість імені макроса в програму підставляється його значення.

Наприклад,

#define РІ 3.14159

Дана директива вказує замінити всі появи у тексті програми символьної константи РІ на чисельну константу 3.14159. Якщо є необхідність змінити значення такої константи у всій програмі, то достатньо це зробити в одному місці програми, в директиві #define, і після повторної компіляції програми всі включення константи в програму будуть автоматично замінені.

Дозволяються також макроси з параметрами, останні підставляються в текст заміни, після чого макрос розширюється, тобто в програму підставляється текст заміни замість ідентифікатора і списку параметрів. Наприклад, для макровизначення з одним аргументом для знаходження площі круга:

#define CIRCLE\_AREA (x) ((PI)\*(x)\*(x))

Кожен раз, коли в програмному коді з’явиться CIRCLE\_AREA (x), значення х підставляється замість х в тексті заміни, символьна константа РІ автоматично заміняється її значенням (визначеним раніше), і макрос розширюється в програмі. Наприклад, оператор

area=CIRCLE\_AREA (4)

розширюється в

area=((3.14159)\*(4)\*(4)).

Макроси є спадщиною мови С, і в С++, зазвичай, замінюються більш безпечними можливостями мови, такими, як шаблони, inline-функції, перерахування enum, перевизначення типів typedef.

Символьні константи і макроси можна відмінити, використовуючи директиву препроцесора ***#undef***. Ця директива відміняє визначення символьної константи або імені макроса. Область їх дії продовжується від місця визначення в програмному коді до місця відміни визначення за допомогою директиви #undef, чи до кінця файлу, якщо дана директива відсутня. Після дії директиви #undef дозволяється повторно визначати макрос за допомогою директиви #define.

**Директиви умовної компіляції**.

Умовна компіляція дозволяє управляти виконанням директив препроцесора і компіляцією програмного коду. Директиви умовної компіляції: #if, #elif, #else та #endif. Кожна конструкція #if закінчується #endif. Умовні конструкції препроцесора, що перевіряють декілька варіантів, реалізуються за допомогою #elif (еквіваленту else if структури if) і #else (еквіваленту else структури if). Порядок використання директив умовної компіляції:

**#if константний\_вираз\_1**

**…**

**#elif константний\_вираз\_2**

**…**

**#elif константний\_вираз\_3**

**#else**

**…**

**#endif**

Кожна із умовних директив препроцесора оцінює значення цілочисельного виразу. Якщо цілий константний вираз у директиві #if має ненульове значення (TRUE), то при компіляції включаються всі наступні рядки до #elif або #endif або #else (elif діє як гілка else-if). Наприклад, змінюючи константу VERSION, можна керувати включенням файлів:

#define VERSION 3

#if VERSION == 1

#define INCLUDE\_FILE "file\_1.h"

#elif VERSION == 2

#define INCLUDE\_FILE "file\_2.h"

#else

#define INCLUDE\_FILE "file\_3.h"

#endif

#include INCLUDE\_FILE

**Зовнішні оголошення**

*Зовнішні оголошення* – це глобальні описи даних (змінних або констант). Змінна у програмуванні є моделлю (зображенням) об’єкта в пам’яті комп’ютера. На фізичному рівні поняттю змінної відповідає група комірок оперативної пам'яті. Ці адреси ставляться у відповідність ідентифікаторам (іменам) змінних під час їх оголошення. Таким чином, ім'я змінної *вказує* (або *посилається*) на на першу комірку з групи, а величину групи визначає тип змінної (рис. 1).

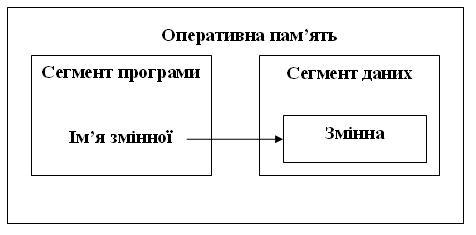


Рис. 1. Ідентифікатор змінної та його асоціація з коміркою пам'яті

**Функція** – це частина програми, що виконує логічно завершений набір дій. В C/С++ вона є єдиним способом представлення підпрограм. Формат оголошення функції:

**тип\_результату ім'я([параметри]) *// заголовок функції***

**{ … } *// тіло функції***

Тіло функції може містити описи констант, змінних і оператори мови, які відокремлюються символом «;». Описи даних можуть зустрічатися в будь-якому місці функції, але до першого звертання до змінної. Якщо параметри у функції відсутні, то в заголовку все рівно записуються круглі дужки. Одна з функцій програми на мові C/С++ повинна мати ім'я main***.*** Саме з неї починається виконання програми. Функція main визначає дії, що виконуються програмою, і викликає інші функції. Якщо програма містить тільки одну функцію, то вона і є головною (має ім'я main).

Програма може містити довільне число директив препроцесора, оголошень і визначень змінних, описів функцій. Порядок появи цих елементів у програмі досить істотний: він впливає на можливість використання змінних, функцій і типів у різних частинах програми.

Логічні операції та операції порівняння в мові Сі оперують поняттями "істина" і "хиба

**Операції порівняння**

У мові С++ можна використовувати операції порівняння:

|  | **операція** | **відповідник** | **приклади** |
| --- | --- | --- | --- |
| Менше  Менше або дорівнює  Більше  Більше або дорівнює | <  <=  >  >= | <  ≤  >  ≥ | a < b  a<=b  a>b  a>=b |
| Дорівнює  Не дорівнює | ==  != | =  ≠ | a==b  a!=b |

Порівнювати можна операнди будь-якого типу, але вони повинні бути того ж самого вбудованого типу даних (порівняння на рівність і нерівність працює для двох величин будь-якого однакового типу даних), або між ними повинна бути визначена відповідна операція порівняння. Результат – логічне значення true або false. Використовуються для порівняння значень і дають відповідь false (0) true (1).

Операнди можуть бути символи, логічні, числові, покажчики.

Умова, яка перевіряється, може містити значення, яке не є логічним. Тоді значення нуль (0) сприймається як false, а значення  ≠0 - як true. Наприклад:

**int a = 10;**

**int b = 4;**

**int c = a == b; // 0**

**int c = 10 <= 4; // 0**

**Логічні операції**

Логічні операції реалізують операції математичної логіки. Серед логічних операцій є унарні й бінарні.. Дають відповідь типу bool:  false (0)  та true (1).

| **Операція** | **Відповідник** | **Приклади** |
| --- | --- | --- |
| ! (унарна) | not | !b, !(c>1 && c<9) |
| &&(бінарна) | and | a && b, a>1 && c<9 |
| || (бінарна) | or | a||b, c<0 || c>10 |

Операнди логічних операцій можуть бути логічні змінні та відношення.

Правила виконання логічних операцій

Операція **not**:

|  |  |
| --- | --- |
| B | !B |
| true(1) | false(0) |
| false(0) | true(1) |

Операція **and**:

|  |  |  |
| --- | --- | --- |
| && | true(1) | false(0) |
| true(1) | true(1) | false(0) |
| false(0) | false(0) | false(0) |

Операція **or**:

|  |  |  |
| --- | --- | --- |
| || | true(1) | false(0) |
| true(1) | true(1) | true(1) |
| false(0) | true(1) | false(0) |

**Логічні вирази**

Логічні (булівські) вирази складаються з  логічних операцій та операндів - логічних змінних і значень та відношень, з урахуванням пріоритетів. Для зміни порядку обчислень можна використовувати круглі дужки. Результат виконання має тип ***bool***. Правила побудови загальні - лінійна форма написання, явно вказуються знаки операцій та операнди без дублювання.

Приклади запису логічних виразів:

**x<0 || x>11**

**(i!=0 ||i>100) && (j!=i || j>0)**

**x!=0 && x!10 && x!=100**

Вираз може містити операнд, який не є логічним. Тоді значення нуль (0) сприймається як false, а значення  ≠0 - як true. Приклади:

**// логічні операції**

**bool res;**

**int a, b;**

**// операція && (AND)**

**a = 8;**

**b = 5;**

**res = a && b; // res = True**

**a = 0;**

**res = a && b; // res = False**

**// операція || (OR)**

**a = 0;**

**b = 0;**

**res = a || b; // res = False**

**b = 7;**

**res = a || b; // res = True**

**// операція ! (логічне "НІ")**

**a = 0;**

**res = !a; // res = True**

**a = 15;**

**res = !a; // res = False**

**Тернарна операція (?:)**

Тернарна операція —це операція, що має 3 операнда. Використовується як альтернатива запису умовного оператора. Синтаксис тернарної операції:

**B ? V1 : V2.**

Виконання:

1. Спочатку обчислюється значення виразу B.

2. Якщо B==true(1), то обчислюється значення виразу V1,

інакше - обчислюється значення виразу V2.

Для читабельності  рекомендується брати у круглі дужки умову та вирази. Приклади:

**(i < 1) ? 1 : i;**

**max = (a<=b) ? b : a;**

**х<0 ? -x : x ;**

**Інструкції розгалуження**.

Інструкції розгалуження реалізують оператори вибору, до яких відносять оператор умовного переходу **if** та оператор-перемикач **switch**.

Оператор умовного переходу **if** використовується для розгалуження процесу обчислень на два напрямки і має такий два форми запису: повна та неповна.

Повна формамає вигляд

**if (*умова*) *інструкція1* else *інструкція2;***

Слова **if** та **else** є зарезервованими, дужки навколо умови обов'язкові. Цій формі оператора відповідає блок-схема на рис. 1 а). В операторі ***умова*** — це вираз, який має логічне значення (**true** — «істина» або **false** —«хиба, неправда»).

Реалізується оператор **if** в цій формі таким чином: спочатку обчислюється вираз ***умова***; якщо значення виразу не дорівнює нулю («істина»), виконується **інструкція 1**, в протилежному випадку —**інструкція 2** і далі управління передається оператору, що є наступним за умовним оператором **if**. Приклад:

**if ( i < j ) і++;**  
**else**  
**{ j = i-3; i++; }**

Неповна формамає вигляд

**if (*умова*) *інструкція1;***

В цій формі оператора **if** друга частина (тобто **else**) може бути відсутня і тоді, якщо вираз приймає значення **false** («хиба, неправда»), виконується зразу наступний оператор програми, що розташований за умовним. Таку конструкцію називають “пропуск оператора”. Цій формі оператора відповідає блок-схема на рис. 1 б). Приклад:

**if ((a>0)&&(b>0)&&(c>0)&&(a+b>c)&&(a+c>b)&&(b+c>a))**

**соut << "a,b,c –сторони трикутника"<<endl;**

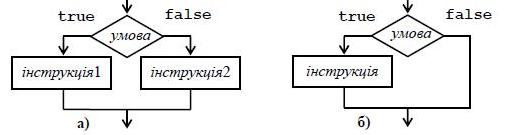


Рис.1 Блок-схеми двох форматів інструкції розгалуження

Коли у будь-якій гілці розгалуження необхідно виконати декілька операторів, їх слід розташовувати у блоці, інакше компілятор не зможе зрозуміти, де закінчується розгалуження. блок може включати різні оператори, у тому числі описи та інші умовні оператори, але не може складатися з одних описів. Потрібно ураховувати, що змінна, яка описана у такому блоці, за межами блока не існує. Синтаксис C++ припускає, що у випадку застосування вкладених умовних операторів кожне **else** відповідає найближчому до нього попередньому **if**. Наприклад:

**if (с > d** **&& (с > r** **|| с = = 0)) d++;**  
**else**  
**{d** **/= с; с = 0 }**

тут декілька умов слід об’єднати знаками логічних операцій, а сукупність операцій необхідно розмістити у блоці.

Обчислення найбільшого значення серед трьох змінних **а, b, с**:

**if (а > b)  
{ if (а > с) max = а;**  
**else max = с; }**  
**else { if (b > c) max = b;**  
**else max = c; }**

У цьому фрагменті програми фігурні дужки можуть бути відсутні, тому що компілятор відносить частину **else** до найближчого **if**;

**if(p++) ps++;** — вираз не використовує операцій відношення.

**Оператор-перемикач** **switch** називають оператором множинного розгалуження. Він використовується для вибору одного а багатьох варіантів рішення і має таку форму запису:

**switch (L)**

**{**

**case к.в.1: *інструкція 1*; [break;]**

**case к.в.2: *інструкція 2*; [break;]**

**..................................................**

**case к.в.n: *інструкція* n; [break;]**

**[default: *інструкція* n+1;]**

**}**

де **switch, case, default** — службові слова;

**break** — оператор (необов’язковий) здійснює вихід з оператopa **switch**;  
**L** — будь-який вираз одного з цілих типів; цей вираз називається **селектором варіантів**.

**к.в.1, …, к.в.n** — константні вирази, які не можуть повтoрюватися і не можуть містити змінних чи викликів функцій; зазвичай, це ціла або символьна константа. Ці значеннями називаються **мітками варіантів**;

**оператор 1; …** – будь-які оператори мови C++.

У процесі виконання цього оператора спочатку обчислюється значення виразу **L**, потім це значення порівнюється (послідовно зверху донизу) зі значеннями константних виразів **к.в.1, …, к.в.n.** У випадку збігу значень **L** і одного з цих константних виразів та, якщо наприкінці гілки розгалуження (варіанту) немає оператора **break**, виконуються всі оператори, починаючи з відповідної гілки. За наявності оператора **break** виконується тільки оператор, що знаходиться у відповідній гілці розгалуження, і керування передається оператору, який розташований за межами oпeратора **switch**. Якщо значення виразу **L** не збігається з жодним із значень константних виразів, то виконується гілка **default** і здійснюється вихід з оператора **switch**. У випадку, коли в операторі **switch** відсутня гілка **default** (вона необов’язкова) і значення **L** не збігається з жодним із значень константних виразів, відбувається вихід з оператора **switch**. Варіант / гілка з міткою **default** можна записати будь-де, але рекомендується записувати його останнім.

Приклад використанням оператора **switch**:

**int а=2;**

**switch (а)**

**{**

**case 1: func1( );**

**case 2: func2( );**

**case 0: …**

**case 4: func3( );**

**default: printf ("good bye \n");**

**}**

Тут оператор **switch** передбачає реалізацію функцій **func2(), func3()** і **default: printf (“good bye \n”);.**

Фрагмент програми можна записати по-іншому:

**int а=2;**

**switch (а)**

**{**

**case 1: funcl( ); break;**

**case 2: func2( ); break;**

**case 0: ...**

**case 4: func4( ); break;**

**default: printf ("good bye \n");**

**}**

Оператор-перемикач приводить до виконання тільки гілки

**case 2: func2( ); break;** після чого здійснюється вихід із **switch**.

Щоб написати кілька інструкцій там, де за правилами мови має бути одна, наприклад, як гілку в умовній інструкції, використовують **блок** – послідовність інструкцій у дужках **{}**. Він має такий загальний вигляд:

**{**

***інструкція***

**...**

***інструкція***

**}**

Виконання блоку полягає в послідовному виконанні інструкцій, записаних у ньому.

*Для самостійного вивчення (2 години)*: Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

***Література***

1. Ковалюк Т. В. Алгоритмізація та програмування: Підручник. — Львів: «Магнолія 2006», 2013. — 400 с., ил.
2. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. / Ю. А. Бєлов, Т. О. Карнаух, Ю. В. Коваль, А. Б. Ставовський. – К. : Видавничо-поліграфічний центр "Київський університет", 2012. – 175 с. с.: іл. ISBN (укр.) . URL: <http://csc.knu.ua/uk/library/books/belov-24.pdf>
3. Зелковиц М., Шоу А., Геннон Дж. Принципы разработки программного обеспечения. — М.: Мир, 1982. — 368 с. URL: [http://computersbooks.net/index.php?id1=4&category=teoriyaprogramirovaniya& author=zelkovic-m&book=1982](http://computersbooks.net/index.php?id1=4&category=teoriyaprogramirovaniya&%20author=zelkovic-m&book=1982)
4. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..

*Контрольні запитання*.

1. Визначте основні частини типової структури програми на С++.
2. Що таке макровизначення? Наведіть приклади.
3. Для чого використовується умовна компіляція?
4. Що може міститися в зовнішніх оголошеннях?
5. Надайте визначення функції.
6. Які значення може містити умова, що перевіряється?
7. Як визначається результат операція **not?**
8. Як визначається результат операція **and?**
9. Як визначається результат операція **or?**
10. Поясніть, що представляє собою логічний вираз.
11. Як визначається тернарна операція?
12. Які оператори реалізують розгалуження у програмі?
13. Як діє умовний оператор if?
14. Як працює оператор-перемикач switch?