**Лекція № 6. Виведення значень виразів. Блок. Область дії оголошення імені. Інструкції розгалуження. Прості інструкції повторення обчислень.**

*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.

Для програми останньої ЛР, з урахуванням її специфіки, загальний вигляд може бути таким.

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

**#include <iomanip> //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 G;**

**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 +1); // замінити на потрібну формулу**

**y3 = a/x + sqrt(x\*x +1); // замінити на потрібну формулу**

**y4 = a/x + сos(M\_PI\*3/4); // замінити на потрібну формулу**

**y5 = a/x + pow (M\_E,x); // замінити на потрібну формулу**

**y6 = a/x + sqrt(x\*x +1); // замінити на потрібну формулу**

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

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

**cout<<"\t 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";**

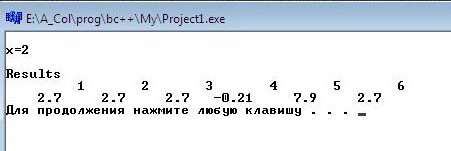
**cout<< setw(7) << setprecision(2)<< y4<<"\t";**

**cout<< setw(7) << setprecision(2)<< y5<<"\t"<<y6<<endl;**

**system("pause"); *// пауза***

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

**}**



Результати виконання.

В C++ оператори об’єднуються в блоки — функції. **Функція** — це послідовність операторів. Кожна програма в C++ повинна містити **головну функцію main()**. Саме з першого оператора в main() і починається виконання програми. Функції, як правило, виконують конкретне завдання. Наприклад, функція max() може містити оператори, які визначають максимальне число з двох переданих їй. Функція calculateGrade() може вираховувати оцінку студента..

**Порада**: Пишіть функцію main() в файлі .cpp з тим же іменем, що має проект. Наприклад, якщо ви пишете програму Chess, то помістіть вашу функцію main() в chess.cpp.

У загальному випадку програма складається з декількох функцій, що не перетинаються (тобто «вкладення» однієї функції в іншу неприпустиме). Перед функціями і між ними можуть бути присутні оголошення об’єктів даних і оператори препроцесорної обробки. Функції користувача, які викликаються у головній функції main(), слід обов’язково описати до їх використання. Наведемо приклад запису фрагмента простої програми:

**//..............фрагмент простої програми на С++**

#include <iostream>

using namespace std;

const int n = 20;

void main()

{

system("color F0");

float mas[n]; //опис одновимірного масиву

//..............введення елементів масиву

for (int i = 0; i < n; i++)

cin >> mas[i];

//......виконання перетворень

//......виведення перетвореного масиву

for (int i = 0; i < n; i++)

cout << " " << mas[i] << " ";

}

**Коментарі** необхідні для пояснень призначення тих чи інших частин програми і їх текст завжди ігнорується компілятором. Мова С++ використовує два різновиди коментарів:

* *// текст* — **однорядковий коментар**, який починається з двох символів «/» («коса риска») і закінчується символом переходу на новий рядок;
* */\* текст \*/* — **багаторядковий коментар**, що розташовується між символами-дужками «/\*» і «\*/».

Багаторядкові коментарі не можуть бути вкладеними один в одний, а однорядкові коментарі можна вкладати в багаторядкові коментарі. Багаторядкові коментарі доцільно застосовувати для тимчасового виключення блоків при налагодженні програми.

Наведемо кілька порад стосовно раціонального складання коментарів:

* коментарі повинні бути добре складеними реченнями, мати правильну пунктуацію та містити тільки потрібну для супроводу інформацію;
* пропуск — один з найбільш ефективних коментарів, що значно поліпшує розуміння програми;
* штрихові лінії коментаря або порожні рядки застосовуються для поділу функцій та інших логічно завершених фрагментів програм.

Директива препроцесора **#include<iostream>** забезпечує підключення до програми засобів зв’язку зі стандартними потоками введення-виведення даних. Ці засоби знаходяться у заголовному файлі **iostream.h**, де **і (*input*)** — введення, **о (*output*)** виведення, **stream** — поток, **h (*head*)** — заголовок. Ураховуючи те, що середовище **Borland C++** не забезпечує затримку результатів на екрані, у текстах програм посібника з цією метою використано стандартний потік введення **сіn** (бібліотека **iostream.h**), функцію введення **getch()** (бібліотека **conio.h**) тощо.

При створенні програми враховують такі основні вимоги:

* усі використані константи, змінні, функції та нестандартні типи повинні бути оголошеними (описаними) до їхнього першого використання, і ці оголошення можна розміщати в будь-якому місці програми;
* кожний оператор мови закінчується символом **«;»;**
* фігурні дужки (**« { »** та **« } »**) виділяють складений оператор і все, що подано між такими дужками, синтаксично сприймається як один оператор;
* вкладені блоки повинні мати відступ у 3-4 символи, при цьому блоки одного рівня вкладеності слід вирівняти за вертикаллю.

**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.

Директива **#define** – слугує для заміни у всіх місцях коду програми замість вказаної послідовності символів обрану користувачем. За негласною домовленість слова підстановки записують великими літерами, для візуального виділення у програмі.

Наприклад,

#define MAXNUMBER 100  
У всіх місцях коду замість MAXNUMBER буде підставлено 100.  
Тобто при компіляції рядок  
if (a>MAXNUMBER) { … }  
Буде замінено на:  
if (a>100) { … }

#define SIMPLETEXT “lorem ipsum dolor sit amet”  
Аналогічно замість:  
printf(SimpleText);  
printf(“lorem ipsum dolor sit amet”);

#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).

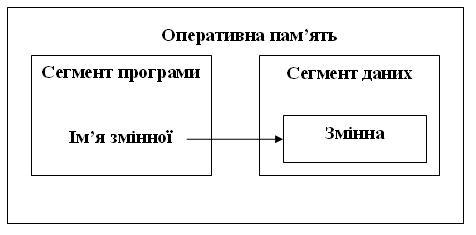


Рис. 1. Ідентифікатор змінної та його асоціація з коміркою пам'яті

**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**

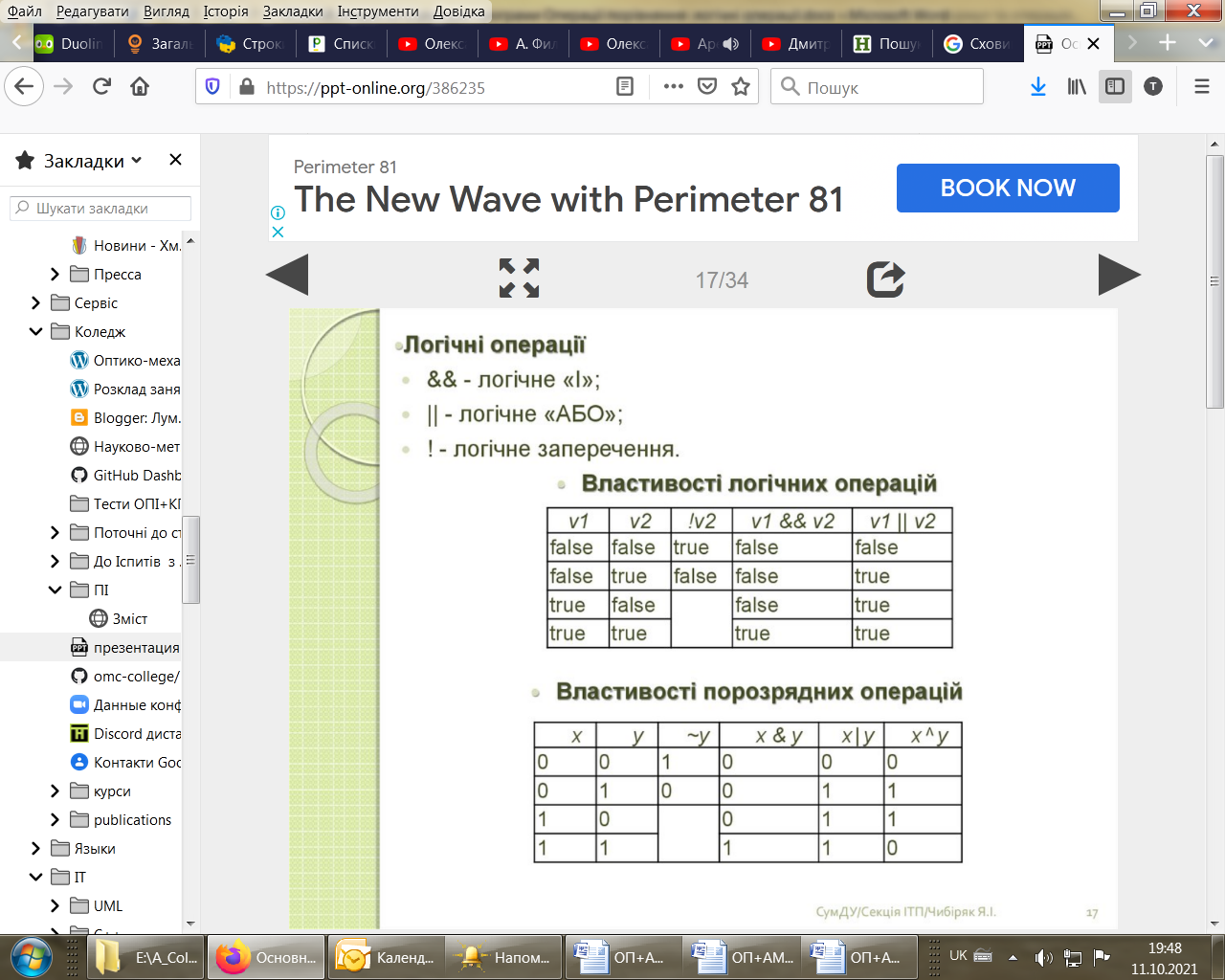
**Логічні операції**

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



**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;**

**abs1=(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 ( i >= j ) j = i-3;**

**i++;**

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

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

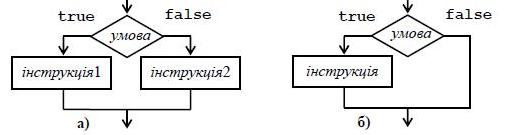


Рис.1 Блок-схеми двох форматів інструкції розгалуження

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.

**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**

**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 oпeратору **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.

**Прості інструкції повторення обчислень**

Циклічні, тобто повторювані, обчислення задають за допомогою оператору циклу. Який має форми:

Інструкція циклу з передумовою **while**,

Інструкція циклу з післяумовою **do while**,

Інструкція циклу **for**.

Оператори циклу використовують для здійснення багаторазового повторення деякої послідовності дій. Кожен цикл складається з тіла циклу, тобто операторів, що виконуються декілька разів. Один прохід циклу називається ітерацією.

Інструкція циклу з передумовою **while** виконується, якщо умова перевіряється до початку циклу, і має вигляд:

**while (умова) *інструкція***

Слово **while** є зарезервованим, дужки обов'язкові, **while (*умова*)** – це **заголовок циклу**, а ***інструкція*** – **тіло**.

Інструкція циклу виконується так. Спочатку обчислюється умова в заголовку. Якщо вона істинна, то виконується тіло циклу та знов обчислюється умова. Якщо вона істинна, то все повторюється. Виконання інструкції циклу закінчується, коли обчислено значення умови **false**, тобто хибність. Отже, в останньому циклі тільки обчислюється умова, а тіло не виконується. Якщо при першому обчисленні умова хибна, то тіло циклу не виконується жодного разу. **Ітерація циклу** складається з перевірки умови циклу та виконання після неї тіла циклу. Інструкції циклу з передумовою відповідає блок-схема на рис. 2.

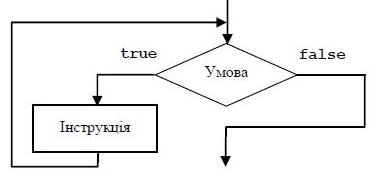


Рис. 2. Блок-схема інструкції циклу з передумовою

Умову в інструкції циклу називають **умовою продовження**, оскільки, якщо вона істинна, то виконання інструкції циклу продовжується. Цикл починається обчисленням умови, тому її ще називають **передумовою**. Інструкції циклу з передумовою застосовують зазвичай тоді, коли кількість повторень циклу наперед невідома.

Приклад: наступна послідовність операторів обчислює суму квадратів перших **n** натуральних чисел:

**int і=0, sum=0; n=100;**  
**while (і < n) sum += ++і \* і;**

Інструкція циклу з **післяумовою**, або **do**-інструкція, має загальний вигляд

**do *інструкція* while (умова);**

**Оператор циклу з післяумовою do while** звичайно застосовується у випадках, коли тіло циклу виконується хоча б один раз. Слово **do** (виконувати) є ключовим. Інструкція циклу з післяумовою виконується так: спочатку виконується тіло циклу, потім обчислюється умова. Якщо вона хибна, то цикл завершується, інакше повторюється тіло й знову обчислюється умова. На відміну від інструкції циклу з передумовою, цикл *починається діями в тілі циклу* та закінчується обчисленням умови. Умова перевіряється після виконання тіла циклу, тому її називають **післяумовою**. Тіло циклу, заданого **do**-інструкцією, виконується обов'язково хоча б один раз (на відміну від **while**-інструкції). Інструкцію циклу з післяумовою використовують, коли потрібно спочатку виконати тіло циклу, і лише потім перевіряти умову продовження. Циклу з післяумовою відповідає блок-схема на рис. 3.

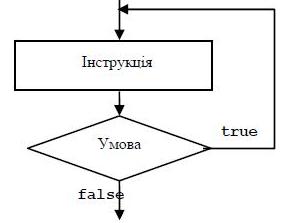


Рис. 3. Блок-схема інструкції циклу з післяумовою

**Приклад.** Потрібно з клавіатури ввести ціле число від 10 до 99. Якщо користувач набрав число за межами цього діапазону, то слід *повторити спробу*. Отже, спочатку треба вводити число, а потім перевіряти умову того, що число є двозначним.

**do {**

**cout << "Enter one integer in [10,99]>";**

**cin >> k;**

**} while (!(10<=k && k<=99)); // 10<=k && k<=99**

Інструкція циклу **for** або **for**-інструкція, має загальний вигляд

**for (*початкова дія*; *умова*; *перехідна дія*)**

***основна дія***

Слово **for** зарезервоване, дужки та два знаки; усередині дужок є обов'язковими. Початкова дія, умова й перехідна дія є *виразами* (кожен із них може бути порожнім), основна дія – *інструкцією*. Тілом циклу **for** називають його основну дію. Інструкція **for** виконується так само, як і інструкції вигляду

***початкова дія*;**

**while (*умова*)**

**{**

***основна дія*;**

***перехідна дія*;**

**}**

Оператор циклу **for** реалізується таким чином:

* виконується початкова дія - вираз ініціювання (виконання цієї нотації може бути здійснено до оператора **for**);
* обчислюється вираз-умова;
* якщо умовний вираз приймає значення «істина» — виконуються оператори циклу;
* обчислюється вираз ітерації;
* знову перевіряється умова;
* як тільки умова прийме значення **0** («хиба, неправда»), керування передається оператору, що розташований за оператором циклу **for**.

Оскільки в операторі **for** перевірка виразу-умови відбувається перед циклом, то у випадку помилкової умови цикл може жодного разу не виконуватися.

Оператор **for** може використовувати декілька змінних, що керують циклом, а будь-які вирази можуть бути відсутніми, наприклад:

**int n, у;**

**for (int k = 0, n = 20;k <= n; k++, n--)**

**y = k \* n;**

або

**int і=0;**

**for (; і < 4; i++)**

Перший фрагмент має два вирази ініціювання і два вирази ітерації. Спочатку відбувається присвоювання значень змінним **k = 0 і** **n = 20**, далі здійснюється порівняння **k <= n** і, якщо ця умова має значення «істина», то буде виконуватися тіло циклу, а потім вираз **k++ і** **n–**, якщо ж умова не виконується, то цикл припиняє свою роботу.

C++ дозволяє поєднати ці дві дії в одному виразі – за допомогою операції послідовного обчислення. Операція зі знаком "**,**" позначає послідовне обчислення виразів, записаних через кому (в прикладі це **к = 0, n = 20;)**. Ця послідовність виразів розглядається як один вираз; його значенням є значення останнього виразу. Операція послідовного обчислення дозволяє на місці одного виразу записати кілька.

Операторам циклів с параметром **for** потрібно віддати перевагу при організації циклів з лічильниками.

В циклі **for** можна використовувати інструкції **break** та **continue.** Інструкція **break** у тілі циклу **for** завершує його виконання, а інструкція **continue** завершує виконання лише тіла циклу; відразу після неї виконується перехідна дія.

**Приклад.** Дуже часто інструкція циклу **for** зустрічається у вигляді

**for (k=0; k<n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 0, 1, 2, …, *n*-1 або у вигляді

**for (k=1; k<=n; ++k) *інструкція***

й задає виконання *інструкції* за значень *k* = 1, 2, …, *n*, або у вигляді

**for (k=n; k>0; --k) *інструкція***

й задає виконання *інструкції* за значень *k* = *n*, *n*-1, …, 2, 1. Змінну **k** у цих ситуаціях інколи називають **лічильником циклу**.

**Переривання break та продовження циклу continue**

Виконання інструкції **break** всередині циклу будь-якого різновиду перериває й завершує цикл; далі виконуються дії, наступні за цим циклом. Якщо **break** записано в інструкції циклу, вкладеній в іншу інструкцію циклу, то виконання **break** завершує вкладений цикл, а зовнішній цикл продовжується.

Інструкція **continue** всередині циклу задає перехід на кінець тіла циклу. В інструкціях циклу з перед- і післяумовою після **continue** обчислюється умова продовження циклу.

**Приклад.** За допомогою клавіатури вводиться послідовність дійсних чисел. Потрібно підрахувати суму її додатних елементів, а за появи 0 видати накопичену суму й завершити роботу.

Запрограмуємо цикл, в якому вводиться й обробляється послідовність чисел. Уведене число зберігаємо в змінній **x**, а суму додатних елементів – у змінній **sum**. Якщо під час уведення трапилася помилка, то подальші дії з уведення не виконуються, а змінна **x** зберігає своє останнє значення. Тому умовою продовження циклу буде саме відсутність помилок (інакше можна отримати цикл, який ніколи не завершиться!). Цю умову задає значення виразу введення **cin>>x**, перетворене до логічного типу.

**#include <iostream>**

**using namespace std;**

**int main()**

**{ double x;**

**double sum=0;**

**cout<<"Enter reals:\n";**

**while (cin>>x){**

**if (x==0.) break; //виходимо з циклу**

**if (x<0.) continue; //пропускаємо від'ємні**

**sum+=x;**

**}**

**cout << "sum=" << sum << endl;**

**system("pause");**

**return 0;**

**}**

Використання інструкції **continue** в цій програмі є дуже штучним. Ще одним недоліком є те, що в кінці не повідомляється, чи були помилки під час уведення. Інструкції програми виконуються *в порядку їх запису в програмі*. Про таку програму кажуть, що вона **структурована**. Інструкції **break** і **continue** *порушують* цей порядок обчислень, заплутуючи текст програми. Тому, користуючись ними, програміст повинен ретельно відслідковувати точку програми, якою продовжуються обчислення. Інколи ці інструкції дійсно скорочують запис розгалужень у циклі, проте в більшості випадків ті ж самі дії *можна описати без них*. Тому краще не зловживати **break** і **continue**.

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

***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?

***Література***

1. 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>
2. Джейс Либерти Освой самостоятельно С++ за 21 день: 3-е изд. пер. с англ.: Уч. пос. – М.: Издательский дом „Вильямс”, 2001. – 816 с.: ил..
3. Бондарев В. М. Программирование на С++: Учеб. пособие. — Харьков: СМИТ, 2004г. — 294 с. URL: <https://www.rulit.me/author/bondarev-v-m/programmirovanie-na-c-get-161082.html>
4. Липпман С. Б., Лажойе Ж. Язык программирования С++: Вводный курс. — М.: ДМК, 2001. URL: <http://www.insycom.ru/html/metodmat/inf/Lipman.pdf>
5. Дейтел Х., Дейтел П. Основы программирования на С++. – М.: Бином, 1999. – 1024 с.