**Лекція № 10. Структура програми.**

**Блок. Область дії оголошення імені. Операції порівняння, логічні операції.**

**Виведення значень виразів.**

*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().**

***Структура програми на С++ в узагальненому виді виглядає так:***

|  |  |
| --- | --- |
| **#include <ім‘я бібліотеки 1>**  **#include <ім‘я бібліотеки 2>**  **…** | Директиви препроцесора   1. підключення заголовних файлів 2. директиви умовної компіляції тощо |
| // прототипи функцій (заголовки) | Оголошення функцій |
| // глобальні ідентифікатори (типи, змінні тощо) | Оголошення глобальних ідентифікаторів |
| **using namespace std;** | Визначення простору імен (за потребою) |
| **int main()** | Головна функція програми |
| **{** |
| //Розділ опису змінних |
| //Розділ операторів |
| **}** |
| // Реалізація функцій | Реалізація оголошених функцій |

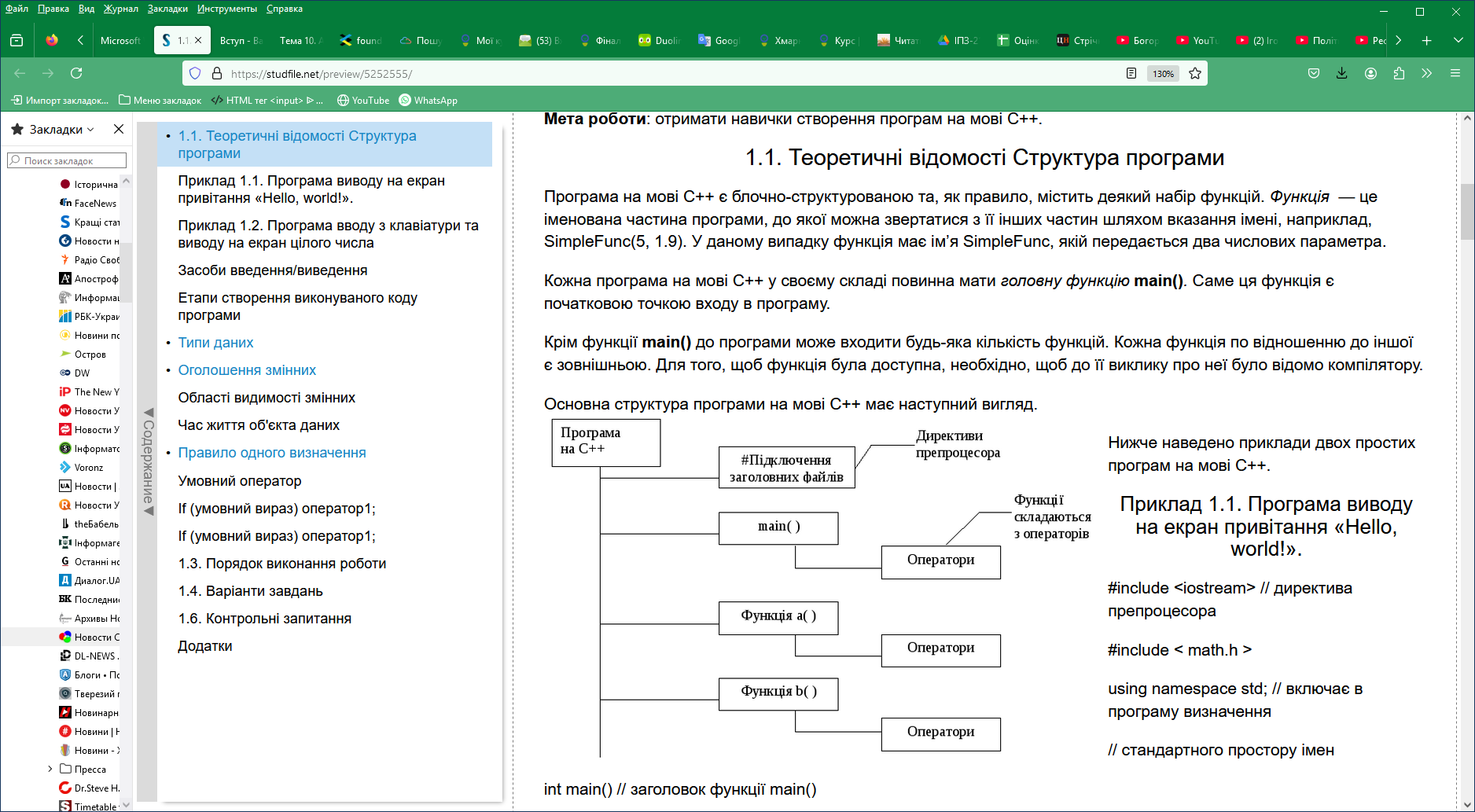


Рисунок 1 - Основна структура програми на мові С++

Порядок дій в головній функції повинен відповідати логіці дій. Наприклад для проведених ПР, 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 ПР, з урахуванням її специфіки, загальний вигляд може бути таким.

**#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 +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";**

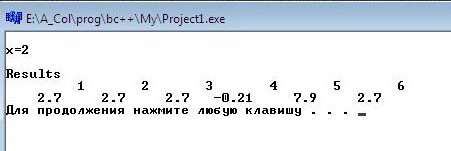
**cout<< setw(7) << setprecision(2)<< y4<<"\t";**

**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.h>

#include <соnio.h> //директиви препроцесора

const int n=20;

void main()

{

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] << " ";

getch (); // затримка результату на екрані

}

**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.h>** 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.

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.

***Деякі важливі зауваження***

***Тип void***

Тип void (порожній) синтаксично поводиться як основний тип. Однак використати його можна тільки як частину похідного типу, об'єктів типу void не існує. Він використовується для того, щоб вказати, що функція не повертає значення, або як базовий тип для вказівників на об'єкти невідомого типу.

***Приклад 1 Тип void***

void f(); //функція не повертає значення

void \* p; //вказівник на об'єкт невідомого типу

void a; //помилка

***Перетворення типів***

Перетворення типів від одного до іншого відбувається при присвоюванні і в змішаних виразах.

float d=3; //дані типу int перетворюються в тип float

В змішаних виразах відбувається два види автоматичного перетворення типів:

1) типи bool, char, unsigned char, signed char, short перетворюються в тип int;

2) якщо після першого кроку вираз має змішаний тип, то відповідно до ієрархії типів операнд з більш вузьким діапазоном перетворюється в операнд з більш широким діапазоном. При цьому загальне значення виразу відповідає більш широкому типу.

***Операція зведення*** ***типів*** полягає у явному перетворенні даних одного типу в дані іншого типу. Ця операція є одномісною та має той же пріоритет, що й унарна операція. Вона наступний вигляд.

(Тип)

***Приклад 2 Операція зведення типів***

int i;

(float)i+1 рівносильно ((float)i) +1

Зведення типів є коректним, якщо воно приводить до розширення типу, і некоректним, якщо воно приводить до звуження типу.

**Класифікація змінних в залежності від місця оголошення в програмі**

Змінні можуть бути оголошені:

* всередині функцій;
* у визначенні параметрів функцій;
* за межами усіх функцій.

В залежності від місця оголошення в програмі, змінні класифікуються наступним чином:

* локальні змінні (оголошуються всередині функцій);
* формальні параметри (є параметрами функцій);
* глобальні змінні (оголошуються за межами функцій).

***Локальні змінні*** – це змінні, які оголошені (описані) всередині функцій. Доступ до таких змінних мають тільки оператори, що використовуються в тілі функції.

Локальні змінні створюються в момент виклику функції. Як тільки виконання функції завершується, локальні змінні знищуються.

Локальні змінні створюються в момент виклику функції. Як тільки виконання функції завершується, локальні змінні знищуються.

Термін “знищення змінної” означає: знищення змінної в оперативній пам’яті після завершення функції, в якій ця змінна була оголошена.

Глобальна змінна – це змінна, доступ до якої мають усі функції в програмі. Глобальні змінні зберігають свої значення на протязі усього часу виконання програми. Цей час називається часом життя глобальної змінної.

**Приклад 3.** У даному прикладі оголошено глобальну змінну maxi та дві функції GetMax2() та GetMax3().

#include <iostream>

#include <Windows.h>

using namespace std;

int maxi=0; // maxi - глобальна змінна

// Оголошення функції GetMax2

int GetMax2(int a, int b)

{ int maxi=1;

// a, b - формальні параметри

// використання глобальної змінної max у функції GetMax2()

if (maxi<b) ::maxi = b;

return maxi;

}

// Оголошення функції GetMax3

int GetMax3(int a, int b, int c)

{ int maxi;

// a, b, c - формальні параметри

// використання глобальної змінної max у функції GetMax3()

if (::maxi<b) maxi = b;

if (::maxi<c) maxi = c;

return maxi;

}

int main()

{system("color F0");

int gma;

**// Зверніть увагу на виведення maxi!**

cout << "1 maxi="<< maxi <<endl;

cout <<" Getmax2(1, 2)=" <<GetMax2(1, 2)<<";Getmax3(4, 5, 6)="<<

GetMax3(4, 5, 6)<< " maxi="<< maxi << endl;

maxi=0;

cout << "2 maxi="<< maxi <<endl; **// Зверніть увагу на виведення maxi!**

cout << "maxi="<< maxi << " Getmax2(1, 2)=" << GetMax2(1, 2)<<";Getmax3(4, 5, 6)="<<GetMax3(4, 5, 6)<<endl;

gma=GetMax2(1, 2) + GetMax3(4, 5, 6);

cout << "maxi="<< maxi<< ";gma="<<gma<<endl;

return 0;

}

***Пригадаємо***

**Області видимості змінних**

При оголошенні змінних у програмі велике значення має те місце, де вони оголошені. Від того, де оголошена змінна, залежать можливість її використання.

У С++ можливі три місця оголошення змінних.

По-перше, поза будь-яких функцій, у тому числі і головної функції main(). Така змінна називається **глобальною** і є **видимою** від місця оголошення **до кінця файлу**.

По-друге, змінна може бути оголошена всередині блоку, у тому числі й всередині тіла функції. Оголошена в такий спосіб змінна називається **локальною** і є **видимою до кінця блоку**. Така змінна поза блоком, у якому вона оголошена, невідома.

По-третє, змінна може бути оголошена як параметр функції. Крім спеціального призначення, а саме для передачі даних у функцію, параметр по відношенню до функції слід розглядати як її локальну змінну.

Таким чином, **в С++ існують 2 області видимості (2 контексти видимості) змінних: блок і файл.**

***Приклад 4***. Оголошення змінних. Програму для обчислення суми k чисел

#include <iostream.h>

void sum(int ); // прототип функції

int s=0; // глобальна змінна

void main()

{

int i,b,k; // локальні змінні

cout<<"\nВведіть кількість доданків";

cin>>k;

for(i=0;i<k;i++)

{

cout<<"\nВведіть новий доданок ";

cin>>b;

sum(b);// виклик функції

}

cout<<"\ns="<<s;

}

void sum(int c)

{

s=s+c;

}

У цій програмі змінна s є глобальною, вона доступна з обох функцій програми — main() і sum(), а змінні i, b, k та с — локальні, доступні тільки в тих функціях, де вони оголошені.

Якщо глобальна й локальна змінні мають одне і теж ім'я, тоді вважається, що оголошені дві різні змінні зі своїми областями використання. При цьому локальна змінна буде видима в тій функції, де вона оголошена, а глобальна у всій програмі за виключенням функції, у якій оголошена локальна змінна.

**Час життя об'єкта даних**

Об'єкт створюється, коли зустрічається його визначення і знищується, коли його ім'я виходить із області видимості.

Оголошення використовуються для визначення інтерпретації, що надається кожному ідентифікатору (імені). Це означає, що потрібно задати тип ідентифікатора щоб повідомити компілятор, до якого виду об'єктів відноситься ім'я.

***Приклад 5 Визначення змінної***

char symbol;

int number = 1;

double sqrt;

Глобальні змінні створюються та ініціалізуються (тільки) один раз й "живуть" до завершення програми (**час життя глобальної змінної – до кінця роботи програми**). Об'єкти, визначені описом із ключовим словом static, поводяться так само.

Не ініціалізована явно статична (static) змінна неявно ініціалізується нулем.

**Час життя локальної змінної – до виходу з блоку**.

***Приклад 6*** Ініціалізація змінної

int a = 1;

void f()

{

int b = 1; // ініціалізація b відбувається з

//кожним викликом функції f()

static int c = 1; // статична змінна створюється тільки

//один раз

cout << " a=" << a++

<< " b=" << b++

<< " c=" << c++ << " \n"; }

int main() { while (a < 4) f(); return 0;}

Результат виконання програми виглядає так:

a = 1 b = 1 c = 1

a = 2 b = 1 c = 2

a = 3 b = 1 c = 3

***Приклад 7 Помилки***

int a;

int a; // помилка! Повторне оголошення

extern int error\_number;

extern short error\_number; // невідповідність типів

**Введення-виведення значень виразів**

Умові С++ основним поняттям уведення й виведення даних є потік – послідовність символів або інших даних. У програмі потік є представником фізичного файлу на зовнішньому носії даних (диску, клавіатурі або екрані монітора), а операції обміну даних із файлом зображено як операції добування даних із потоку або дописування їх до нього.

У програмуванні існує поняття стандартних файлів уведення й виведення – зазвичай ними є клавіатура та екран. У С++-програмі їм відповідають стандартний потік уведення з ім'ям cin і стандартний потік виведення з ім'ям cout. Отже, імена cin і cout, означені у файлі iostream, насправді позначають не клавіатуру та екран, а потоки, що відповідають цим пристроям у програмі. На початку виконання програми обидва потоки cin і cout порожні.

Із засобів обробки потоків розглянемо лише операції введення >>, або добування, даних зі вхідного потоку, і виведення <<, або вставки (insertion), даних у вихідний потік. Розгляд інших засобів виходить за межі цієї книги.

Вираз із операцією введення (вставки з потоку) >> має вигляд cin >> ім'я-змінної. Додавши в кінці виразу символ ;, маємо інструкцію введення.

Виконуючи операцію введення, комп'ютер забирає зі вхідного потоку послідовність непорожніх символів, за якою створює відповідне значення та присвоює змінній. Усі порожні символи пропускаються. Якщо у вхідному потоці немає непорожніх символів (потік порожній), то комп'ютер очікує на його поповнення. Потік поповнюється, коли людина набирає на клавіатурі деяку послідовність символів (вони з'являються на екрані) і натискає на клавішу Enter.

Згідно з типами змінних, в які вводяться значення, операція >> розбиває вхідний потік на лексеми (послідовності непорожніх символів) і перетворює їх на значення певних типів (інтерпретує лексеми). Отже, операцію >> можна розглядати як інтерпретацію вхідного потоку символів.

Значенням виразу введення є той потік, з якого взято символи. До нього знову можна застосувати операцію >>, тобто можливі вирази вигляду

cin >> ім'я-змінної\_1 >> ім'я-змінної\_2

та аналогічні їм із більшою кількістю змінних. Наведений вираз еквівалентний виразу

(cin >> ім'я-змінної\_1) >> ім'я-змінної\_2

й задає послідовне введення значень двох змінних.

Приклад. Якщо є змінні char c1, c2; і під час виконання

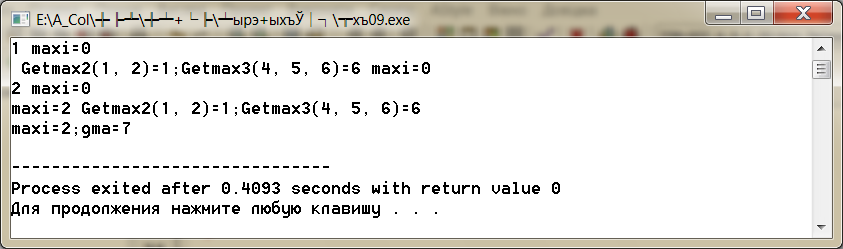
cin >> c1 >> c2; натиснути клавіші Space, W, Space, Space, 3, Q, Enter, то змінні отримають значення 'W' та '3', а Q залишиться у вхідному потоці.

За стандартних налаштувань операція >> пропускає всі порожні символи вхідного потоку.

За стандартних налаштувань, утворюючи за вхідною послідовністю числове значення, операція >> розглядає вхідну послідовність як десяткове зображення числа. Дійсні числа можна задавати в нормалізованій формі, а також без дробової частини. У дробовій частині використовується крапка, а не кома. Для цілих чисел дробова частина й порядок не допускаються; задане число повинно належати діапазону можливих значень типу, який має змінна, інакше введення може мати непередбачувані наслідки.

Приклад. Якщо є змінна int k; і під час виконання cin >> k; натиснути клавіші 0, 1, 1, Enter, то значенням k буде 11. Якщо є змінна double x; і під час виконання cin >> x; натиснуто клавіші 8, Enter, то x отримує значення 8.0, а якщо 1, e, 2, Enter, – значення 100.0.

***Під час уведення з клавіатури помилки найчастіше трапляються з числовими змінними. Якщо під час виконання виразу введення сталася помилка, то вхідний потік переходить у стан помилки, в якому виконання подальших операцій уведення неможливе. За наявності помилки значення змінної, в яку мало відбутися введення, не змінюється. Отже, якщо змінну не ініціалізовано та значення їй не присвоєно, то її значення залишається "випадковим сміттям"!***



При розробці власних програм, що містять глобальні змінні, може виникнути ситуація коли всередині функції використовується локальна змінна з таким самим іменем. У цьому випадку ім’я локальної змінної перекриває ім’я глобальної змінної. При звертанні до імені всередині функції приймається до уваги локальна змінна (глобальна змінна ігнорується). Якщо виникла така ситуація, то для доступу до глобальної змінної можна використати операцію розширення області видимості, яка позначається **::** .

З допомогою операції розширення області видимості **::** можна отримати доступ не тільки до змінних, але і до констант та функцій.

**Приклад.**

У прикладі продемонстровано:

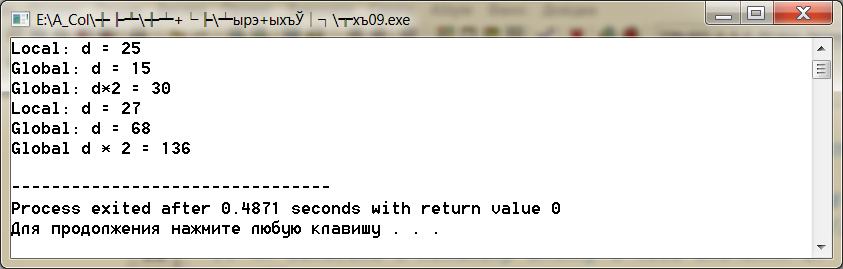
* використання локальних та глобальних змінних;
* використання прототипу функції;
* використання операції розширення області видимості **::** для доступу до глобальної змінної.

#include <iostream>  
using namespace std;  
// Глобальні та локальні змінні. Операція ::  
// Оголосити глобальну змінну d.  
// Ця змінна є доступна в обох функціях: MultGlobal2() та main()  
int d = 15;  
// Прототип функції, яка множить глобальну змінну d на 2  
int MultGlobal2(void);  
int main()  
{  
  // Оголосити локальну змінну d у функції main()  
  int d;  
  // 1. Вивести значення локальної змінної d  
  d = 25;  
  cout << "Local: d = " << d << endl; // d = 25 - локальна змінна перевизначає глобальну змінну  
  // 2. Вивести значення глобальної змінної d,  
  //   використовується операція :: - розширення області видимості  
  cout << "Global: d = " << ::d << endl;  
  // 3. Вивести подвоєне значення глобальної змінної d  
  cout << "Global: d\*2 = " << MultGlobal2() << endl; // d\*2 = 30  
  // 4. Записати в локальну змінну d нове значення 27  
  d = 27;  
  cout << "Local: d = " << d << endl;  
  // 5. Записати в глобальну змінну d нове значення 68  
  ::d = 68;  
  cout << "Global: d = " << ::d << endl; // глобальна змінна d = 68  
  // 6. Вивести подвоєне значення глобальної змінної ::d  
  cout << "Global d \* 2 = " << MultGlobal2() << endl; // буде виведено 136

return 0;

}// Функція, яка використовує значення глобальної змінної d.  
// До локальної змінної d у функції main() ця функція доступу не має  
int MultGlobal2(void)  
{  
  // тут доступна глобальна змінна d  
  return d \* 2;  
}

Результат виконання програми



При оголошенні змінної для неї виділяється пам’ять. Розмір пам’яті, що виділяється, залежить від типу змінної. Розмір виділеної пам’яті впливає на діапазон значень, які може приймати змінна.

**Приклад. Дослідження виконання інкременту.**

**#include <iostream>**

**#include <Windows.h>**

**using namespace std;**

**int main()**

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

**int i=0,j,k;**

**cout << "1 i++="<< i++ << endl;**

**i=0;**

**cout << "2 ++i="<< ++i<< endl;**

**i=0;**

**cout << "3 ++i="<< ++i << " i++="<< i++ << endl;**

**k=i=0;**

**j=k+++i++;**

**cout <<"4 j="<<j<<";k="<<k<<";i="<<i<<endl;**

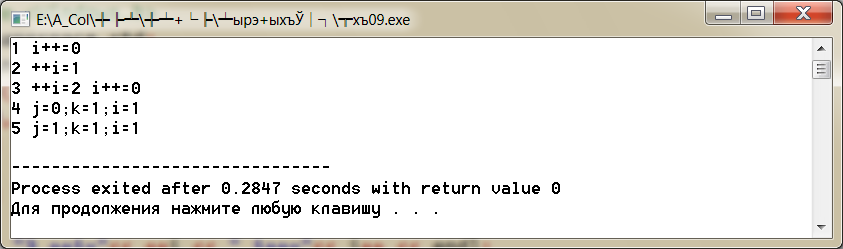
**k=i=0;**

**j=k+++(++i);**

**cout <<"5 j="<<j<<";k="<<k<<";i="<<i<<endl;**

**return 0;**

**}**



**Правило одного визначення** (**One Definition Rule, ODR**) — важлива концепція в мові програмування [C++](https://uk.wikipedia.org/wiki/C%2B%2B), що визначена в ISO C++ Standard(ISO/IEC 14882) 2003, в розділі 3.2.

Коротко ODR стверджує:

1. В окремій одиниці трансляції ([файлі](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB), якщо конкретна реалізація зберігає тексти програм у файлах, після обробки [препроцесором](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80)) [шаблон](https://uk.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), [клас](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [функція](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), [об'єкт](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), або перерахування можуть мати не більше одного визначення. Хоча деякі можуть мати яку завгодно кількість оголошень.
2. В програмі об'єкт або не[вставна функція](https://uk.wikipedia.org/w/index.php?title=%D0%92%D1%81%D1%82%D0%B0%D0%B2%D0%BD%D0%B0_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F&action=edit&redlink=1) не можуть мати більш ніж одне визначення. Якщо об'єкт чи невставна функція не використовуються, тоді достатньо оголошення без визначення. У випадку використання вони повинні мати рівно одне визначення.
3. Деякі сутності, наприклад, класи, шаблони або вставні функції можуть мати більш ніж одне визначення тільки якщо:
   1. вони містяться в різних одиницях трансляції;
   2. вони ідентичні [лексема](https://uk.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D1%81%D0%B5%D0%BC%D0%B0) за лексемою;
   3. значення лексем однакові в обох одиницях трансляції.

[Компілятори](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) не завжди знаходять порушення ODR. Багато з них виявляє [компонувальник](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D1%83%D0%B2%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA).

## Приклади

//file 1.c

struct S { int a; char b; };

void f(S\*);

//file 2.c

struct S { int a; char b; };

void f(S\*) { /\* ... \*/ };

Правило ODR каже, що цей приклад дозволений, і в обох файлах під S буде матися на увазі одна і та сама структура. Хоча це й невдалий спосіб описання структур, оскільки людина, яка супроводжує file2.c, може припустити, що визначення в її файлі єдине і вважатиме, що його можна змінювати, що може призвести до помилок, які буде складно знайти.

Метою ODR є реалізація можливості включення визначення класу в різні одиниці трансляції з одного [заголовного файлу](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B3%D0%BE%D0%BB%D0%BE%D0%B2%D0%BD%D0%B8%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB).

//file s.h

struct S { int a; char b; };

void f(S\*);

//file 1.c

#include "s.h"

// використання f()

//file 1.c

#include "s.h"

void f(S\*) { /\* ... \*/ };

Наведемо приклад трьох випадків порушення правила ODR:

//file 1.c

struct S1 { int a; char b; };

struct S1 { int a; char b; }; // помилка: повторне визначення

Структуру не можна двічі визначити в одному файлі.

//file 1.c

struct S2 { int a; char b; };

//file 1.c

struct S2 { int a; char bb; }; // помилка

Не ідентичні лексема за лексемою.

//file 1.c

typedef int X;

struct S3 { X a; char b; };

//file 2.c

typedef char X;

struct S3 { X a; char b; }; // помилка

Значення лексем різне в різних одиницях трансляції.

***Для самостійного вивчення*:** 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. В чому особливості функції main()?
3. Які особливості функції користувача?
4. Для чого призначені коментарі, які їх форми, які рекомендації по їх складанню?
5. Що таке макровизначення? Наведіть приклади.
6. Для чого використовується умовна компіляція?
7. Наведіть директиви умовної компіляції.
8. Що таке зовнішні оголошення?
9. Що може міститися в зовнішніх оголошеннях?
10. Надайте визначення функції.
11. Які операції порівняння ви знаєте та з якими операндами вони застосовуються?
12. Які логічні операції ви знаєте?
13. Як визначається тернарна операція?
14. Як реалізується введення-виведення значень виразів?
15. Як класифікуються змінні в залежності від місця оголошення в програмі?
16. Для чого використовується операція розширення області видимості **::?**
17. Які значення може містити умова, що перевіряється?
18. Як визначається результат операція **not?**
19. Як визначається результат операція **and?**
20. Як визначається результат операція **or?**
21. Поясніть, що представляє собою логічний вираз.

***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.