



***ЕЛЕМЕНТИ АЛГОРИТМІЧНИХ МОВ. ТИПИ ДАНИХ.  
СИСТЕМИ КОНТРОЛЮ ВЕРСІЙ***

# Загальна структура C++ програми

У загальному випадку програма мовою C++ має таку структуру:

1. *Коментар про призначення програми.*
2. *Область глобальних оголошень*
  - 2.1 Підключення бібліотек.
  - 2.2 Визначення препроцесора (констант, макросів тощо).
  - 2.3 Оголошення прототипів функцій.
  - 2.4. Оголошення глобальних змінних.
- 3 *Визначення головної функції.*
  - 3.1 Оголошення локальних змінних.
  - 3.2 Виклик функцій.
  - 3.3 Оператори.
- 4 *Визначення інших функцій.*
  - math.h – математичні функції та макроси
  - stdio.h – функції введення-виведення, а також типи та макроси
  - string.h – функції для роботи з рядками символів
  - time.h – типи і функції, пов'язані з датою та часом

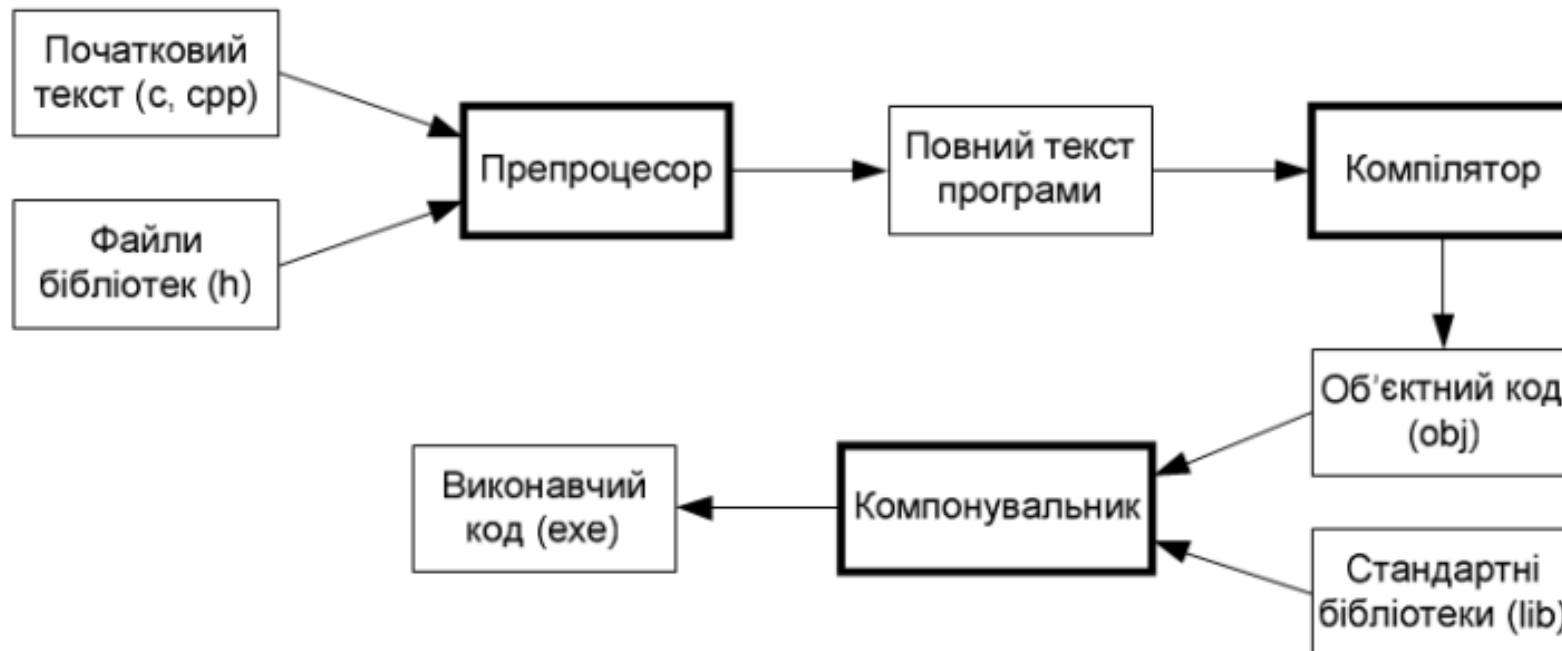
```
//функції введення - виведення, а також типи та макроси
#include <stdio.h>
//директива препроцесора включення файлу
#include <iostream>
//використання проостору імен
using namespace std;
//Визначення головної функції
int main() {
    //Оголошення локальних змінних
    int X, Y, rs;
    //зчитування даних з переходом на новий рядок
    cout << "Enter X "<< endl;
    cin >> X;
    cout << "Enter Y "<< endl;
    cin >> Y;
    //обчислення
    rs = ((120 - X - Y - Z) / 5);
    //виведення результату
    cout << "Amount of programs "<< rs;
    //завершення виконання функції,значення 0 -програма успішно завершила
    виконання
    return 0;
}
```

# Компіляція та виконання програми



три етапи опрацювання

- 1) *препроцесорне перетворення тексту*
- 2) *компіляція* тексту програми в об'єктний код;
- 3) *компонування* – зв'язування об'єктного коду з кодом функцій із стандартних бібліотек та генерування виконуваного коду



*Етапи опрацювання програми*

*Директиви препроцесора* – команди, які керують перетворенням тексту програми до її трансляції.

*# <ім'я\_директиви> <лексми\_препроцесора>*

- *#define* – визначення макроса чи препроцесорного ідентифікатора;
- *#include* – підключення тексту з файла;
- *#undef* – скасування визначення макросу чи препроцесорного ідентифікатора;
- *#if* – перевірка умови-виразу;
- *#ifdef* – перевірка визначеності ідентифікатора;
- *#ifndef* – перевірка невизначеності ідентифікатора;
- *#else* – початок альтернативної гілки для *#if*;
- *#endif* – завершення умовної директиви *#if*;
- *#elif* – складена директива *#else/#if*;
- *#line* – ім'я файла і бажаний початковий номер рядка;

## Коментарі

Коментарі використовуються лише для пояснень і жодних дій у програмі не виконують, тому що ігноруються компілятором.

Однорядкові коментарі позначаються символами `//` і записуються перед текстом коментаря:

`// Текст після цього символу і до кінця рядка є коментарем`

`// Це також однорядковий коментар`

Багаторядкові коментарі записуються між символами `/*` та `*/`.

`/* Коментар до програми може займати кілька  
рядків */`

## cin

```
int n; double m;  
// введення значень окремо  
cin >> n;  
cin >> m;  
// введення значень через пробіл  
cin >> n >> m;
```

## cout

```
//зчитування даних з переходом на новий рядок  
cout << "Enter X " << endl;  
  
rs = ((120 - X - Y - Z) / 5);  
//виведення результату  
cout << "Amount of programs " << rs;
```

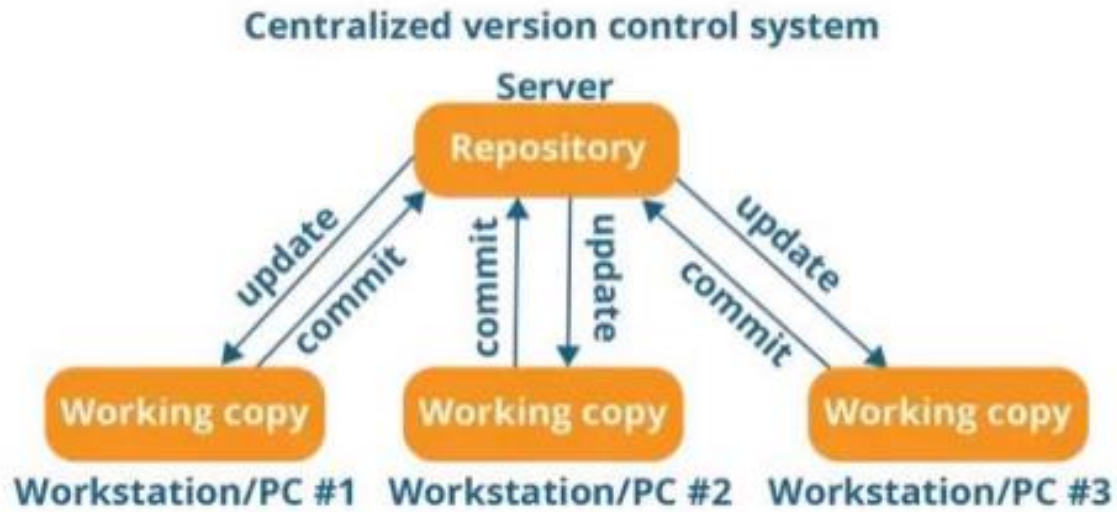
`#include <iomanip>`

- endl – забезпечує переведення курсору на новий рядок;
- setw(n) – задає ширину поля,
- setprecision(n) – задає точність значення,
- dec, hex, oct – задають системи числення

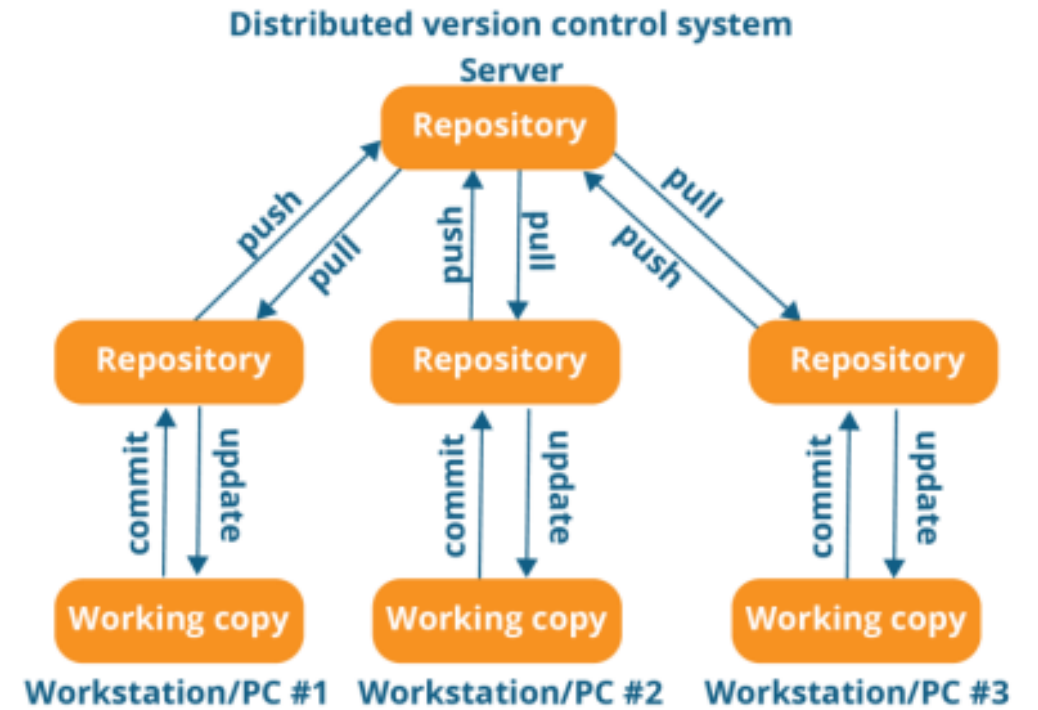


# Системи контролю версій

## Централізовані системи контролю версій



## Розподілені системи контролю версій



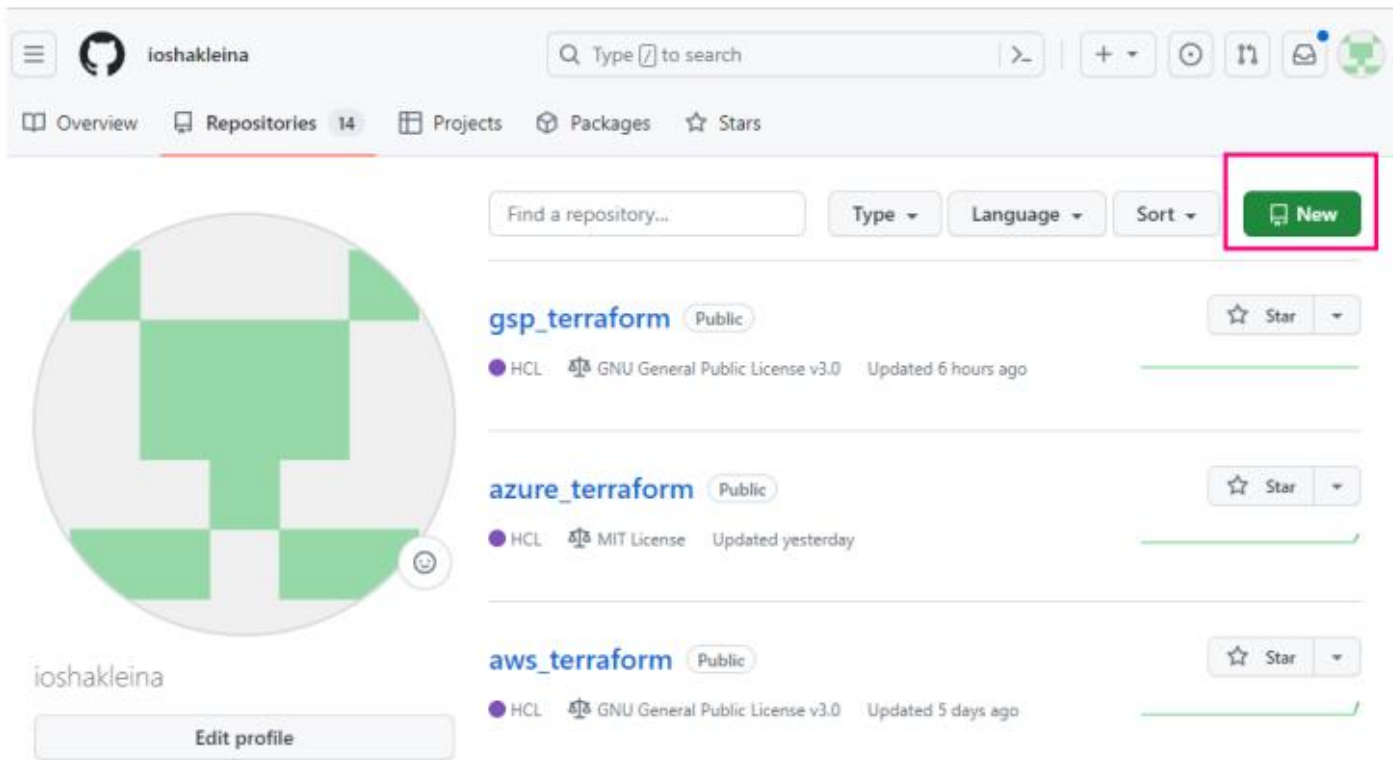


**Git** – найпопулярніша розподілена система контролю версій з відкритим вихідним кодом.

- *Система контролю:* Git відстежує і зберігає контент.
- *Система контролю версій:* код, який зберігається в Git може змінюватися і додаватися.
- *Розподілена система контролю версій:* в Git є віддалений репозиторій, який розташовується на сервері, і локальний репозиторій на машині кожного розробника.

### Три стани Git

- *Збережений у коміті* означає, що дані безпечно збережено в локальній базі даних.
- *Змінений* означає, що у файл внесено зміни, які ще не збережено в базі даних.
- *Індексований стан* виникає тоді, коли позначили змінений файл у поточній версії, щоб ці зміни ввійшли до наступного знімку коміту.



ioshackleina

Overview Repositories 14 Projects Packages Stars

Find a repository... Type Language Sort **New**

**gsp\_terraform** Public  
HCL GNU General Public License v3.0 Updated 6 hours ago

**azure\_terraform** Public  
HCL MIT License Updated yesterday

**aws\_terraform** Public  
HCL GNU General Public License v3.0 Updated 5 days ago

ioshackleina

Edit profile


## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?

[Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*

 ioshackleina /

✓ programming is available.

Great repository names are short and memorable. Need inspiration? How about [potential-octo-bassoon](#) ?

2. Ініціалізувати репозиторій – `$ git init`

3. Вибрати файли, які потрібно перенести:

`$ git add .`

4. Створити збереження (commit) і дати назву

`$ git commit -m "назва"`

5. Вказати власний репозиторій за допомогою команди

`$ git remote add origin <шлях_до_репозиторію>`

6. Відправити файл(и), вказавши гілку (branch)

`$ git push -u origin master`

Команди останніх двох пунктів можна скопіювати з GitHub:

...or push an existing repository from the command line

```
git remote add origin https://github.com/ioshackleina/programming.git
git branch -M main
git push -u origin main
```

# ***ОСНОВНІ ТИПИ ДАНИХ ВИРАЗИ ТА ОПЕРАЦІЇ. ПЕРЕТВОРЕННЯ ТИПІВ***

**Тип даних** – визначає:

- 1) множину значень, які може набувати певний екземпляр – змінна чи константа;
- 2) множину операцій, які можна виконувати над екземплярами цього типу;
- 3) внутрішнє представлення екземплярів цього типу в пам'яті комп'ютера.

Типи даних у мові C++ поділяються на *основні (базові)* та *структуровані*.

### *Базові типи*

- ✓ `Char` – символний
- ✓ `wchar_t` – символний двохбайтовий
- ✓ `int` – цілочисельний
- ✓ `float` – з плаваючою крапкою
- ✓ `double` – з плаваючою крапкою подвійної точності
- ✓ `bool` – логічний
- ✓ `void` – без значення

модифікатори:

`igned`

`unsigned`

`long`

`short`

Тип	Модифікатор			
	знаку		розміру	
	signed	unsigned	long	short
char	+	+		
wchar_t				
int	+	+	+	+
float				
double			+	
bool				
void				

$2.38e3$       позначає       $2.38 * 10^3$   
 $3.61e-4$       позначає       $3.61 * 10^{-4}$   
 $1e3$       позначає       $1 * 10^3$   
 $-2.7e-4$       позначає       $-2.7 * 10^{-4}$

Тип	Розмір в бітах	Діапазон значень
char	8	−127 ... 127
unsigned char	8	0 до 255
signed char	8	−127 ... 127
int	32	−2 147 483 647 ... 2 147 483 647
unsigned int	32	0 ... 4 294 967 295
signed int	32	те саме, що <code>int</code>
short int	16	−32 767 ... 32 767
unsigned short int	16	0 ... 65 535
signed short int	16	−32 767 ... 32 767
long int	32	те саме, що <code>int</code>
signed long int	32	те саме, що <code>signed int</code>
unsigned long int	32	те саме, що <code>unsigned int</code>
float	32	1,8E−38 ... 3,4E+38
double	64	2,2E−308 ... 1,8E+308
long double	64	2,2E−308 ... 1,8E+308
bool	—	ІСТИНА або ХИБНІСТЬ
wchar_t	16	0 ... 65 535

*Перелічувальний тип* – це тип даних, множиною значень якого є обмежений список ідентифікаторів – констант даного типу.

```
enum [<ім'я>] {<список_елементів>} [<список_змінних>];
```

Змінна перелічувального типу може отримувати значення однієї з іменованих констант списку. Наприклад, оголошення переліку днів тижня:

```
enum week { mon, tues, wed, thur, fri, sat, sun }; // тип перелік  
week day; // оголошено змінну типу перелік
```

Змінну `day` переліку `week` ще можна оголосити так:

```
enum week { sun, mon, tues, wed, thur, fri, sat } day;
```

Властивості даних типу `enum` аналогічні властивостям даних типу `int`.



# Константи



два способи визначення іменованих констант:

- за допомогою директиви препроцесора *#define*;
- за допомогою ключового слова *const*.

```
#define <ідентифікатор> <текст>
```

```
#define FALSE 0  
#define TRUE 1
```

```
#define N 10  
int N; /* ПОМИЛКА! */
```

```
#define N 10  
#define N 5 // макроім'я N буде 5
```

```
#define n 10 // за допомогою директиви #define  
const int n = 10; // за допомогою інструкції const
```

```
const <тип> <ім'я_константи> = <значення_константи>;
```

```
const int MONTH = 12;
```

```
const int MONTH; // значення не визначене  
MONTH = 12; // помилка: змінювати значення константи не можна
```

використання директиви *#define* у мові C/C++ є небажаним!



Операція	Пояснення
()	Дужки
++	Постфіксний інкремент
--	Постфіксний декремент
++	Префіксний інкремент
--	Префіксний декремент
+	Унарний +
-	Унарний –
!	Логічне заперечення
~	Бітове заперечення
*	Множення
/	Ділення
%	Остача від ділення націло
+	Додавання
-	Віднімання
<<	Розрядний зсув ліворуч
>>	Розрядний зсув праворуч
<	Перевірка чи менше
<=	Перевірка чи менше або дорівнює
>	Перевірка чи більше
>=	Перевірка чи більше або дорівнює
==	Перевірка чи дорівнює
!=	Перевірка чи не дорівнює
&	Розрядне «і»
^	Розрядне виключне «або»
	Розрядне «або»
&&	Логічне «і»
	Логічне «або»
?:	Умовна операція
=	Присвоєння

## Виклики функцій

Виклик функції = звертання до функції для обчислення її значення

Загальний синтаксис:

<ім'я функції> (<аргумент/аргументи>)

$\sin x$   $\longrightarrow$   $\sin(x)$

$\sin x^2$   $\longrightarrow$   $\sin(x * x)$

## Типові помилки запису виразів

$a + b / c + d$

$(a + b) / (c + d)$

Дужки

$2 a$

$2 * a$

Пропущений знак операції

$a + b +$

$a + b +$

$+ c + d$

$c + d$

При перенесенні виразу  
знак операції не  
повторюється

$1 / 2$

$1.0 / 2$

$1 / 2.0$

$1.0 / 2.0$

Цілочисельне ділення –  
в результаті буде 0

## *Операції префіксного та постфіксного інкременту та декременту*

```
int a = 1;
int b = ++a; // змінюємо a, потім – використовуємо нове значення
// a = 2, b = 2
int a = 1;
int b = a++; // використовуємо поточне значення, потім – змінюємо a
// a = 2, b = 1
```

# Змінні



**Змінна** — це ділянка оперативної пам'яті, яка має власне ім'я, тип і використовується для зберігання даних під час роботи програми.

*Перед використанням усі змінні повинні бути оголошеними.*

Змінні можуть бути оголошені у таких місцях програми:

- Всередині функції або блоку.
- Поза всіма функціями.

Оголошення змінних має такий формат:

`<тип_даних> *<ім'я_змінної>*[= <значення_змінної>];`

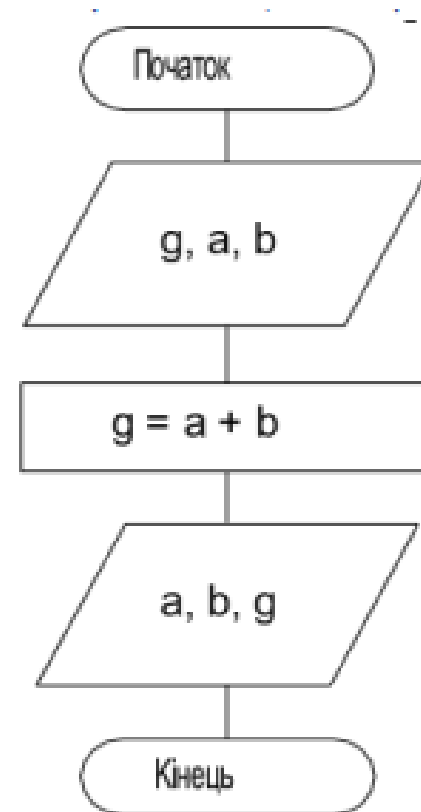
```
unsigned int a;  
int b;    // інтерпретується як signed int  
short c;  // інтерпретується як signed short  
unsigned d;    // інтерпретується як unsigned int  
int f, g = 4;  // змінна f - неініціалізована, g має значення 4  
int h(7), i = 2 * g, j = h + g; // h = 7, i = 8, j = 11  
int q = 057;   // ініціалізація вісімковим значенням 057  
int t = 0xBBB; // ініціалізація шістнадцятковим значенням 0xBBB
```

```
bool d = true, e = 5, f = -10, g = 0;  
// d=true, e=true, f=true, g=false  
bool rez2 = e < d; // rez2=false  
bool rez3 = f - 5; // rez3=true
```

```
char d = 'a';    // cx - ініціалізована символом 'a'  
char x, c = '\n'; //x - неініціалізована змінна, c - містить символ <Enter>  
char st[5] = "Lviv"; // масив із 5 символів зі значенням "Lviv"  
char s[] = "student"; // рядок ініціалізований значенням "student"
```

```
a = sizeof(int); // a = 4  
b = sizeof(long double); // b = 10
```

```
#include <iostream>  
using namespace std;  
  
int g; // оголошення глобальної змінної  
// оголошення функції main()  
int main() {  
    int a, b; // оголошення локальних змінних  
    a = 10;  
    b = 20;  
    g = a + b;  
    cout << "a=" << a << endl;  
    cout << "b=" << b << endl;  
    cout << "g=" << g;  
    return 0;  
}
```



*Використання глобальних змінних без потреби не рекомендується!*

# Перетворення типів



Перетворення типів даних  $\longrightarrow$  неявне та явне

```
int a = 5, b;  
double p = 3.14, c;  
c = a; // результат: c = 5.0  
b = p; // результат: b = 3
```

*Загальні правила перетворення типів:*

- 1) операнди типу `float` перетворюються до типу `double`;
- 2) якщо один операнд `long double`, то другий операнд перетворюється до цього ж типу;
- 3) якщо один операнд типу `double`, а другий – `float` чи ціле число, то другий операнд перетворюється до типу `double`;
- 4) операнди цілих типів `char` та `short` перетворюються до типу `int`;
- 5) цілі операнди типу `unsigned char` та `unsigned short` перетворюються до типу `unsigned int`;

```
unsigned char c;  
double f, d;  
unsigned long n;  
int i;  
d = f * (i + c / n);
```

Операцію *явного перетворення типів* використовують для уникнення помилок.

(<тип>) <змінна або вираз>;

```
int a = 2, b = 3;
double x = a / b; // результат: x=0.0
double y = (double)a/b; // результат: y=0.66(6)
y = a/(double)b; // результат: y=0.66(6)
y = (double)a/(double)b; // результат: y=0.66(6)
int z = 1, x = 3;
double y = z / (x + 25);
```

1) дописати десяткову крапку до числа 25:

```
y = z / (x + 25.0); // еквівалентно y = z/(x+25.)
```

2) домножити чисельник або знаменник ліворуч на 1.0:

```
y = 1.0 * z / (x + 25); // еквівалентно y = 1.*z/(x+25)
```

3) явно вказати тип результату:

```
y = (double)z / (x + 25);
```



## Логічні операції

A	B	!A	A    B	A&&B
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1

```
bool a, b, c = true, rez;  
a = !c; // результат: a=!true=false  
b = a; // результат: b=false  
b -= c; // результат: b=false-true=true  
rez = !(a < c); // результат: rez=!(false<true)=!true=false  
int z = 0, x, y, t=5, i;  
x = !z; // результат: x=!0=1  
y = (a == z) || (c && a); // результат: y=(0==0) || (1&&0)=1 || 0=1  
t -= c; // результат: t=5-1=4  
i = c - 2; // результат: i=1-2=-1
```



# Дякую за увагу

Лектор:  
кандидат фіз.-мат. наук, доцент  
Шаклеїна Ірина  
[iryna.o.shakleina@lpnu.ua](mailto:iryna.o.shakleina@lpnu.ua)  
кафедра ІСМ, ІКНІ