ОСНОВНІ ТИПИ ДАНИХ ВИРАЗИ ТА ОПЕРАЦІЇ. ПЕРЕТВОРЕННЯ ТИПІВ

Основні типи даних

Основне призначення будь-якої програми полягає в опрацюванні даних. Дані різного типу зберігаються і опрацьовуються по-різному. У будь-якій алгоритмічній мові кожна константа, змінна, результат обчислення виразу або функції повинні мати певний тип.

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 С++ поділяються на *основні (базові)* та *структуровані.*

Базові типи

В С++ визначено сім основних типів даних

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

В С++ перед типами char, int і double можна записувати *модифікатори*. Модифікатор використовується для уточнення базового типу та зміни його значення. Можливі модифікатори:

igned
unsigned
long
short

Модифікатори signed, unsigned, long та short можна застосовувати до цілочисельних типів. Крім того, модифікатори signed та unsigned можна використовувати з типом char, а модифікатор long – з типом double. Модифікатори в парах signed / unsigned та long / short – взаємовиключні (з кожної такої пари можна використовувати лише один):

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

Специфікатор unsigned використовують при роботі з додатними числами, а специфікатор signed – з додатними та від'ємними. За замовчуванням призначається знаковий тип, тому специфікатор signed вказувати *необов'язково*.

Для відокремлення *цілої частини числа від дробової* використовується десяткова крапка. Крім звичайної форми, дійсні константи можна записувати у формі з плаваючою крапкою. Число перед символом “e” називається **мантисою**, а після нього – **порядком**, тобто замість основи 10 використовується літера “e”, після якої ставиться показник степеня.

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

Типові розміри в бітах та діапазони представлення для кожного C++-типу даних в 32- розрядному середовищі:

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 int
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 int
signed long int	32	те саме, що signed int
unsigned long int	32	те саме, що unsigned int
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

Тип char використовується для подання символа, а значенням змінної цього типу є певний код символа. Для визначення десяткових кодів символів кирилиці слід використовувати тип unsigned char. Крім того, тип char використовується для оголошення рядків типу char*.

Змінна *логічного* типу bool може отримувати значення true (істина) та false (хибність). Змінні цього типу переважно використовуються для індикації стану чи побудови логічних виразів. У мові C++ змінним логічного типу можна присвоювати цілочисельні значення – 0 сприймається як false, а будь-яке ненульове значення – як true.

Тип void використовується для визначення типу функцій, які не повертають жодного результату, а всі дії та обчислення виконуються всередині цих функцій. Цей тип використовують для подання порожнього списку аргументів функції як базовий тип вказівників, а також в операціях перетворення типів.

Крім оголошень змінних різних типів, можна оголошувати *власні типи.* Таке оголошення можна реалізувати наступним чином:

- для оголошення типу використати ключове слово typedef.
- при оголошенні структури, об'єднання чи переліку вказати ім'я тега, а потім використати це ім'я при оголошенні змінних та функцій, як посилання на цей тег.

При оголошенні за допомогою ключового слова typedef створюється *новий*

тип даних, який надалі може бути використаний у межах видимості цього типу для оголошення змінних.

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

Переліки (enum) використовуються для вибору та призначання об'єктів з деякої множини значень.

Оголошення переліків визначає відповідний тип змінної та список логічно зв'язаних іменованих констант, значеннями яких є певні цілі числа. Оголошення переліку має такий формат:

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

Компілятор забезпечує, щоб змінні перелічувального типу отримували значення лише зі списку констант. Якщо ім'я перелічуваного типу не вказане, то визначається анонімний перелік.

Список елементів – це список ідентифікаторів (нумераторів, елементів переліку), відокремлених комами.

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

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

Константи mon, tues, wed, thur, fri, sat, sun є цілими числами, перша з яких (mon) за замовчуванням отримує значення 0, а кожна наступна – значення на одиницю більше за значення попереднього елемента (`tues = 1`, `wed = 2`, ..., `sun = 6`).

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`. Змінні типу `enum` можуть використовуватись в індексних виразах, арифметичних операціях та операціях порівняння як операнди. При виконанні арифметичних операцій переліки перетворюються у цілі числа.

Константи

Константа – це стала величина, значення якої не змінюється під час виконання програми.

Існує два способи визначення іменованих констант:

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

1) Директива #define використовується дл заміни констант, які часто використовуються, ключових слів, операторів чи виразів на певні ідентифікатори. Ідентифікатори, які заміняють текстові чи числові константи, називають *іменованими константами*. Ідентифікатори, які заміняють фрагменти програм, називають

макровизначеннями, причому макровизначення можуть мати аргументи. Директива *#define* має дві синтаксичні форми:

```
#define <ідентифікатор> <текст>

#define <ідентифікатор> <(список_параметрів)> <текст>
```

Директива #define заміняє всі наступні входження ідентифікатора текстом. Такий процес називається *макропідстановкою*. Текст може бути будь-яким фрагментом програми на мові С++, а також може бути відсутнім. В останньому випадку всі екземпляри ідентифікатора вилучаються з програми. *Заміна виконується до моменту компіляції програми*. Якщо ідентифікатор є частиною іншого ідентифікатора або входить у рядок символів чи коментар, то його заміна не відбувається

Найбільш часто директива *#define* використовується для визначення символьних констант, наприклад:

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

Після цього в усіх виразах програми замість чисел 0 та 1 можна використати їх символьні імена відповідно *FALSE* та *TRUE*.

Визначені символьні константи не повинні оголошуватись у тексті програми. Наприклад, наступне перевизначення константи призведе допомилки:

```
#define N 10
int N; /* ПОМИЛКА! Макроім'я N не може бути оголошене як змінна */
```

Символьна константа може бути перевизначена іншою символьною константою за допомогою директиви *#define*, наприклад:

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

2) Формат оголошення констант за допомогою специфікатора const такий:

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

Компілятор виділяє комірку пам’яті, записує в неї вказане значення та помічає цю комірку як заблоковану для запису (тобто, можна лише зчитувати значення, яке зберігається в цій комірці). Наприклад:

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

При оголошенні константам **обов'язково потрібно задавати певні значення**, які пізніше змінювати не можна. Значення константі обов’язково слід присвоювати в тій самій команді, в якій вона визначається; наступний код – неправильний:

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

Нижче наведено оголошення константи n зі значенням 10 двома способами

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

Використання кваліфікатора `const` для визначення констант – ***кращий спосіб,*** ніж використання макросу #define, тому що

1) явно вказується тип константи;

2) можна обмежити область дії (видимість) константи тим блоком, в якому вона визначається;

3) кваліфікатор const можна застосовувати і до складних структур даних (наприклад, – масивів, структур, об'єднань)

Отже, використання директиви #define у мові C/C++ без особливої на те потреби є небажаним!

Розрізняють чотири типи констант: цілі, дійсні, символьні константи та рядки. Значення константи може бути подане як певне числове чи символьне значення, так і константний вираз, який містить раніше оголошені константи та змінні. Для цілих констант тип можна не вказувати, оскільки за замовчуванням їм присвоюється тип *int*.

Виклики функцій

Виклик функції = звертання до функції для обчислення її значення При цьому слід вказати аргумент функції. Загальний синтаксис:

```
<ім'я функції> (<аргумент/аргументи>)
```

$\sin x$ ⟶ `sin(x)`

$\sin x^2$ ⟶ `sin(x * x)`

Вирази та операції

Вираз – це записане мовою програмування правило для обчислення деякого значення. Вираз складається з *даних* (операндів) та *дій над даними* (операцій). Операндами можуть бути константи, змінні, результати функцій.

Порядок виконання дій при обчисленні виразу визначається *пріоритетами операцій*: насамперед виконуються операції з вищим пріоритетом. Частина виразу, записана в круглих дужках, розглядається як окремий операнд. Іншими словами, дужки змінюють пріоритет операцій: в першу чергу обчислюється частина виразу, записана в дужках.

Пріоритети операцій

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

	Арифметичні операції присвоєння:
+=	Додати і присвоїти
-=	Відняти і присвоїти
*=	Помножити і присвоїти
/=	Поділити і присвоїти
%=	Обчислити остачу від ділення націло і присвоїти
&=	Обчислити розрядне «і» та присвоїти
^=	Обчислити розрядне виключне «або» і присвоїти
|=	Обчислити розрядне «або» і присвоїти
<<=	Зсунути ліворуч і присвоїти
>>=	Зсунути праворуч і присвоїти

Типові помилки запису виразів

`a + b / c + d`	`(a + b) / ( c + d )`	Дужки
`2 a`	`2 * a`	Пропущений знак операції
`a + b +` `+ c + d`	`a + b +` `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

Дії комп'ютера при їх виконанні

A = A + 1; → A++; або ++A;

A = A - 1; → A--; або --A;

```
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.

При оголошенні змінної для неї резервується певна ділянка пам'яті, розмір якої залежить від типу змінної.

Значення змінної – це фактичне значення, що міститься у виділеній для змінної ділянці пам'яті.

Ім'я змінної може складатись із літер латиниці (a...z,A...Z), цифр (0...9) та символа підкреслення ("_"), але обов'язково має починатись з літери чи символа підкреслення. Мова C++ є чутливою до регістра, тобто відповідні великі та малі літери вважаються різними символами.

При виборі імені змінної потрібно дотримуватись таких *вимог*:

1) ідентифікатор не повинен співпадати з ключовими словами, зарезервованими словами та назвами функцій бібліотек компілятора;
2) символ підкреслення ("_") як перший символ імені слід використовувати обережно, бо імена можуть співпасти з іменами системних функцій та змінних, а також можуть виникнути проблеми з їх використанням на комп'ютерах інших типів.

Перед використанням усі змінні повинні бути оголошеними.

Змінні можуть бути оголошені *у таких місцях програми*:

- Всередині функції або блоку.
- Поза всіма функціями.

- У визначенні параметрів функції.

Оголошення змінних має такий формат:

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

Оголошення змінних цілого типу.

```
unsigned int a;
int b;   // інтерпретуеться як signed int
short c; // інтерпретуеться як signed short
unsigned d;    // інтерпретуеться як unsigned int d
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 шістнацятковим значенням 0xBBB
```

Оголошення логічних змінних.

```
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
```

Оголошення символьних та рядкових констант

```
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 знаценням "Lviv"
char s[] = "student"; // рядок ініціалізований знаценням "student"
```

Для вирначення рорміру пам'яті, виділеної для змінної існує операція sizeof(), яка повертає значення довжини заданого типу. Наприклад

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

Змінні, оголошені всередині функції або блоку, називаються ***локальними.*** Областю видимості таких змінних є функція, в якій вони оголошені. Для решти функцій програми ці змінні та їхні значення є невідомими.

Якщо локальній змінній потрібно забезпечити довгу тривалість життя, то використовується слово *static*. *Такі змінні є визначеними протягом всього виуконання програми але залишаються локальними.*

Важливо пам'ятати, що після оголошення всі локальні змінні обов'язково *потрібно ініціалізувати*, інакше їх значення будуть *невизначеними* (випадковими).

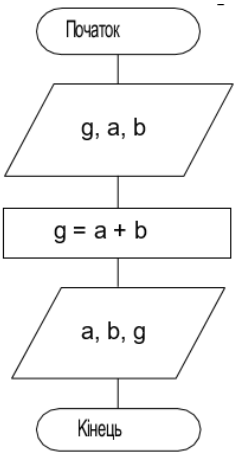
Глобальні змінні переважно визначаються у верхній частині програми перед усіма визначеннями функцій. *Областю видимості глобальних змінних є всі функції у цьому ж файлі, що стоять після оголошення таких змінних, а часом їхнього життя – час виконання програми.* Після оголошення глобальні змінні ініціалізовуються автоматично

Слід намагатися уникати використання глобальних змінних, бо їх використання часто призводить до помилок!

Нижче наведено приклад викортстання глобальної та локальних змінних.

```
#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

У мові С++ допускаються операції над різними типами даних. Якщо у вираз входять дані різних типів, то відбувається їх перетворення до певного типу. Перетворення типів даних може ути **неявне** та **явне.**

При ***неявному*** (автоматичному) перетворенні типів значення з меншим числом двійкових розрядів перетворюється до значення з більшим числом розрядів згідно наступної схеми вкладеності типів в порядку збільшення розміру пам'яті:

`char < short < int < long < float < double < long double`

При виконанні операції присвоєння тип виразу в правій частині перетворюється до типу виразу в лівій частині. Перетворення даних меншого типу до більшого виконується правильно. Перетворення значення більшого типу до значення меншого типу *може викликати втрату даних* (відкидаються старші байти).

```
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 дотипу `unsigned int`;
6) якщо один операнд типу `unsigned long`, то другий цілий операндперетворюється

до типу `unsigned long`;

7) якщо один операнд типу `long`, то другий операнд перетворюється до типу `long`;
8) якщо один операнд типу `unsigned int`, то другий цілий операнд перетворюється до цього ж типу.

Таким чином при обчисленні виразів операнди перетворюються до типутого операнда, який має більший розмір.

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

Порядок перетворення типів у прикладі 3. такий:

1) тип змінної `c` перетворюється до `unsigned int` (правило 5);
2) тип змінної `c` перетворюється до `unsigned long` (правило 6);
3) тип `c/n` перетворюється до `unsigned long` (правило 6);
4) тип `(i+c/n)` перетворюється до `unsigned long` (правило 6);
5) тип виразу перетворюється до `double` (правило 3).

Операцію ***явного перетворення типів*** використовують для уникнення, розглянутих у попередньому пункті, помилок.

При явному перетворенні типів перед змінною або виразом у круглихдужках вказується ім'я типу, до якого слід перетворити їх значення:

```
(<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);
```

У прикладі змінні `z` та `x` є цілого типу, тому після обчислення виразу дробова частина втрачається. Щоб уникнути цього потрібно перетворити чисельник або знаменник до дійсного типу одним з таких способів:

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);
```

Явне перетворення типу є джерелом можливих помилок, оскільки вся відповідальність за його результат покладається на програміста. Операцію явного

перетворення типів слід використовувати обережно.

Операції відношень та логічні операції

Операції відношень (<, >, <=, >=, ==, !=) використовуються для по- рівняння двох операндів. Результат цих операцій для значень типу `bool` є `false` або `true`, а для значень типу `int` є 0 або 1.

Логічна операція – це дія, яка виконується над логічними змінними, результатом якої є `false` або `true`. Логічні операції обчислюють кожний операнд з огляду на його еквівалентність нулю.

Якщо значення першого операнда вистачає, щоб визначити результат операції, то другий операнд не обчислюється.

Правила опрацювання логічних операцій:

1) якщо перший операнд операції логічного додавання (операція "АБО" (дизʼюнкція) позначається `||`) має ненульове значення, то другий операнд не обчислюється;
2) якщо значення першого операнда операції логічного множення (операція "І" (кон'юнкція) позначається `&&`) дорівнює 0, то другий операнд не обчислюється;
3) операція логічного заперечення (операція "НЕ" (інверсія) позначається `!`) виконується над однією логічною змінною.

Логічні вирази будуються з логічних змінних, логічних операцій та дужок

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

Операнди логічних виразів обчислюються зліва направо. Наприклад

```
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
```