ЛЕКЦІЯ № 5-6

ФУНКЦІЇ В С++

ТИПИ ДАНИХ

Назва	Опис	Діапазон	Обсяг пам'яті
Дійсні типи			
long double		$-3.4*10^{-4932}$ ... $3.4*10^{4932}$	10 байтів
double	Число з плаваючою комою подвійної точності	$-1.7*10^{-308}$ ... $1.7*10^{308}$	8 байтів
float	Число з плаваючою комою одинарної точності	$-3.4*10^{-38}$ ... $3.4*10^{38}$	4 байти

Назва	Опис	Діапазон	Обсяг пам'яті
Цілі типи			
unsigned long int (unsigned long)	Довге ціле	0 … 4 294 967 295	4 байти
long int (long)	Довге ціле	-2 147 483 648 … 2 147 483 647	4 байти
unsigned int (unsigned)	Ціле	0 ... 4 294 967 295	4 байти
int	Ціле	-2 147 483 648 … 2 147 483 647	4 байти
unsigned short int (unsigned short)	Коротке ціле	0 … 65 535	2 байти
short int (short)	Коротке ціле	-32 768 … 32767	2 байти
Символьний тип			
unsigned char	Символ або мале ціле	0 … 255	1 байт
char	Символ або мале ціле	-128 … 127	1 байт
Булевий тип			
bool	Булева змінна	false, true	1 байт

КОНСТАНТИ

- const int a=5;
- const float pi = 3.14l5926;
- const char symbol='t';
- const char symbol='\t';

ПРИКЛАД

```
main()
{
    const int x = 7; // присваивание начального значения
                     // именованной константе
    cout << "Значение именованной константы x равно: "
         << x << '\n';
    return 0;
}
```

```
Значение именованной константы x равно: 7
```

ПРИКЛАД

```
//Постоянный объект должен получить начальное значение
main()
{
   const int x;    // Ошибка: x должен получить
                   // начальное значение
   x = 7;          // Ошибка: постоянный объект (именованную
                   // константу) изменять нельзя
   return 0;
}
```

```
Compiling FIG4_6.CPP:
Error FEG4_6.CPP 4: Constant variable 'x' must be
   initialized
Error FEG4_6.CPP 6: Cannot modify a constant object
```

МАТЕМАТИЧНІ БІБЛІОТЕЧНІ ФУНКЦІЇ

```
#include <cmath>
```

Функція	Опис	Приклад
sqrt(x)	Корінь квадратний з x	sqrt(900.0)=30.0 sqrt(9.0)=3.0
exp(x)	Експоненційна функція e^x	exp(1.0)=2.718282 exp(2.0)=7.389056
log(x)	Логарифм натуральний x (за основою e)	log(2.718282)=1.0 log(7.389056)=2.0
log10(x)	Логарифм десятковий x (за основою 10)	log10(1.0)=0.0 log10(10.0)=1.0 log10(100.0)=2.0
fabs(x)	Абсолютне значення x (|x|)	якщо x>0, то fabs(x)=x якщо x=0, то fabs(x)=0.0 якщо x<0, то fabs(x)=-x

МАТЕМАТИЧНІ БІБЛІОТЕЧНІ ФУНКЦІЇ

```
#include <cmath>
```

Функція	Опис	Приклад
ceil(x)	Заокруглення x до найменшого цілого, не меншого ніж x	ceil(9.2)=10.0 ceil(-9.8)=-9.0
floor(x)	Заокруглення x до найбільшого цілого, не більше ніж x	floor(9.2)=9.0 floor(-9.8)=-10.0
pow(x, y)	x в степені y (x^y)	pow(2, 7)=128.0 pow(9, 0.5)=3.0
sin(x)	сінус x (x в радіанах)	sin(0.0)=0.0
cos(x)	косінус x (x в радіанах)	cos(0.0)=1.0
tan(x)	тангенс x (x в радіанах)	tan(0.0)=0.0

```
int square(int);  // прототип функции

main()
{
   for (int x = 1; x <= 10; x++)
      cout << square(x) << "  ";

      cout << endl;

   return 0;
}

// описание функции
int square(int y)
{
   return y * y;
}
```

```
 1   4   9   16   25   36   49   64   81   100
```

ОПИС ФУНКЦІЇ

тип_повертаємого_значення імя_функції (список параметрів)
{
  оголошення змінних та констант;
  оператори;
}

```
// описание функции
int square(int y)
{
    return y * y;
}
```

void – функція не повертає ніякого значення

Якщо функція не отримує ніяких параметрів, список параметрів задається як void або пусті ()

Аргументи (float x,y) – помилка

Аргументи(float x, float y) - вірно

ПОВЕРНЕННЯ ДО ТОЧКИ ВИКЛИКУ ФУНКЦІЇ

- return;
- функція не повертає результат

- { … } // тіло функції без слова return
- функція не повертає результат

- return вираз;
- функція повертає результат – значення виразу

ЗНАХОДЖЕННЯ МАКСИМАЛЬНОГО З ТРЬОХ ЧИСЕЛ

```
int maximum(int, int, int); // прототип функции

main()
{
   int a, b, c;

   cout << "Введите три целых числа: ";
   cin >> a >> b >> c;
   cout << "Максимум равен " << maximum(a, b, c) << endl;

   return 0;
}

// Определение функции maximum
int maximum(int x, int y, int z)
{
   int max = x;

   if (y > max)
      max = y;
   if (z > max)
      max = z;

   return max;
}
```

```
Введите три целых числа: 22  85  17
Максимум равен 85

Введите три целых числа: 92  35  14
Максимум равен: 92

Введите три целых числа: 45  19  98
Максимум равен: 98
```

ЗАГОЛОВОЧНІ ФАЙЛИ

- Кожна стандратна бібіліотека має відповідний заголовочний файл, який містить прототипи всіх функцій бібліотеки та оголошення різних типів даних і констант, які використовуються цими функціями.
- Заголовочні файли, які закінчують на .h – заголовочні файли старого стилю, які витиснені заголовочними файлами стандартної бібліотеки C++

ЗАГОЛОВОЧНІ ФАЙЛИ

Заголовочний файл стандартної бібліотеки	Пояснення
<cctype>	Містить прототипи бібліотечних функцій для обробки смволів. Замінив заголовочний файл <ctype.h>
<cmath>	Містить прототипи математичних бібліотечних функцій. Замінив заголовочний файл <math.h>
<cstdio>	Містить прототипи функцій вводу, виводу. Замінив - <stdio.h>
<cstdlib>	Містить прототипи функцій для перетворення чисел в текст, текста в число, для виділення памяті, генерації випадкових чисел. Замінив - <stdlib.h>
<ctime>	Містить прототипи функцій та типи для роботи з часом і датами. Замінив - <time.h>

ЗАГОЛОВОЧНІ ФАЙЛИ

Заголовочний файл стандартної бібліотеки	Пояснення
<iostream>	Містить прототипи функцій для функцій стандартного введення та виведення. Замінив - <iostream.h>
<iomanip>	Містить прототипи функцій для операцій з потоками, яка дають можливість форматування потоків даних. Замінив - <iomanip.h>

ЗАГОЛОВОЧНІ ФАЙЛИ

- Програміст може сам створювати потрібні йому заголовочні файли.
- Заголовочні файли, створені програмістом, повинні мати розширення .h.
- Включаються ці файли за допомогою директиви #include
- Наприклад, файл square.h підключається до програми як

#include “square.h”

ГЕНЕРАЦІЯ ВИПАДКОВИХ ЧИСЕЛ

- #include < cstdlib>
- rand() – генерує випадкове число від 0 до RAND_MAX (константи, визначеної у <cstdlib>)

$$0 \leq \text{rand}() \leq \text{RAND\_MAX}$$

- RAND_MAX=32 767
- rand() % 6 - випадкове число від 0 до 5
- 1+rand() % 6 - випадкове число від 1 до 6

ПРИКЛАД

- #include<iostream>
- using std::cout; using std::endl;
- #include<iomanip>
- using std::setw;
- #include<cstdlib>

```
5    5    3    5    5
2    4    2    5    5
5    3    2    2    1
5    1    4    6    4
```

```
main()
{
    for (int i = 1; i <= 20; i++)
    {
        cout << setw(10) << 1 + rand() % 6;

        if  (i % 5 == 0)
            cout << endl;
    }
      return 0;
}
```

- #include<iostream>
- using std::cout;
- using std::endl;
- #include<iomanip>
- using std::setw;
- #include<cstdlib>

```
main()
{
   int   frequency1 = 0, frequency2 = 0,
         frequency3 = 0, frequency4 = 0,
         frequency5 = 0, frequency6 = 0;

   for (int roll = 1; roll <= 6000; roll++) {
      int face = 1 + rand() % 6;

      switch (face)  {
         case 1:
            ++frequency1;
            break;
         case 2:
            ++frequency2;
            break;
         case 3:
            ++frequency3;
            break;
         case 4:
            ++frequency4;
            break;
         case 5:
            ++frequency5;
            break;
         case 6:
            ++frequency6;
            break;
      }
   }
```

```
    cout  << "Грань" << setw(13) << "Частота"
          << endl << "     1" << setw(13) << frequency1
          << endl << "     2" << setw(13) << frequency2
          << endl << "     3" << setw(13) << frequency3
          << endl << "     4" << setw(13) << frequency4
          << endl << "     5" << setw(13) << frequency5
          << endl << "     6" << setw(13) << frequency6 << endl;

    return 0;
}
```

Грань	Частота
1	987
2	984
3	1029
4	974
5	1004
6	1022

ГЕНЕРАЦІЯ ВИПАДКОВИХ ЧИСЕЛ

- #include < cstdlib>
- rand() – генерує псевдовипадкові числа
- **srand (unsigned int x)** – рандомізація (послідовність чисел, утворених функцією rand(), завжди буде різною)

```
srand(time(NULL));
```

- #include < ctime>
- time(NULL) – повертає поточний час в секундах

ПРИКЛАД

- #include<iostream>
- using std::cout; using std::endl;
- #include<iomanip>
- using std::setw;
- #include<cstdlib>

```
main()
{

   unsigned seed;

   cout << "Введите число: ";
   cin >> seed;
   srand(seed);

   for (int i = 1; i <= 10; i++) {
      cout << setw(10) << 1 + rand() % 6;

      if (i % 5 == 0)
         cout << endl;
   }
   return 0;
}
```

```
Введите число: 67
         1         6         5         1         4
         5         6         3         1         2

Введите число: 432
         4         2         6         4         3
         2         5         1         4         4

Введите число: 67
         1         6         5         1         4
         5         6         3         1         2
```

ФУНКЦІЇ БЕЗ ПАРАМЕТРІВ

- Пустий список параметрів – або void або пусті ()

- void print (void);
- void print();

ПРИКЛАД

```
void f1();
void f2(void);

main()
{
    f1();
    f2();

    return 0;
}

void f1()
{
    cout << "Функция f1 не требует аргументов" << endl;
}
void f2(void)
{
    cout << "Функция f2 также не требует аргументов" << endl;
}
```

Функция f1 не требует аргументов
Функция f2 также не требует аргументов

ФУНКЦІЇ, ЩО ВБУДОВУЮТЬСЯ

- В C++ для зниження витрат пам'яті на виклик функцій (особливо маленьких функцій) передбачені функції, що вбудовуються, (**inline-функції**)
- Специфікація **inline** говорить компілятору створити копію коду функції у відповідному місці, для того, щоб уникнути виклику цієї функції.
- Компілятор може ігнорувати специфікацію **inline**, що зазвичай і робить для всіх функцій крім самих маленьких.

ПРИКЛАД

```
inline float cube(const float s) { return s * s * s; }

main()
{
    cout << "Введите длину стороны вашего куба: ";

    float side;

    cin >> side;
    cout << "Объем куба со стороной  "
         << side << " равен " << cube(side) << endl;

    return 0;
}
```

```
Введите длину стороны вашего куба: 3.5
Объем куба со стороной 3.5 равен 42.875
```

const говорить компілятору, що функція не змінює параметр s

ПОСИЛАННЯ ТА ПАРАМЕТРИ-ПОСИЛАННЯ

В C++ існує два способи звертання до функції:

- виклик за значенням;
- виклик за посиланням.

Виклик за значенням – створюється копія аргументу, копія передається функції, що викликається. Зміна копії не впливає на значення оригіналу. Недолік – збільшення витрат пам'яті та часу роботи програми.

Виклик за посиланням - за допомогою параметрів-посилань. Функція безпосередньо звертається до даних та має можливість їх змінити. Перевага – зменшення витрат пам'яті, пришвидшення роботи програми.

ПАРАМЕТРИ-ПОСИЛАННЯ

- Параметр-посилання – псевдонім відповідного аргументу. В прототипі функції для таких параметрів ставить &.
- int &count

```
int squareByValue(int);
void squareByReference(int &);

main ( )
{
   int x = 2, z = 4;

   cout << "x = " << x << " перед squareByValue" << endl
        << "Значение, возвращенное squareByValue: "
        << squareByValue(x) << endl
        << "x = " << x << " после squareByValue" << endl << endl;

    cout << "z = " << z << " перед squareByReference" << endl;
    squareByReference(z);
    cout << "z = " << z << " после squareByReference" << endl;

    return 0;
}
```

```
x = 2 перед squareByValue
Значение, возвращенное squareByValue: 4
x = 2 после squareByValue

z = 4 перед squareByReference
z = 16 после squareByReference
```

```
int squareByValue(int a)
{
   return a *= a;      //аргумент оператора вызова не изменяется
}

void squareByReference(int &cRef)
{
   cRef *= cRef;      // аргумент оператора вызова изменяется
}
```

ПАРАМЕТРИ-ПОСИЛАННЯ

- int &x, &y, &z; - вірно!
- int &x, y, z; - не вірно!
- int& x, y, z; - не вірно!

- const - посилання є константою

ПОСИЛАННЯ

- Посилання можна використовувати як псевдоніми для інших змінних всередині функції (хоча для цього є мало підстав)

```
int count = 1;          // объявление целой переменной count
int &cRef = count;      // создание cRef как псевдонима для count
++cRef;                 // приращение count (используется псевдоним)
```

```
main()
{
    int x = 3, &y; //Ошибка: y должна получить начальное значение
    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "x = " << x << endl << "y = " << y << endl;

    return 0;
}
```

```
Compiling FIG3_21.CPP:
Error FIG3_21.CPP 6: Reference variable 'y' must be initialized
```

Змінні, які є посиланнями, повинні отримувати початкові значення при їх оголошенні.

```
main()
{
    int x = 3, &y = x;  // y теперь является псевдонимом x

    cout << "x = " << x << endl << "y = " << y << endl;
    y = 7;
    cout << "x = " << x << endl << "y = " << y << endl;

    return 0;
}
```

```
x = 3
y = 3
x = 7
y = 7
```

АРГУМЕНТИ ПО ЗАМОВЧЕННЮ

- Зазвичай при виклику функції до неї передається конкретне значення кожного аргументу.
- Але програміст може вказати, що аргумент є аргументом по замовченню і приписати цьому аргументу значення по замовченню.
- Тоді, якщо аргумент по замовченню не вказано при виклику функції, то в цей виклик автоматично передається значення по замовченню.
- Аргументи по замовченню повинні бути самим правими (останніми) аргументами в списку параметрів функції!

АРГУМЕНТИ ПО ЗАМОВЧЕННЮ

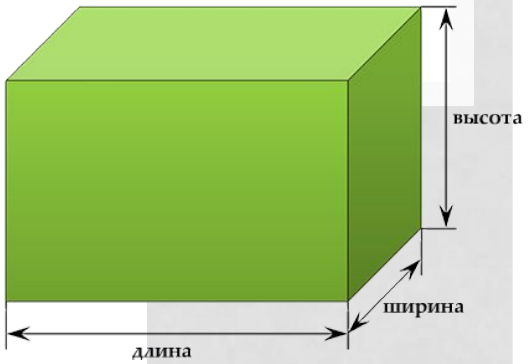
- Аргументи по замовченню повинні бути вказані при першому згадуванні імені функції – зазвичай у прототипі.
- Аргументи по замовченню можна використовувати з функціями inline.
- Якщо викликається функція з двома і більше аргументами по замовченню і якщо пропущений аргумент не є самим правим у списку параметрів, то всі аргументи справа від пропущеного також пропускаються.

АРГУМЕНТИ ПО ЗАМОВЧЕННЮ

```
// Расчет объема параллелепипеда
inline int boxVolume(int length = 1, int width = 1,
                     int height = 1)
   { return length * width * height; }

main()
{
   cout << " Объем параллелепипеда по умолчанию равен: "
        << boxVolume() << endl << endl
        << " Объем параллелепипеда с длиной 10," << endl
        << " шириной 1 и высотой 1 равен: "
        << boxVolume(10) << endl << endl
        << " Объем параллелепипеда с длиной 10," << endl
        << " шириной 5 и высотой 1 равен: "
        << boxVolume(10, 5) << endl << endl
        << " Объем параллелепипеда с длиной 10," << endl
        << " шириной 5 и высотой 2 равен: "
        << boxVolume(10, 5, 2)
        << endl;

   return 0;
}
```




```
Объем параллелепипеда по умолчанию равен: 1
Объем параллелепипеда с длиной 10,
шириной 1 и высотой 1 равен: 10
Объем параллелепипеда с длиной 10,
шириной 5 и высотой 1 равен: 50
Объем параллелепипеда с длиной 10,
шириной 5 и высотой 2 равен: 100
```

КЛАСИ ПАМ'ЯТІ

- Ми використовували ідентифікатори для імен змінних.
- Атрибути змінних: ім'я, тип, розмір і значення.

float a;

a = 5.45;

- Використовували ідентифікатори для імен функцій.

```
// описание функции
int square(int y)
{
   return y * y;
}
```

- Кожен ідентифікатор в програмі має і інші атрибути: клас пам'яті, область дії та компоновку.

АТРИБУТИ ІДЕНТИФІКАТОРІВ

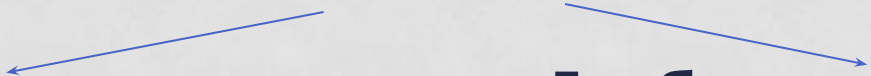
- Клас пам'яті ідентифікатора визначає його час життя – період, протягом якого ідентифікатор існує у пам'яті.
- Областю дії (областю видимості) ідентифікатора називається область програми, в якій на ідентифікатор можна посилатися.
- Компоновка ідентифікатора визначає для програми з багатьма файлами, чи відомий цей ідентифікатор тільки в одному поточному файлі або ж в будь-якому файлі з відповідним оголошенням.

КЛАСИ ПАМ'ЯТІ

В C++ є п'ять специфікаторів класу пам'яті:

- auto
- register
- extern
- static
- mutable

Специфікатори класів пам'яті



Локальний час життя

auto

register

Глобальний час життя

extern

stastic

КЛАСИ ПАМ'ЯТІ

Специфікатори класів пам'яті

Локальний час життя	Глобальний час життя
auto	*extern*
register	*static*

Локальними можуть бути лише ідентифікатори змінних.
Такі змінні створюються при вході до блоку, в якому вони оголошені. Існують лише під час активності блока та зникають при виході з блоку

```
auto float x, y;
register int counter = 1;
```

Глобальними можуть бути ідентифікатори змінних та функцій
Такі змінні та функції існують з моменту початку виконання програми.

```
extern double z;
static int counter = 1;
```

КЛАС ПАМ'ЯТІ
AUTO

- **auto** – локальні змінні функцій, параметри функцій
- **auto float x, y;**

- змінні з локальними часом життя, тобто існують тільки в тілі функції (блоку), в якому було оголошення.

- По замовчуванню локальні змінні є змінними типу auto

```
int square(int);  // прототип функции

main()
{
   for (int x = 1; x <= 10; x++)
      cout << square(x) << "  ";

      cout << endl;

   return 0;
}

// описание функции
int square(int y)
{
   return y * y;
}
```

КЛАС ПАМ'ЯТІ
REGISTER

- **register int counter = 1;**
- Специфікатор **register** означає, що компілятор зберігає змінну не в пам'яті, а в одному із високо швидких апаратних регістрів комп'ютера.
- Доцільно для змінних, що інтенсивно використовуються, таких як лічильники, суми. Тому що витрати на повторне завантаження змінних з пам'яті в регістр і назад можуть бути виключені.
- Компілятор може проігнорувати оголошення **register.**
- Часто оголошення register не є необхідним. Сучасні компілятори мають функцію оптимізації і можуть розпізнавати змінні, що часто використовуються, і вирішувати розміщувати їх в регістри або ні.

КЛАС ПАМ'ЯТІ
EXTERN

- Для оголошення ідентифікаторів змінних і функцій як ідентифікаторів з глобальним часом життя.
- Такі змінні існують з моменту початку виконання програми. Пам'ять виділяється та ініціалізується відразу після виконання програми.
- Імена функцій також існують з початку виконання програми.
- Глобальні змінні та імена функцій по замовчуванню відносяться до класу **extern**.

```
void a(void);       // прототип функции
void b(void);       // прототип функции
void c(void);       // прототип функции

int x = 1;          // глобальная переменная

main ( )
{
```

КЛАС ПАМ'ЯТІ
STATIC

- Змінні, оголошені зі специфікатором **static**, відомі тільки в тій функцій, в якій вони оголошені, але на відміну від звичайних локальних змінних, вони зберігають свої значення протягом всього часу існування функції. При кожному наступному виклику функцій змінні містять ті значення, які вони мали на попередньому виклику.

- **static int counter = 1;**

- Всі числові змінні класу **static** приймають значення 0 (по замовченню), якщо явно не вказане інше початкове значення

ПРАВИЛА, ЩО ВИЗНАЧАЮТЬ ОБЛАСТЬ ДІЇ (ОБЛАСТЬ ВИДИМОСТІ) ІДЕНТИФІКАТОРІВ

- **Область дії (видимості) ідентифікатора** – частина програми, а якій на ідентифікатор можна посилатись.
- Існують **чотири області дії ідентифікатора**:

- область дії файл;
- область дії функція;
- область дії блок;
- область дії прототип функції.

ОБЛАСТЬ ДІЇ
ФАЙЛ

- Ідентифікатор, оголошений поза будь-якої функції (на зовнішньому рівні) має **область дії файл.**
- Глобальні змінні, опис функцій, прототипи функцій – область дій файл.

```
int square(int);  // прототип функции

main()
{
   for (int x = 1; x <= 10; x++)
      cout << square(x) << "  ";

      cout << endl;

   return 0;
}

// описание функции
int square(int y)
{
   return y * y;
}
```

```
void a(void);      // прототип функции
void b(void);      // прототип функции
void c(void);      // прототип функции

int x = 1;         // глобальная переменная

main ( )
{
```

ОБЛАСТЬ ДІЇ
ФУНКЦІЯ

- **Мітки** (ідентифікатор з подальшою :)– єдині ідентифікатори, що мають **областю дії функцію**.
- switch
- goto

```
int main()
{ int n=10;
 loop:
    cout << n << ", ";
 n--;
 if (n>0)
   goto loop;
 cout << "FIRE!\n";
 return 0;
}
```

```
switch (x)
{
   case 1:
   case 2:
   case 3:
      cout << "x - це 1, 2 або 3";
      break;
   default:
      cout << "x – це не 1, 2 та не 3";
}
```

ОБЛАСТЬ ДІЇ
БЛОК

- Ідентифікатори оголошені всередині блок мають **область дії блок**.
- Область дії блок починається з оголошення ідентифікатора і закінчується правою фігурною дужкою }
- Локальні змінні, оголошені в функції; параметри функції; локальні змінні типу static - область дії блок.
- Будь-який блок може містити оголошення змінних.
- Якщо ідентифікатор у зовнішньому блоці має теж ім'я як і ідентифікатор у внутрішньому блоці, то ідентифікатор зовнішнього блоку є невидимим до моменту завершення роботи внутрішнього блоку.

ОБЛАСТЬ ДІЇ
ПРОТОТИП ФУНКЦІЙ

- Єдині ідентифікатори з **областю дії прототип функції** – ті, які використовуються в списку параметрів прототипу функції.

```
int square(int);  // прототип функции

main()
{
   for (int x = 1; x <= 10; x++)
      cout << square(x) << "  ";

      cout << endl;

   return 0;
}

// описание функции
int square(int y)
{
   return y * y;
}
```

```
void a(void);      // прототип функции
void b(void);      // прототип функции
void c(void);      // прототип функции

int x = 1;         // глобальная переменная

main ( )
{
   int x = 5;      // локальная переменная main

   cout << "локальная x во внешней области действия main = "
        << x << endl;
   {               // начало новой области действия
      int x = 7;

      cout <<"локальная x во внутренней области действия main = "
           << x << endl;
   }               // конец новой области действия

   cout << "локальная x во внешней области действия main = "
        << x << endl;

   a();      // a имеет автоматическую локальную переменную x
   b();      // b имеет статическую локальную переменную x
   c();      // c использует глобальную переменную x
   a();      // a заново дает начальное значение x
   b();      //статическая локальная x сохраняет предыдущее
             // значение
   c();      //глобальная x также сохраняет свое значение

   cout << "локальная x в main = " << x << endl;
return 0;
}
```

```
локальная x во внешней области действия main = 5
локальная x во внутренней области действия main = 7
локальная x во внешней области действия main = 5

локальная переменная x в a = 25 после входа в a
локальная переменная x в a = 26 перед выходом из a

локальная статическая переменная x = 50 при входе в b
локальная статическая переменная x = 51 при выходе из b

глобальная переменная x = 1 при входе в c
глобальная переменная x = 10 при выходе из c

локальная переменная x в a = 25 после входа в a
локальная переменная x в a = 26 перед выходом из a

локальная статическая переменная x = 51 при входе в b
локальная статическая переменная x = 52 при выходе из b

глобальная переменная x = 10 при входе в c
глобальная переменная x = 100 при выходе из c

локальная x в main = 5
```

```
void a(void)
{
   int x = 25;     // каждый раз a присваивается начальное значение

   cout << endl << "локальная переменная x в a = " << x
        << " после входа в a" << endl;
   ++x;
   cout << "локальная переменная x в a = " << x
        << " перед выходом из a" << endl;
}

void b(void)
{
   static int x = 50;    // Начальное значение присваивается только
                         // при первом вызове b

   cout << endl << "локальная статическая переменная x = " << x
        << " при входе в b" << endl;
   ++x;
   cout << "локальная статическая переменная x = " << x
        << " при выходе из b" << endl;
}
void c(void)
{
   cout << endl << "глобальная переменная x = " << x
        << " при входе в c" << endl;
   x *= 10;
   cout << "глобальная переменная x = " << x << " при выходе из c"
<< endl;
}
```

УНАРНА ОПЕРАЦІЯ ДОЗВОЛУ ОБЛАСТІ ДІЇ (ВИДИМОСТІ)

:: унарна операція дозволу області дії (видимості)

- дає доступ до глобальної змінної, навіть якщо під тим самим ім'ям в області дії (видимості) оголошена локальна змінна.

```
float value = 1.2345;

main()
{
    int value = 7;

    cout << "Локальное значение = " << value << endl
         << "Глобальное значение = " << ::value << endl;

    return 0;
}
```

```
Локальное значение = 7
Глобальное значение = 1.2345
```

ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ

- Перевантаження функції – визначення декількох функцій з одним іменем, але різним набором параметрів (або з різними типами параметрів)

```
int square(int x) { return x * x; }

double square(double y) { return y * y; }

main()
{
    cout << "Квадрат целого числа 7 равен "
         << square(7) << endl
         << "Квадрат числа 7.5 типа double равен "
         << square(7.5) << endl;

    return 0;
}
```

```
Квадрат целого числа 7 равен 49
Квадрат числа 7.5 типа double равен 56.25
```

ПЕРЕВАНТАЖЕННЯ ФУНКЦІЙ

- Створення перевантажених функцій з однаковими списками параметрів і різних типів значень, що повертаються, приводить до синтаксичних помилок!
- int square (int x) {…}
- double square (int x) {…}

- Перевантажені функції не обов'язково повинні мати однакову кількість параметрів!

ШАБЛОНИ ФУНКЦІЙ

- ***Перевантажені функції*** зазвичай використовуються для виконання схожих операції над різними типами даних.
- Якщо операції ідентичні для кожного типу можна використати ***шаблони функцій***.

```
template <class T>
T maximum(T value1, T value2, T value3)
{
   T max = value1;

   if (value2 > max)
      max = value2;

   if (value3 > max)
      max = value3;

   return max;
}

main()
{
   int int1, int2, int3;
   cout << "Введите три целых значения: ";
   cin >> int1 >> int2 >> int3;
   cout << "Максимальное целое значение равно: "
        << maximum(int1, int2, int3);           // версия int

   double double1, double2, double3;
   cout << endl << "Введите три значения double: ";
   cin >> double1 >> double2 >> double3;
   cout << "Максимальное значение double равно: "
        << maximum(double1, double2, double3);   // версия double

   char char1, char2, char3;
   cout << endl << "Введите три символа: ";
   cin >> char1 >> char2 >> char3;
   cout << "Максимальное значение символа равно: "
        << maximum(char1, char2, char3) << endl;  // версия char

   return 0;
}
```

```
Введите три целых значения: 1 2 3
Максимальное целое значение равно: 3
Введите три значения double: 3.3  2.2  1.1
Максимальное значение double равно: 3.3
Введите три символа: A  B  C
Максимальное значение символа равно: C
```

```
template <class T>
T maximum(T value1, T value2, T value3)
{
   T max = value1;

   if (value2 > max)
      max = value2;

   if (value3 > max)
      max = value3;

   return max;
}
```

```
int maximum(int value1, int value2, int value3)
{
   int max = value1;

   if (value2 > max)
      max = value2;

   if (value3 > max)
      max = value3;

   return max;
}
```