Лекція 2

Функції на Сі , області дії змінних та специфікатори

Базові типи Сі

-Порожній тип **void** (empty type) – неповний тип

-Цілі (integer)

-- символи(**char**) : can be set as integer (0..255) – *signed char, unsigned char, char*

➢Знакові цілі (signed integer types):

- standard: *signed char, short, int, long, long long* (since C99),
- extended: визначені додатково, e.g. *__int128*,

➢Натуральні цілі (unsigned integer types):

- standard: *_Bool* (since C99), *unsigned char, unsigned short, unsigned int, unsigned long, unsigned long long* (since C99) ,
- extended: визначені додатково, e.g. *__uint128*

- Дійсні float

➢Дісні (float types): *float, double, long double*

➢Комплексні (complex types): *float _Complex, double _Complex, long double _Complex,*

➢Уявні (imaginary types): *float _Imaginary, double _Imaginary, long double _Imaginary*

Повторення

Розгалудження:

1) a>b?a:b 2) if () { **} else{**} 3) switch() { case 1: *** }

Цикл з передумовою (while loop)

Цикл з лічильником (for loop)

Цикл з післямовою (do...while loop)

Вкладені цикли (nested loops)

Опис функції

```
тип ім'я_функції(список_параметрів або void)
{
   тіло_функції
   [return] (вираз);
}
```

де

тип – тип значення, яке повертає функція. Якщо "*тип*" містить слово void, то функція не повертає значення;

ім'я_функції – це ім'я функції. За цим іменем відбувається виклик програмного коду, реалізованого в *тілі_функції*. Крім того, *ім'я_функції* є *покажчиком на цю функцію*. Значенням покажчика є адреса точки входу в функцію;

список_параметрів – будь-який набір параметрів, що передаються в функцію. Якщо описується функція без параметрів, то в дужках вказується слово void або **порожні дужки**;

тіло_функції – набір операторів програмного коду;

return (*вираз*) – ключове слово return вказує, що функція повертає значення задане в

Функція що повертає значення

```c
//Ci (Ci++)
// функція, що множить параметр на 5
int Mult5(int d)
{
    int res;
    res = d * 5;
    return res; // повернення результату
}
```

Виклик функції з іншого програмного коду

...

```c
// виклик функції з іншого програмного коду
int x, y;
x = 20;
y = Mult5(x);   // y = 100
y = Mult5(-15); // y = -75
```

```python
# Python
# функція, що множить параметр на 5
def Mult5(d):

    res = d * 5;
    return res; # повернення результату
```

Виклик функції з іншого програмного коду

...

```python
# виклик функції з іншого програмного коду
x = 20;
y = Mult5(x);   # y = 100
y = Mult5(-15); # y = -75
```

Функція що повертає дійсне число

// C або C++

```
// функція, що знаходить максимум між двома дійсними числами
float MaxFloat(float x, float y)
{
    if (x>y) return x;
    else return y;
}
```

У даній функції 2 рази зустрічається оператор return.

Виклик функції MaxFloat() :

```
// виклик функції з іншого програмного коду
float Max; // змінна - результат
Max = MaxFloat(29.65f, (float)30); // Max = 30.0
double x = 10.99;
double y = 10.999;
Max = MaxFloat(x, y); // Max = 10.999
Max = MaxFloat((float)x, (float)y);
// Max = 10.999 - так надійніше
```

Python

```
# функція максимум між двох чисел
def MaxFloat(x, y):

    if x>y:
        return x
    else:
        return y
```

Виклик функції MaxFloat() з

```
# виклик функції
Max = MaxFloat(29.65, float(30))
x = 10.99
y = 10.999
Max = MaxFloat(x, y) #Max = 10.999
Max = MaxFloat(float(x), float(y)) # Max = 10.999
```

Функція яка не повертає значення

```
// опис функції, яка не отримує і не повертає параметрів
void MyFunc1(void)
{
    // тіло функції - вивід тексту на форму в компонент label1
    label1->Text = "MyFunc1() is called";
    return; // необовязкове
}

// опис функції, яка не отримує і не повертає параметрів
void MyFunc2(int x, double y)
{
    // тіло функції - вивід тексту на форму в компонент label1
     double z = x + y;
     printf("MyFunc2(%d %lf) is called", x, y);
     printf("sum = %lf", z);

     // return - необовязкове
}
```

```
# опис функції, яка не отримує і не повертає параметрів
def MyFunc1():
    # тіло функції - вивід тексту на форму в компонент label1
    Label1.Text = "MyFunc1() is called";
    return; // необовязкове

# опис функції, яка не отримує і не повертає параметрів
def MyFunc2(x, y):
    #тіло функції - вивід тексту на форму в компонент label1
     z = x + y;
     print("MyFunc2(%d %lf) is called"%(x, y));
     print("sum = {}".format(z));

   # return - необовязкове
}
```

Використання функції

```
// функція, що знаходить максимум між трьома цілими числами: 3 цілочисельних параметри з іменами a, b, c
int MaxInt3(int a, int b, int c){
    int max;
    max = a;
    if (max<b) max = b;
    if (max<c) max = c;
    return max;
}
void printSmallerSquares(int x) { // функція, що друкує квадрати чисел до x
int y;
for(y = 0; y * y <= x; y++) { printf(«%*d,\t \n»,y * y) ; }
}

int main(int argc, char **argv) { // виклик функції з іншого програмного коду
printSmallerSquares(7);  // 1, 4, 9, 16, 25, 36, 49
int a = 8, b = 5, c = -10;
int res;
// виклик функції з іншого програмного коду
res = MaxInt3(a, b, c);       // res = 8
res = MaxInt3(a, b+10, c+15); // res = 15
res = MaxInt3(11, 2, 18);     // res = 18
}
```

Прототипи

```
int max(int a, int b); // прототип max
int sum(int a, int b); // прототип sum
int main(){
    int Z= sum(1,2);  // використання sum
    printf(“z=%d”, Z);
}
int sum(int a, int b){ // реалізація sum
     return a+max(a,b); // використання max в sum
}
int max(int a, int b){ // реалізація max
     if (a>b) return a;
     else return b;
}
```

Тип void

- int func (**void**), він вказує, що функція не має параметрів.
- **void** func (int n), це вказує на те, що функція не повертає результату.
- **void** * - це тип вказівника, який не вказує, на що він вказує. При використанні в оголошенні покажчика **void** вказує, що покажчик "універсальний".

Приклади еквівалентні

```
void f(){};
void f(void){};
void f(){ return;}
void f(void) {return;}
```

Рекурсія

```
unsigned gcd(unsigned x, unsigned y); // рекурсивний
    НСД(Евклід)
unsigned gcd(unsigned x, unsigned y){ // реалізація алг-му
    Евкліда
  if(y==0) return x;   // обов□ язковий вихід з рекурсії
  return gcd(y,x%y); // рекурсія
}
```

```
int main(){
   int x;
   printf("Ввести число x=");
   scanf("%d",&x);
   unsigned y;
   printf("Ввести натуральне y=");
   scanf("%u",&y);
   printf("НСД=%u",gcd(x,y));
// (unsigned,int) -> (unsigned, unsigned) Ok
}
```

Типи змінних

Локальні змінні - всередині функції до програмного блоку. Вони існують тільки під час роботи функ-ції, а після реалізації функції система видаляє локальні змінні і звільняє пам’ять.

Ззовні всіх функції - **глобальні змінні** (global variables). Змінні, які описані поза всіма функціями, тобто на початку програми. До глобальних змінних можна звернутися з будь-якої функції та блока.

В визначенні функції, в якості аргументів - **формальні параметри** (formal parameters). В описі підпрограми присутні формальні параметри, які створюються в момент виклику підпрограми. При виклику з головної програми у назві підпрограми перераховуються фактичні параметри, які повинні мати значення. Ці значення передаються формальним параметрам

Локальні(автоматичні)

```
void test2(); // декларація функції test

int main()
{
int m = 22, n = 44;// m, n  локальні змінні
/* m та n мають область дії всередині main(). Функція test їх
не бачить.
m =a +b; --- помилка! При спробі використати тут a або b і в
інших функціях,буде помилка 'a' undeclared та 'b'
undeclared */
printf("\nvalues : m = %d and n = %d", m, n);
test2(); // Виклик test()
}

void test2() {// реалізація функції test
int a = 50, b = 80; // a, b локальні змінні test function
```

Глобальні змінні

```
void test();
int m = 22, n = 44; // Глобальні змінні
int a, b; // Глобальні змінні — ініціалізовані нулем: a=b=0
int main(){
 printf("All variables are accessed from main function");
 printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
// всі змінні доступні test();
}
void test(){
 printf("\n\nAll variables are accessed from test function");
  // всі змінні доступні
 printf("\nvalues: m=%d:n=%d:a=%d:b=%d", m,n,a,b);
 }
```

Ініціалізація глобальних змінних

Тип даних	Значення при створенні
int	0
char	'\0'
float	0
double	0
pointer	NULL

Формальні параметри

```
// функція, що знаходить модуль дійсного числа
float MyAbs(float x){ // x - формальний параметр
    if (x<0) return (float)(-x);
    else return x;
}
// виклик функції з іншого програмного коду (іншої функції)
float res, a; a = -18.25f;
res = MyAbs(a); // res = 18.25f; змінна a - фактичний параметр
res = MyAbs(-23); // res = 23; константа 23 - фактичний параметр
```

Властивості формальних параметрів

```
void change (int x, int y) {  // Ця функція міняє місцями x,y
 int k=x; x=y; y=k;
}
int main(){
  int x,y;   scanf("%d %d", &x, &y);
  change(x,y);          // але насправді не робить НІЧОГО!!!
  printf("%d %d", x, y);
}
```

Області дії (scope)

```
int a = 20; /* глобальна змінні */
int sum(int a, int b);
int main() {   /* main function */
  int a = 10; int b = 20; int c = 0; /* локальні змінні */
  printf ("value of a in main() = %d\n",  a);
  c = sum( a, b);
  printf ("value of c in main() = %d\n",  c);
  int k;
  return 0;
}

/*Реалізація */
int sum(int a, int b) { // фактичні параметри
   printf ("value of a in sum() = %d\n",  a);
   printf ("value of b in sum() = %d\n",  b);
   return a + b;
}
```

Результат:

```
value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30
```

Специфікатори змінних

Відсутність специфікаторів(auto) – застосовується для локальних змінних по замовчуванню. *Область видимості – обмежена блоком, в якому вони оголошені.*

static – застосовується як для локальних, так і для глобальних змінних. Область видимості локальної статичної змінної зберігається після виходу з блока чи функції, де ця змінна оголошена. Під час повторного виклику функції змінна зберігає своє попереднє значення.

```
static int x, y;  void f(){static float p = 3.25; }
```

extern – використовується для передачі для передачі значень глобальних змінних з одного файлу в інший (часто великі програми складаються з кількох файлів). Область дії – всі файли, з яких складається програма.

```
extern int N;
```

const - цей специфікатор вказує, що даній змінна не може бути модифікована в процесі роботи з нею і залишиться константою.

```
const unsigned N=100;    void fun(const int x, const double *a) {...}
```

Специфікатори

```
int global_var = 1; //може бути використаний в будь-якому місці цього
// вихідного файлу або в інших файлах, якщо відповідні зовнішні
// декларації в них є
extern int extern_globalvar;//його ініціалізація знаходиться в іншому файлі
static int private_var; /* Він може використовуватися в будь-якому місці цього
вихідного файлу, але функції в інших вихідних файлах не можуть отримати до неї
доступ */
static const int M=100; // статична та ще й незмінна
void F( const param_const){
  int localvar; // локальна змінною у функції f () неініціалізована
  int localvar2 = 2; //буде ініціалізована 2 кожного разу при виклику
  static int persistentvar; //локальна змінна  f (), ініціалізована 0
     //зберігає своє значення між викликами f ().
  const int local_const=10; // локальна константа
 localvar = param_const; persistentvar+=local_const; // це нормальний код
 //local_const=1; M++;  // а цей не відкомпілюється
  }
```

Перетворення типів (type casting)

```
#include <stdio.h>
main() {
   int sum = 17, count = 5;  double mean;
   mean = (double) sum / count; // перетворення int у double (пріорітет над діленням)

      float mean2 = (float) sum / count; //  перетворення int у float

      int z = (int) mean2; // перетворення float до int
      unsigned m = z;   // Ok.  Буде перетворення Int до Unsigned
      unsigned m2= (unsigned) z; //Ok. І тут теж буде
      int m3 = (int)m2; // Запрацює, але ...треба бути впевненим у значеннях  m2

   printf("Value of mean : %f\n", mean );
}
```

Функції зі змінною кількістю аргументів

```
#include<stdarg.h> // stdarg.h – описує макроси для variadic functions
#include<stdio.h>
int sum(size_t num_args, ...) {   // опис функції зі змінною к-тю арг-тів
   int val = 0;
   va_list ap;   // задає змінну для списку аргументів
   int i;
   va_start(ap, num_args); // задає початковий елемент списку арг-тів
   for(i = 0; i < num_args; i++) {
      val += va_arg(ap, int);  // перебирає арг-ти та приводить їх (double, int, char*)
   }
   va_end(ap); //  завершує перебор арг-тів
   return val;
}

int main(void) {
   printf("Sum of 10, 20 and 30 = %d\n",  sum(3, 10, 20, 30) );
   printf("Sum of 4, 20, 25 and 30 = %d\n",  sum(4, 4, 20, 25, 30) );
   return 0;
}
```

Головна функція

- int main(){ …} // не використвуємо аргументи командного рядку
- int main(int argc, char** argv) {…} // використуємо аргументи: приймаємо к-ть аргументів (argc) та список рядків (argv)
- int main(int argc, char* argv[ ]) {// використуємо аргументи: приймаємо к-ть аргументів (argc) та список рядків (argv) …}

Робота з головною функцією

```
#include <stdio.h>
   int main (int argc, char** argv) {
   int i;
   printf ( "% d \ n", argc);
   for (i = 0; i <argc; i ++){
   puts (argv [i]);
}
}
```

```
import sys

if __name__==‘__main__’:
   m = len(sys.argv)
   print ( “%d”%(m))
   for i in range(m):
   print(sys.argv [i]);
}
}
```

Передача значень у функцію

Виклик функції з передачею параметрів за допомогою формальних аргументів-значень або передача аргументів **за значенням** (Call-By-Value). Це є проста передача копій змінних в функцію. У цьому випадку зміна значень параметрів в тілі функції не змінить значення, що передавались у функцію ззовні (при її виклику):

```
void fun (int p) // функція fun() – аргмент p за значенням
{
        ++p;
      printf("%d", p);
}
 void main ( ) //---------- головна функція
{
      int x = 10;
      fun (x); //---------- виклик функції
      printf("%d",x); // вона не змінила  змінну x
}
```

Результат : **p=11, x=10**

Виклик функцій з передачею даних за допомогою глобальних змінних

```
#include < stdio.h >
int a, b, c;   // глобальні параметри
int sum ( );   //---------- прототип функції
 int main ( ) {
 scanf("%d %d",&a,&b);
sum();  //---------- виклик sum()
printf("c=%d",c);
}
 int  sum( )   //---------- функція sum()
{ c = a + b; }
```

Передача аргументів по вказівнику

```
/* Неправильне використання параметрів */
 void change (int x, int y) {
    int k=x;
    x=y;  y=k;
}
int main(){
   int x=2,y=3;
   change(x,y);
   printf("%d, %d",x,y);
}
```

Передача аргументів по вказівнику

```
void change (int *x, int* y) {  // функція приймає 2 вказівника
    int k= *x;  // змінюємо значення за адресами
    *x=*y;  *y=k; // самі адреси незмінні
}
int main(){
   int x=2, y=3;
   change(&x, &y); /* викликаємо функцію передаючі адреси
 бо  change(x, y) – не відповідність типів */
   printf("%d, %d", x, y);
}
```

Stack

int x =10

int x =10

Stack

int x =10

Stack

int* x =0xfff

int* x =0xfff

Stack

int* x =0xfff

Stack

int* x =0xfff

int* x =0xfff

Stack

int* x =0xfff

Heap

0xfff

Heap

0xfff

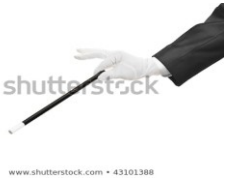
x=10

sizeof(x)

int* x =0xfff

int* x =0xfff

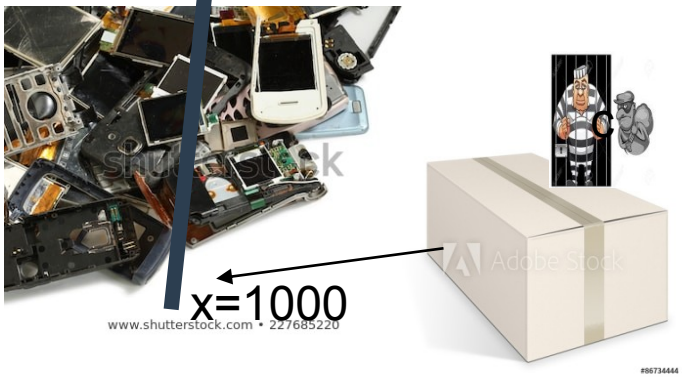
int* x =0xfff



*x = 1000

0xfff

0xfff



Виклик за посиланням: Call-By-Reference (C++ only)

```
#include <iostream>
#include <cstdio>
int func( int& v){  // функція за посиланням
    v++; return 0;
}
int main(){
    int x;
    func(x); // виклик функції за посиланням
    printf(“%d”, x);
}
```

```
// параметр x - передається за значенням (параметр-значення)
// параметр y - передається за адресою
// параметр z - передається за посиланням
void MyFunction(int x, int* y, int& c){
    x = 8; // значення параметра змінюється тільки в межах тіла функції
    *y = 8; // значення параметра змінюється також за межами функції
    c = 8; // значення параметра змінюється також за межами функції
    return;
}
```

```
int a, b, c;
a = b = c = 5;
// виклик функції MyFunction()
// параметр a передається за значенням a->x
// параметр b передається за адресою   b->y
// параметр c передається за посиланням c->z
MyFunction(a, &b, c); // на виході a = 5; b = 8; c = 8;
```

Вказівник на функцію

```
/* Ця функція має аргумент типу  int(*func)(int, int) тобто вказівник на функцію вигляду (2 цілих аргументи)-> цілий результат . Назва функціонального аргументу func */
int calculate ( int op1, int op2, int(*func)(int, int)) {
    return func(op1, op2); //  просто застосуваємо func до двох перших арг-тів
}

int summ (int op1, int op2){ // функція сума цілих чисел
    return op1 + op2;
}
int diff(int op1, int op2) {  // функція різниця цілих чисел
return op1 - op2;
}

int main(){
   // викликаємо calculate з коректними аргументами
   calculate(2, 3, summ);
   calculate(3, 4, diff);
}
```

Застосування вказівника на функцію

```
typedef int(*fint_t)(int, int); // ТИП: fint_t - вказівник на функцію
// функції 6-ти арифметичних операцій
fint_t foper[] = { // масив вказівників на функції
    summ, diff, mult, divd, bals, powr // // перелік функцій
};

int main() {
    setlocale(LC_ALL, "Ukrainian"); // українізуємо програму
    char coper[] = { '+', '-', '*', '/', '%', '^' }; // масив символів операцій
    int noper = sizeof(coper) / sizeof(coper[0]); // розмір масиву операцій
    do {
        char buf[120];
        char *str = buf;
        char *endptr;
        char oper;
        printf("вираз для обчислення (<op1><знак><op2>): ");
        fflush(stdin); // очищуємо буфер
        fscanf(stdin, "%s", buf);      // вводимо рядок
        if(strncmp(buf,"stop", 4)==0) break; // якщо він ‘stop’, виходимо
```

Застосування вказівника на функцію

```
    int op1, op2; // змінні під оператори
    op1 = strtod(str, &endptr); // конвертуємо рядок до цілого
    oper = *endptr++; // беремо наступний символ
    op2 = strtod(str = endptr, &endptr); //конвертуємо до цілого що залишилося

    int i;
    for (i = 0; i < noper; i++) { // обираємо потрібну операцію
        if (oper == coper[i]) { // та викликаємо calculate з нею
            printf(" %d %c %d = %d \n", op1, oper, op2 ,calculate(op1, op2, foper[i]));
            break;
        }
    }
    if (i == noper)
        printf("невірна операція: %c \n", oper);
  } while (1);
  // Якщо заданий тип fint_t, то можна визначити й змінні цього типу
 fint_t fun = summ; // такими 2-ма способами : просто рівністю
fint_t fun1 = &diff;  // або через адресу змінної
 printf("%d",calculate(7,3,fun)); printf("%d",calculate(7,3,fun1)); // та використати
}
```