Лекція 15.
Функції у мові Сі

Поняття функції

Функція *– це фрагмент програмного коду, до якого можна звернутися з іншого місця програми.*

Функція має ***ім'я****, за допомогою якого її можна* ***викликати****.*

Функція може ***приймати параметри і повертати*** *результат.*

Приклад використання функцій бібліотеки мови Сі:

```
int main() {
   double x, y;
   printf("x = ");
   scanf("%lf", &x);
   y = fabs(1-x) * fabs(x) / 2;
   printf("y = %f", y);
return 0;
}
```

Головні цілі використання функцій – **уникнення дублювання** коду та **спрощення** програмного коду.

```
y = ((1-x)?x-1:1-x) * (x ? x:-x) / 2;
```

так виглядає код без функції

```
y = fabs(1-x) * fabs(x) / 2;
```

а так - з функцією

Для того, щоб створити власну функцію, її треба **оголосити** (**описати**):

тип **ім‘я функції** (**параметри**)

{

тіло функції

}

набір параметрів, які потрібно буде вказувати при виклику функції

параметри записуються у формі:

(**тип1 змінна1**, **тип2 змінна2**, …)

Для того, щоб **повернути** (вказати) результат, потрібно використати у тілі функції оператор:
return значення результату;

Після того, як функція оголошена, її можна викликати, передаючи **аргументи** (**фактичні параметри**):

ім‘я функції(арг1, арг2, …);

Фактичні аргументи підставляються замість значень змінних, вказаних при оголошенні функції.

Приклад функції

```
#include<stdio.h>
int add(int r, int k) {
   r += k;
return r;
}
int main() {
   int x = 5;
   int z = 12;
   int y = add(x, z);
   printf("y = %d y = %d\n", y, add(x, z));
return 0;
}
```

У відладчику є безліч способів спостереження за виконанням коду.

- можна покроково пройти код і переглянути значення, що зберігаються в змінних;
- можна встановити контроль над змінними, щоб побачити, коли значення змінюються;
- можна перевірити шлях виконання коду.

Перехід по коду у відладчику за допомогою покрокових команд

Щоб відкрити програму з підключеним відладчиком натисніть клавішу F11 (Отладка -> Шаг с заходом).

При запуску програми за допомогою клавіші F11 відладчик зупиняється на першому виконуваному операторі.

Приклад функції

```
1  #include "pch.h"
2  #include<stdio.h>
3  int add(int r, int k) {
4      r += k;
5      return r;
6  }
7
8  int main() {
9      int x = 5;
10     int z = 12;
11     int y = add(x, z);
12     printf("y = %d y = %d\n", y, add(x, z));
13     return 0;
14 }
```

Приклад функції

```
double myabs(double r) {
  if (r > 0)
    return r;
  else
    return -r;
}
int main() {
  double x, y;
  printf("x = ");  scanf("%lf", &x);
  y = myabs(1 - x) * myabs(x) / 2;
  printf("y = %f", y);
return 0;
}
```

Приклад функції

```
1   #include "pch.h"
2   #include<stdio.h>
3
4   double myabs(double r) {
5       if (r > 0)
6           return r;
7       else
8           return -r;
9   }
10
11  int main() {
12      double x, y;
13      printf("x = ");
14      scanf_s("%lf", &x);
15          y = myabs(1 - x) * myabs(x) / 2;
16      printf("y = %f", y);
17      return 0;
18  }
```

ПРИКЛАД

```c
#include<stdio.h>

void Star(int);

int main() {
    Star(5);
    Star(3);
return 0;
}

void Star(int n) {
    for (int i = 0; i<n; i++)
        printf("%c", '*');
    printf("\n");
}
```

```c
#include<stdio.h>

int Star(int);

int main() {
    int n = 5;
    n=Star(n);
    n=Star(n);
return 0;
}

int Star(int n) {
    for(int i=0; i<n; i++)
        printf("%c", '*');
    printf("\n");
return n-2;
}
```

Оголошення функції:

тип ім‘я функції (параметри)

{

тіло (код) функції

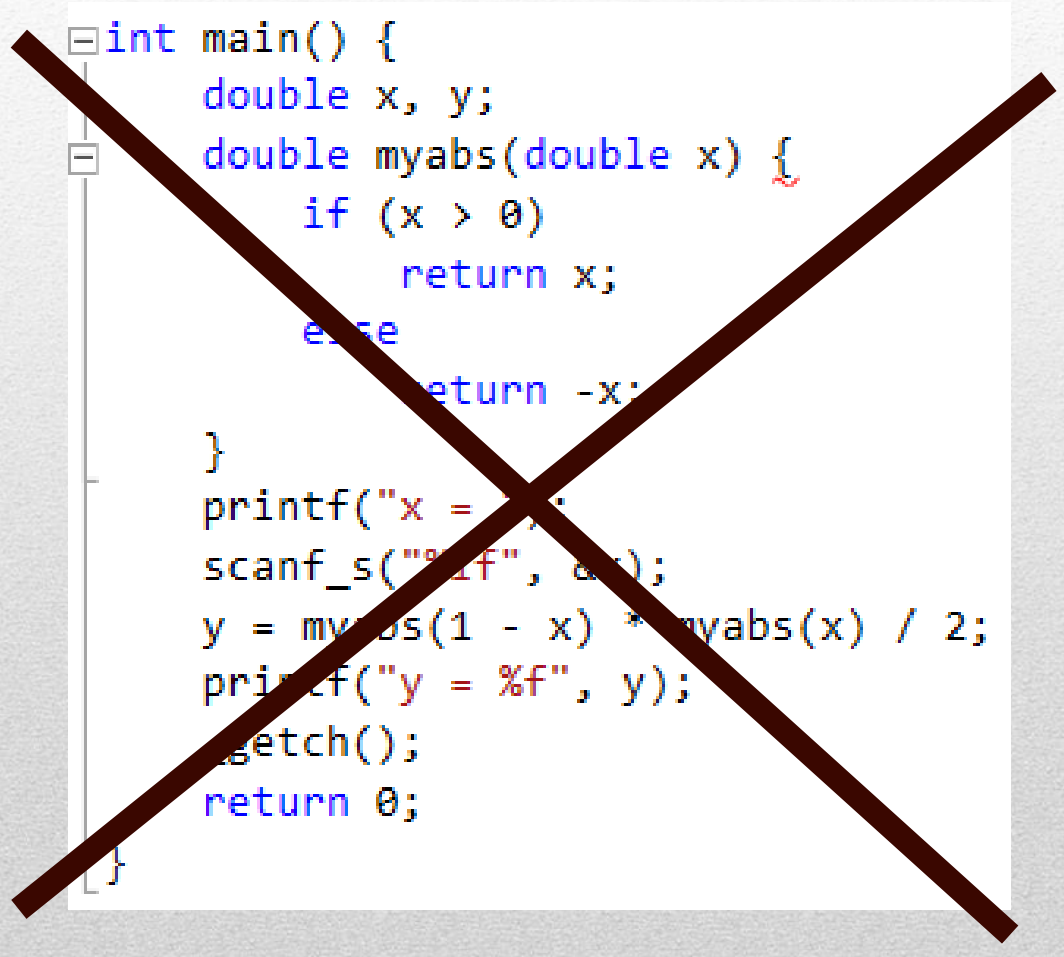
}

Прототип / заголовок / сигнатура функції

Сигнатура функції визначає правила використання функції.

Зазвичай сигнатура є описом функції, що включає ім'я функції, перелік формальних параметрів з їх типами і тип значення, що повертається.

Функції не можуть оголошуватися в середині інших функцій



Функція повинна бути оголошена до її виклику.

```
double myabs(double x) {
  if (x > 0) return x;
  else return -x;
}
int main() {
  double x, y;
  printf("x = ");  scanf("%lf", &x);
  y = myabs(1 - x) * myabs(x) / 2;
  printf("y = %f", y);
return 0;
}
```

Якщо потрібно викликати функцію до її оголошення, то до виклику функції потрібно розмістити її прототип.

```
double myabs(double x);
int main() {
    double x, y;
    printf("x = "); scanf("%lf", &x);
    y = myabs(1 - x) * myabs(x) / 2;
    printf("y = %f", y);
    _getch();
    return 0;
}
double myabs(double x) {
  if (x > 0) return x; else return -x;
}
```

У такому прототипі можна вказувати типи даних без назв змінних.

```
double myabs(double);
int main() {
    double x, y;
    printf("x = "); scanf("%lf", &x);
    y = myabs(1 - x) * myabs(x) / 2;
    printf("y = %f", y);
    _getch();
    return 0;
}
double myabs(double x) {
  if (x > 0) return x; else return -x;
}
```

При передачі у вигляді аргументів значення змінних не можуть бути змінені із середини функції:

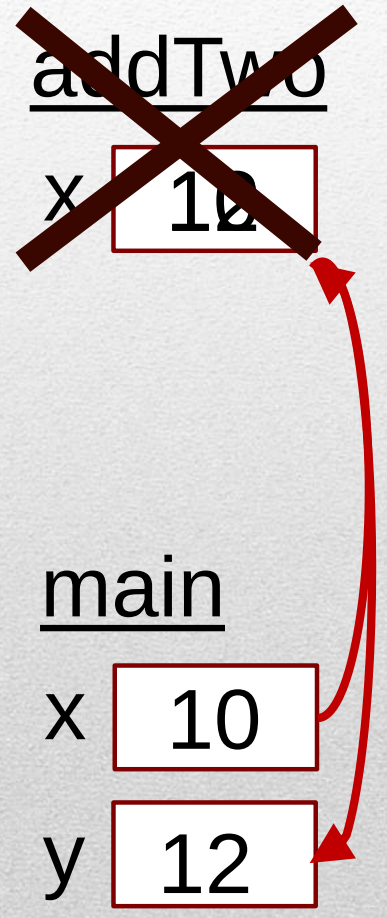
```
int addTwo(int x) {
   x += 2;
   return x;
}

int main() {
   int x = 10;
   int y = addTwo(x);
   printf("x = %d\ny = %d\n", x, y);
return 0;
}
```

усі змінні, які були створені у функції ***addTwo*** видаляються

значення змінної ***x***, яка знаходиться у функції ***main*** копіюється у змінну ***x***, яка знаходиться у функції ***addTwo***.

повертається значення змінної ***x*** з функції ***addTwo***; воно записується у змінну ***y***



При передачі у вигляді аргументів значення змінних не можуть бути змінені із середини функції.

Оскільки при виклику функції значення аргументів копіюються з місця виклику в параметри функції, то немає різниці, як саме називатимуться параметри функції.

```
int addTwo(int r) {
   r += 2;
   return r;
}
int main() {
   int x = 10;
   int y = addTwo(x);
   printf("x = %d\ny = %d\n", x, y);
return 0;
}
```

При виклику функції ***addTwo*** значення змінної ***x*** = 10 скопіюється у змінну ***r***, яка оголошена як параметр функції ***addTwo.*** Після завершення виконання функції ***addTwo*** змінна ***r*** буде видалена

Щоб передати масив у функцію потрібно:

1) у параметрі, який буде представляти собою масив, після імені поставити порожні квадратні дужки []:

```
double addTwo(double arr[]) {
   …
}
```

2) передати ще один параметр цілого типу, в якому міститиметься кількість елементів масиву:

```
double addTwo(double arr[], int count)
{
   …
}
```

```c
#include<time.h>
#include<stdlib.h>
#include<stdio.h>

void out(int arr[], int k);

int main() {
    srand(time(NULL));
    const int n = 10;
    int arr[n];
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 20 - 10;
    out(arr, n);
return 0;
}

void out(int arr[], int k) {
    for (int i = 0; i < k; i++)
    printf("%4d", arr[i]);
    printf("\n");
}
```

```c
#include<time.h>
#include<stdlib.h>
#include<stdio.h>
#define n 5

void out(int arr[][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
                printf("%4d", arr[i][j]);
        printf("\n");
    }
}

int main() {
    srand(time(NULL));
    int arr[n][n];
    for (int i = 0; i < n; i++)
        for(int j=0; j<n; j++)
                arr[i][j] = rand() % 20 - 10;
    out(arr);
return 0;
}
```