

Функції



Лекція №11

Дисципліна «Програмування»



Поняття функції

Функція – це самодостатня одиниця коду програми, яка спроектована для виконання окремої задачі.

Структура функції та способи її можливого використання визначаються синтаксичними правилами.

Припустимо, що треба написати програму, яка:

- читає список чисел;
- сортує ці числа;
- знаходить середнє значення цих чисел;
- виводить гістограму на екран.

```
#include <stdio.h>
#define SIZE 50

int main(void)
{
    float list[SIZE];

    readlist(list, SIZE);
    sort(list, SIZE);
    average(list, SIZE);
    bargraph(list, SIZE);

    return 0;
}
```



Функція як «чорний ящик»



Такий підхід по відношенню до функцій дозволяє зосередитися на загальній структурі програми, не відволікаючись на деталі.

Що необхідно знати про функції? Необхідно знати, **як їх правильно визначати, викликати та забезпечувати взаємодію між ними.**



Створення простої функції

```
#include <stdio.h>
#include <windows.h>

#define WORK    "Практична робота №7"
#define NAME    "Отримання випадкових чисел"
#define AUTHOR  "Виконав: студент групи КІТ-220а ЧЕРАБАЙ Ю.Ю."
#define WIDTH   50

void starbar(void);           // прототип функції

int main(void)
{
    SetConsoleOutputCP(1251);

    starbar();                // виклик функції
    printf("%s\n", WORK);
    printf("%s\n", NAME);
    printf("%s\n", AUTHOR);
    starbar();                // виклик функції

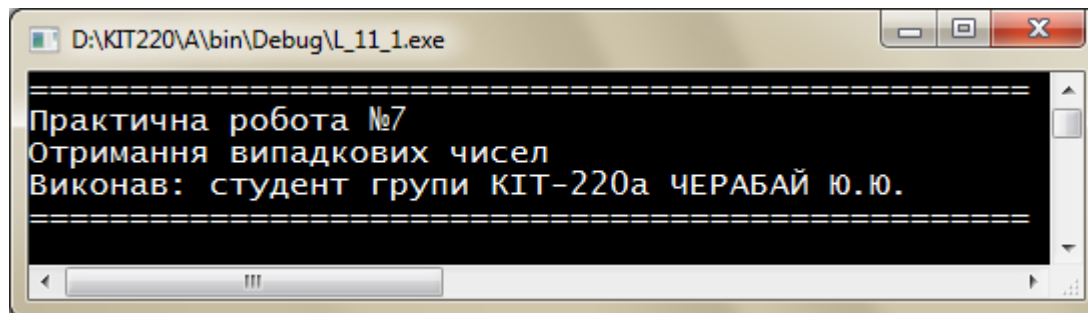
    return 0;
}
```



Створення простої функції

```
void starbar(void)           // визначення функції
{
    int count;

    for(count = 1; count <= WIDTH; count++)
        putchar('=');
    putchar('\n');
}
```





Аргументи функції

```
#include <stdio.h>
#include <string.h>                // для функції strlen()
#include <windows.h>
#define WORK    "Практична робота №7"
#define NAME    "Отримання випадкових чисел"
#define AUTHOR  "Виконав: студент групи КІТ-220а ЧЕРАБАЙ Ю.Ю."
#define WIDTH   50
#define SPACE   ' '
void show_n_char(char ch, int num); // прототип функції

int main(void)
{
    int spaces;

    SetConsoleOutputCP(1251);

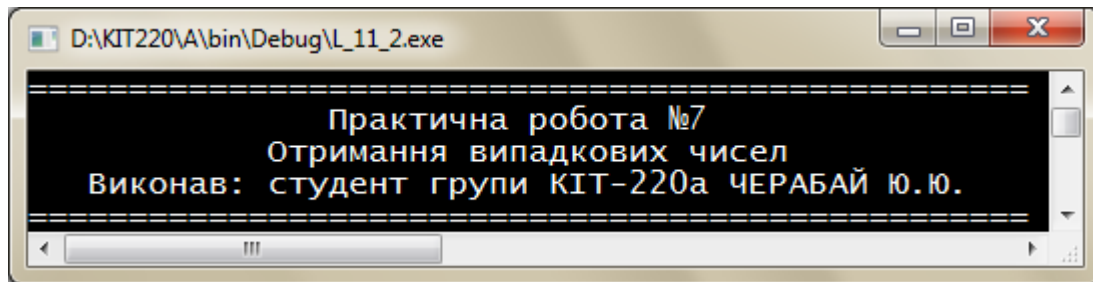
    // використання констант в якості аргументів
    show_n_char('=', WIDTH);
    putchar('\n');
    // використання констант в якості аргументів
    show_n_char(SPACE, 15);
    printf("%s\n", WORK);
}
```

Аргументи функції

```
// дозволити програмі обчислити кількість пробілів  
spaces = (WIDTH - strlen(NAME)) / 2;  
// використання констант в якості аргументів  
show_n_char(SPACE, spaces);  
printf("%s\n", NAME);  
show_n_char(SPACE, (WIDTH - strlen(AUTHOR)) / 2);  
printf("%s\n", AUTHOR);  
show_n_char('=', WIDTH);  
putchar('\n');
```

```
return 0;
```

```
}
```



```
// визначення функції show_n_char()  
void show_n_char(char ch, int num)  
{  
    for(int count = 1; count <= num; count++)  
        putchar(ch);  
}
```



Визначення функції з аргументами

Визначення функції починається з заголовка ANSI C:

```
void show_n_char(char ch, int num)
```

Цей рядок інформує компілятор про те, що функція `show_n_char()` приймає два аргументи з іменами `ch` і `num`.

`ch` має тип `char`, а `num` – тип `int`.

Змінні `ch` і `num` звуться формальними аргументами або формальними параметрами.

Формальні параметри є локальними змінними, які є закритими для функції. Значення цим змінним будуть присвоюватися під час виклику функції.

Зверніть увагу, що форма ANSI C вимагає, щоб кожній змінній передувала її тип.



Прототип функції з аргументами

Прототип ANSI с застосовується, щоб оголосити функцію перед її застосуванням:

```
void show_n_char(char ch, int num);
```

Коли функція приймає аргументи, **прототип визначає їх кількість і типи**, використовуючи розділений комами список типів.

За бажанням імена змінних в прототипі можна не вказувати:

```
void show_n_char(char, int);
```

Застосування імен змінних в прототипі не призводить до створення цих змінних. Це просто прояснює той факт, що **char** означає змінну типу **char** та ін.



Виклик функції з аргументами

```
show_n_char(SPACE, 15);
```

Фактичними аргументами є `SPACE` і `15`. Ці значення присвоюються відповідним формальним параметрам функції `show_n_char()` – змінним `ch` і `num`.

Формальний параметр – це змінна у функції, що викликається.

Фактичний аргумент – це конкретне значення, яке функція, що викликає, присвоює змінній всередині функції, що викликається.

```
show_n_char(SPACE, (WIDTH - strlen(AUTHOR)) / 2);
```

Обчислення довгого виразу, який утворює другий фактичний аргумент, дає в результаті `2`. Потім значення `2` присвоюється змінній `num`.



Повернення значення з функції

```
#include <stdio.h>
#include <windows.h>

int imin(int, int);

int main(void)
{
    int evil1, evil2;

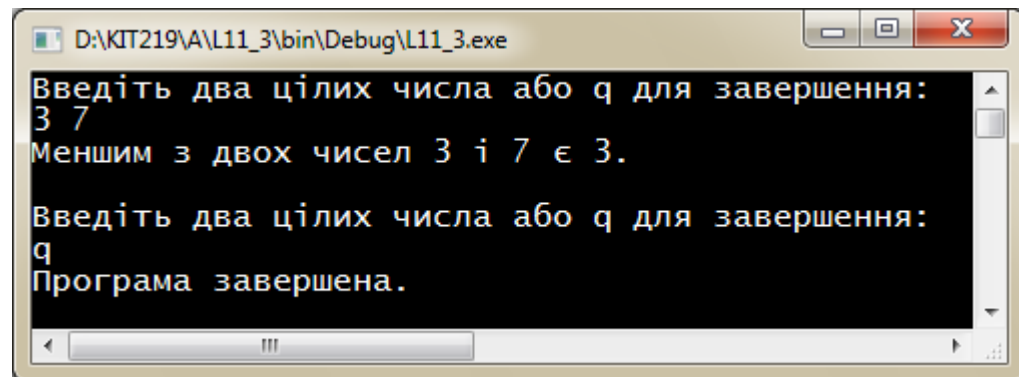
    SetConsoleOutputCP(1251);

    printf("Введіть два цілих числа або q для завершення:\n");
    while(scanf("%d %d", &evil1, &evil2) == 2)
    {
        printf("Меншим з двох чисел %d і %d є %d.\n\n",
               evil1, evil2, imin(evil1, evil2));
        printf("Введіть два цілих числа або "
               "q для завершення:\n");
    }
    printf("Програма завершена.\n");

    return 0;
}
```

Повернення значення з функції

```
int imin(int n, int m)
{
    return (n < m) ? n : m;
}
```





Невизначені аргументи

Для позначення того, що функція не приймає аргументів, слід вказати в круглих дужках ключове слово **void**:

```
void print_name(void) ;
```

Деякі функції, такі як `printf()` і `scanf()`, приймають змінну кількість аргументів. Наприклад, у `printf()` першим аргументом є рядок, але решта аргументів не фіксовані ні за типом, ні за кількістю.

Для таких випадків стандарт ANSI C дозволяє часткове зазначення прототипу.

```
int printf(const char *, ...);
```

Файл заголовку **stdarg.h** у бібліотеці функцій C надає стандартний спосіб для визначення функції зі змінною кількістю параметрів.



Уникнення прототипу

Існує один спосіб уникнути прототипу. Для цього необхідно помістити повне визначення функції до її першого використання. Частіше за все це робиться з короткими функціями:

```
// наступний код є визначенням і прототипом
int imax(int a, int b)
{
    return a > b ? a : b;
}

int main(void)
{
    int x, z;
    ...
    z = imax(x, 50);
    ...
}
```



Рекурсія

В мові с функції дозволено викликати саму себе. Цей процес має назву **рекурсії**. Часом рекурсія буває складною, але іноді й дуже зручною. Складність пов'язана з доведенням рекурсії до кінця, оскільки функція, яка викликає сама себе, має тенденцію робити це нескінченно, якщо в коді не передбачена перевірка умови завершення рекурсії.

Рекурсія часто може застосовуватися там, де застосовується цикл.

Іноді більш очевидним є рішення з циклом, а іноді – рішення з рекурсією.

Рекурсивні рішення є більш елегантними, але менш ефективними, ніж рішення з циклами.

Рекурсія

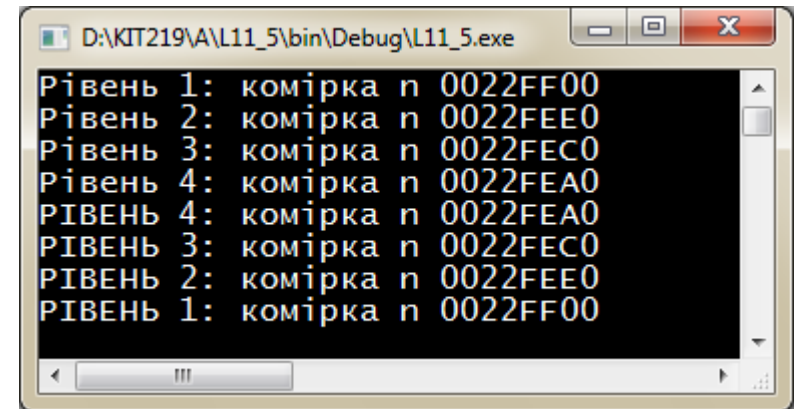
```
#include <stdio.h>
#include <windows.h>
```

```
void up_and_down(int);
```

```
int main(void)
{
    SetConsoleOutputCP(1251);

    up_and_down(1);
    return 0;
}
```

```
void up_and_down(int n)
{
    printf("Рівень %d: комірка n %p\n", n, &n);    // 1
    if(n < 4)
        up_and_down(n + 1);
    printf("Рівень %d: комірка n %p\n", n, &n);    // 2
}
```



```
D:\KIT219\A\L11_5\bin\Debug\L11_5.exe
Рівень 1: комірка n 0022FF00
Рівень 2: комірка n 0022FEE0
Рівень 3: комірка n 0022FEC0
Рівень 4: комірка n 0022FEA0
РІВЕНЬ 4: комірка n 0022FEA0
РІВЕНЬ 3: комірка n 0022FEC0
РІВЕНЬ 2: комірка n 0022FEE0
РІВЕНЬ 1: комірка n 0022FF00
```




Змінні рекурсії

Змінні:	n	n	n	n
Після виклику рівня 1	1			
Після виклику рівня 2	1	2		
Після виклику рівня 3	1	2	3	
Після виклику рівня 4	1	2	3	4
Після повернення з рівня 4	1	2	3	
Після повернення з рівня 3	1	2		
Після повернення з рівня 2	1			
Після повернення з рівня 1				
	(все завершено)			



Хвостова рекурсія

```
#include <stdio.h>
#include <windows.h>

long fact(int n);
long rfact(int n);

int main(void)
{
    int num;

    SetConsoleOutputCP(1251);

    printf("=====\n");
    printf(" Програма для обчислення факторіалу\n");
    printf("=====\n");
    printf("Введіть значення в діапазоні 0-12"
           " (q для завершення):\n");
    while(scanf("%d", &num) == 1)
    {
        if(num < 0)
            printf("Від'ємні числа не підходять.\n");
        else
        {
```



Хвостова рекурсія

```
    if(num > 12)
    {
        printf("Значення, що вводиться, повинно"
               " бути менше 13.\n");
    }
    else
    {
        printf("\nЦикл:      %d! = %ld\n",
               num, fact(num));
        printf("Рекурсія: %d! = %ld\n",
               num, rfact(num));
    }
}

printf("\nВведіть значення в діапазоні 0-12"
       " (q для завершення):\n");

printf("Програма завершена.\n");
return 0;
}
```

Хвостова рекурсія

// функція, що основана на циклі

```
long fact(int n)
```

```
{
```

```
    long ans;
```

```
    for(ans = 1; n > 1; n--)  
        ans *= n;
```

```
    return ans;
```

```
}
```

// рекурсивна версія

```
long rfact(int n)
```

```
{
```

```
    long ans;
```

```
    if(n > 0)  
        ans = n * rfact(n - 1);
```

```
    else
```

```
        ans = 1;
```

```
    return ans;
```

```
}
```



Зміна порядку на протилежний

```
#include <stdio.h>
#include <windows.h>

void to_binary(unsigned long n);

int main(void)
{
    unsigned long number;

    SetConsoleOutputCP(1251);

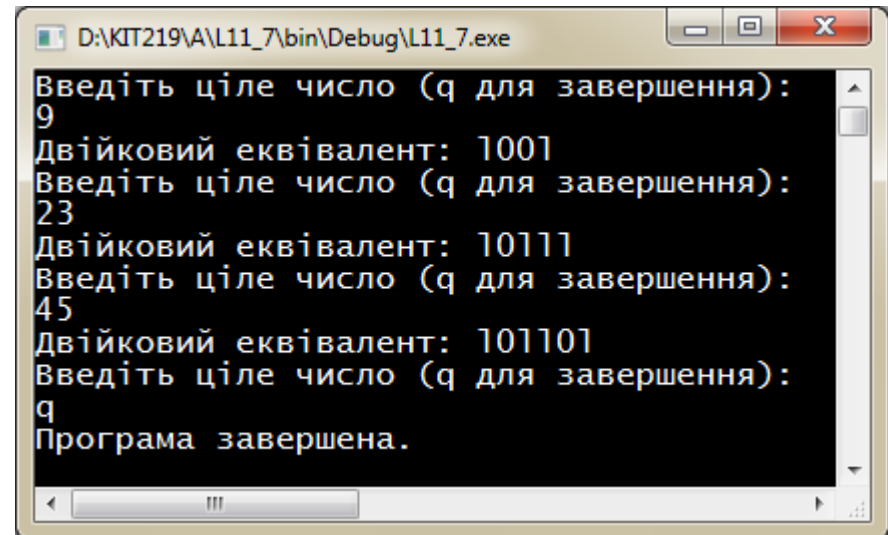
    printf("Введіть ціле число (q для завершення):\n");
    while(scanf("%lu", &number) == 1)
    {
        printf("Двійковий еквівалент: ");
        to_binary(number);
        putchar('\n');
        printf("Введіть ціле число (q для завершення):\n");
    }
    printf("Програма завершена.\n");
    return 0;
}
```

Зміна порядку на протилежний

```
void to_binary(unsigned long n)    // рекурсивна функція
{
    int r;

    r = n % 2;

    if (n >= 2)
        to_binary(n / 2);
    putchar(r == 0 ? '0' : '1');
    return;
}
```



```
D:\KIT219\A\L11_7\bin\Debug\L11_7.exe
Введіть ціле число (q для завершення):
9
Двійковий еквівалент: 1001
Введіть ціле число (q для завершення):
23
Двійковий еквівалент: 10111
Введіть ціле число (q для завершення):
45
Двійковий еквівалент: 101101
Введіть ціле число (q для завершення):
q
Програма завершена.
```