Лабораторна робота №17
Функції, багатофайлові проекти

Мета роботи: отримання практичних навичок написання та використання функцій, створення багатофайлових проектів.

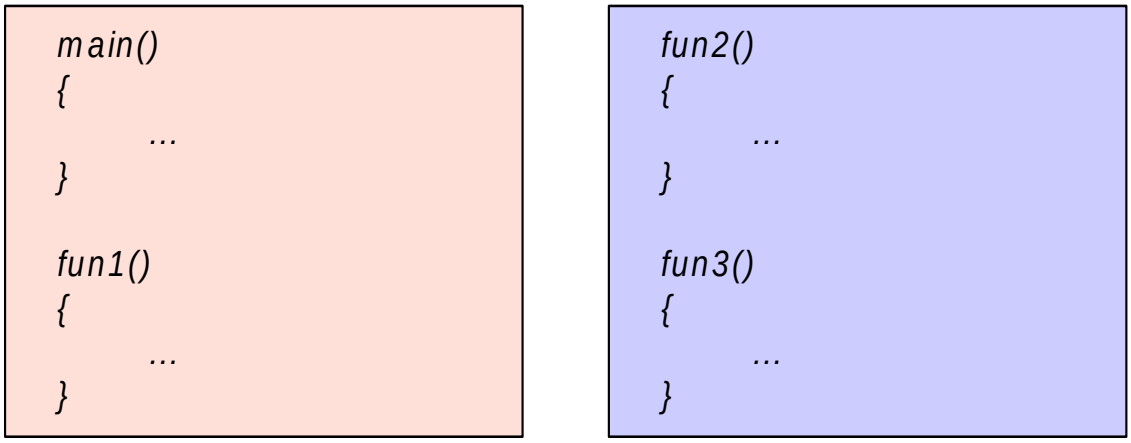
Зміст:

Короткі теоретичні відомості

Багатофайлові проекти

Звичайна *C*-програма складається із функції *main*, яка для виконання необхідних дій викликає інші функції. Зв'язок між функціями здійснюється за допомогою передачі параметрів і повернення значень функцій. Але компілятор мови *C* дозволяє також розбити програму на декілька окремих частин (чи навіть файлів).

При такій структурі функції, що знаходяться в різних файлах можуть використовувати глобальні зовнішні змінні. Всі функції в мові *C*, за замовчуванням, зовнішні і завжди доступні з будь-яких файлів. Наприклад, якщо програма (проект) складається із двох файлів (див. рисунок нижче), то функція *main* може викликати будь-яку з трьох функцій *fun1*, *fun2*, *fun3*, а кожна з цих функцій може викликати будь-яку іншу.



Для того, щоб функція могла виконувати будь-які дії, вона повинна використовувати змінні. У мові *C* всі змінні повинні бути оголошені до їх використання. Оголошення встановлюють відповідність імені та атрибутів змінної, функції або типу. Визначення змінної призводить до виділення пам'яті для зберігання її значення.

Область видимості змінної, пов'язані з поняттям блоку програми. У мові *C* блоком вважається послідовність оголошень, визначень і операторів, які обмежені фігурними дужками. Існують два види блоків:

- ✓ складений оператор,
- ✓ визначення функції, що складається із складеного оператора, який є тілом функції, і заголовка функції (в який входять ім'я функції, типи значення, що повертається і формальних параметрів) який передує тілу функції.

Блоки можуть включати в себе складені оператори, але не визначення функцій. Внутрішній блок називається вкладеним, а зовнішній блок - осяжним.

Час життя об'єкту – це інтервал часу виконання програми, протягом якого програмний об'єкт (змінна або функція) існує.

Час життя змінної може бути локальним або глобальним.

Змінна з глобальним часом життя має виділену для неї пам'ять і певне значення протягом усього часу виконання програми, починаючи з моменту виконання оголошення цієї змінної.

Змінна з локальним часом життя має виділену для неї пам'ять і певне значення тільки під час виконання блоку, в якому ця змінна визначена або оголошена. При

кожному вході в блок для локальної змінної виділяється нова пам'ять, яка звільняється при виході з блоку.

Всі функції в *C* мають глобальний час життя і існують протягом усього часу виконання програми.

Область видимості об'єкту – це частина тексту програми, в якій може бути використаний цей об'єкт. Об'єкт вважається видимим в блоці або у вихідному файлі, якщо в цьому блоці або файлі відомі ім'я та тип об'єкта. Об'єкт може бути видимим в межах блоку, вихідного файлу або у всіх вихідних файлах, що утворюють програму. Це залежить від того, на якому рівні оголошений об'єкт: на внутрішньому, тобто усередині деякого блоку, або на зовнішньому, тобто поза всіма блоками.

Якщо об'єкт оголошений усередині блоку, то він видимий в цьому блоці, і в усіх внутрішніх блоках. Якщо об'єкт оголошений на зовнішньому рівні, то він видимий від точки його оголошення до кінця цього файлу.

Об'єкт може бути зроблений глобально видимим за допомогою відповідних оголошень у всіх файлах, що утворюють програму.

Приклад:

```
------------ file1.c ------------
main()
{
    …
}

fun1()
{
    long int i[];
    …
}
--------------------------------
```

```
------------ file2.c ------------
long int i[MAX]={0};
fun2()
{
    …
}

fun3()
{
    …
}
---------------------------------
```

Перше оголошення змінної *i[]* в наведеному прикладі робить її видимою всередині функції *fun1()*. Визначення цієї змінної у файлі *file2.c* робить її видимою у функціях *fun2()* та *fun3()*.

Оголошення функцій

Оголошення функції складається з типу значення, що повертається, імені та списку параметрів. Разом ці три елементи складають прототип. Оголошення може з'явитися у файлі декілька разів.

Оголошення функцій найкраще поміщати в файли із заголовками (хедер-файли, файли з розширенням «*h*»), які можуть включатися всюди, де необхідно викликати функцію. Таким чином, всі файли використовують одне спільне оголошення.

Включення хедер-файлів

Хедер-файли включаються в текст програми за допомогою директиви препроцесора **`#include`**. Директиви препроцесора починаються зі знака «**#**». Програма, яка обробляє ці директиви, називається препроцесором (в сучасних компіляторах препроцесор, зазвичай, є частиною самого компілятора).

Директива **`#include`** включає в програму вміст зазначеного файлу. Файл може бути зазначено двома способами:

```
#include <path_specification>
#include " path_specification"
```

Якщо ім'я файлу вказано в кутових дужках (<>), то пошук файла для включення ведеться у наперед визначених місцях (залежить від реалізації). Якщо ім’я файлу вказано у лапках, то якщо це повне ім’я файлу, то пошук ведеться лише за вказаним шляхом. Якщо ім’я файла неповне, то пошук починається від каталогу розташування файл, що містить директиву включення. Якщо такий пошук не вдався, то пошук повторюється так, як для форми із кутовими дужками, тобто у наперед визначених каталогах. Перелік цих каталогів може бути задано ключем компілятора чи змінною оточення. Хедер-файл також може містити директиви **`#include`**.

Довідка. *Детальніше дивіться у стандарті* http://port70.net/~nsz/c/c11/n1570.html#6.10.2 або у документації виробника компілятора, наприклад, https://docs.microsoft.com/en-us/cpp/preprocessor/hash-include-directive-c-cpp.

Робоче завдання

1. Скласти програму, яка б реалізовувала обробку масиву (Таблиця 1) з використанням **ОКРЕМИХ** функцій користувача, для цього:
    a. написати універсальну функцію1 користувача, яка б заповнювала масив (матрицю) випадковими числами в діапазоні від 0 до 100 (в якості аргументу функції використовувати вказівник на масив, розмірності масиву задати за допомогою констант директивою ***#define***);

b. написати універсальну функцію2 користувача форматного друку на екран масиву (матриці) (в якості аргументів функції використовувати вказівник на масив і його розмірності);
c. написати функцію3 користувача, яка б реалізовувала обробку масиву відповідно до варіанту. В якості аргументів функції використовувати:
    - ✓ вказівник на функцію користувача, що заповнює масив випадковими числами,
    - ✓ вказівник на масив,
    - ✓ розмірності масиву.

2. Зберегти всі функції користувача в окремому (від функції *main*) файлі.
3. Реалізувати логіку програми шляхом послідовного виклику раніш створених функцій користувача з функції *main*.

Номер варіанта	Завдання
1.	У заданій матриці розмірності 4 на 5 визначити положення максимального по модулю елемента.
2.	У заданій матриці розмірності 6 на 3 визначити положення мінімального по модулю елемента.
3.	У заданій матриці розмірності 5 на 8 визначити положення і величину останнього негативного елементу.
4.	У заданій матриці розмірності 4 на 6 визначити номер рядка з максимальною сумою елементів і величину цієї суми.
5.	У заданій матриці розмірності 6 на 4 визначити номер стовпця з максимальною сумою елементів і величину цієї суми.
6.	У заданій матриці розмірності 4 на 6 визначити номер рядка з мінімальною сумою елементів і величину цієї суми.
7.	У заданій матриці розмірності 6 на 4 визначити номер стовпця з мінімальною сумою елементів і величину цієї суми.

Номер варіанта	Завдання
8.	У заданій матриці розмірності 6 на 6 визначити номер рядка з найбільшим по модулю діагональним елементом і величину цього елемента.
9.	У заданій матриці розмірності 7 на 5 визначити номер стовпця з найменшим по модулю елементом і величину цього елемента.
10.	У заданій матриці розмірності M на N поміняти місцями рядки з номерами m та n.
11.	У заданій матриці розмірності M на N поміняти місцями стовпчики з номерами m та n
12.	У заданій матриці розмірності 5 на 4 всі елементи менше заданої величини зробити рівними нулю, повернути кількість та суму цих елементів елементів.
13.	У заданій матриці розмірності 6 на 4 всі елементи, які більше заданої величини замінити значенням найбільшого елемента матриці. Повернути кількість та середнє значення замінених елементів.
14.	У заданій матриці розмірності 5 на 6 переставити рядки з найбільшим і найменшим за модулем елементами і повідомити номери цих рядків.
15.	У заданій матриці розмірності 6 на 5 переставити стовпчики з найбільшим і найменшим за модулем елементами і повідомити номери цих стовпців.
16.	У заданій матриці розмірності 4 на 6 знайти кількість і середнє значення парних елементів.
17.	У заданій матриці розмірності 5 на 6 знайти кількість і середнє значення непарних елементів.

Номер варіанта	Завдання
18.	У заданій матриці розмірності 6 на 5 визначити кількість та середнє значення елементів, кратних 3.
19.	У заданій матриці розмірності 7 на 5 визначити середнє значення елементів та кількість елементів, менших середнього значення.
20.	У заданій матриці розмірності 7 на 4 визначити кількість та суму елементів, розташованих між найбільшим і найменшим за модулем елементами, включаючи їх самих
21.	У заданій матриці розмірності 5 на 7 визначити кількість та суму елементів, які не перебувають між максимальними і мінімальними по модулю елементами, не включаючи їх самих.

Контрольні питання

1. У яких випадках зручно розташовувати функції в різних файлах?
2. Дайте визначення часу життя програмного об'єкта.
3. Який час життя мають функції мови *C*?
4. Дайте визначення області видимості програмного об'єкта.
5. Як і де оголошуються глобальні та локальні змінні?
6. Для яких цілей, зазвичай, служать хедер-файли?
7. Яким чином існуючий хедер-файл можна включити в текст програми?