**ЛАБОРАТОРНА РОБОТА № 7.**

**Функції в C++. Робота з файлами**

**Мета роботи**

Ознайомитися із особливостями застосування функцій та роботи з файлами у мові С++.

## 7.1. Функція.

**Визначення функції. Переваги використання функцій**

При написанні програм середнього та високого рівня складності виникає потреба в їх розбитті на частини. Розбиття великої програми на частини дозволяє зменшити ризик виникнення помилок, підвищує читабельність програмного коду завдяки його структуруванню.

Крім того, якщо деякий програмний код повторюється декілька разів (або є близьким за змістом), то є доцільним організувати його у вигляді функції, яку потім можна викликати багатократно за її іменем. Таким чином, відбувається економія пам’яті, зменшення вихідного коду програми, тощо.

**Функція** – це частина програми, яка має такі властивості чи ознаки:

* є логічно самостійною частиною програми;
* має ім’я, на основі якого здійснюється виклик функції (виконання функції). Ім’я функції підпорядковується правилам задавання імен ідентифікаторів мови C++;
* може містити список параметрів, які передаються їй для обробки або використання. Якщо функція не містить списку параметрів, то така функція називається функцією без параметрів;
* може повертати (не обов’язково) деяке значення. У випадку, якщо функція не повертає ніякого значення, тоді вказується ключове слово void;
* має власний програмний код, який береться у фігурні дужки { } і вирішує задачу, яка поставлена на цю функцію. Програмний код функції, реалізований в фігурних дужках, називається “тіло функції”.

**Використання функцій у програмах дає такі переваги**:

* компактна організація програми шляхом зручного виклику програмного коду за його іменем, який у програмі може зустрічатись декілька разів (повторюватись);
* економія пам’яті, розміру вихідного та виконавчого коду і т.д.;
* зменшення ризику виникнення помилок для великих наборів кодів;
* підвищення читабельності програмного коду.

**Загальна форма опису функції?**

Загальна форма опису функції виглядає наступним чином:

тип ім’я\_функції(список\_параметрів або void)

{

тіло\_функції

[return] (вираз);

}

де

* тип – тип значення, яке повертає функція. Якщо поле “тип” містить ключове слово void, то функція не повертає ніякого значення;
* ім’я\_функції – це безпосередньо 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 – параметри, які передаються в функцію. Функція може отримувати будь-яку кількість параметрів. Якщо описується функція без параметрів, то в дужках вказується слово void;
* тіло\_функції – набір операторів програмного коду, що реалізують алгоритм обчислення всередині функції;
* return (вираз) – ключове слово return вказує, що функція повертає значення задане в (вираз). Слово return може зустрічатись в декількох місцях тіла функції залежно від алгоритму (повторюватись).

**3. Приклади опису та використання функцій, що не повертають значення**

Якщо функція не повертає значення, тоді вона повинна починатися з ключового слова void.

**Приклад 1.** Функція MyFunc1() без параметрів, яка не повертає значення.

Якщо в тілі деякого класу або модуля описати функцію:

// опис функції, яка не отримує і не повертає параметрів

void MyFunc1(void)

{

// тіло функції - вивід тексту на форму в компонент label1

std::cout<<"MyFunc1() is called";

return; // повернення з функції

}

тоді викликати цю функцію можна наступним чином:

MyFunc1();

...

**Приклад 2.** Отримання параметру за значенням. Функція, яка отримує один параметр цілого типу, множить його на 5 і повертає результат. Функція не виконує виведення результату.

// 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

// виклик функції з іншого програмного коду

int x, y;

x = 20;

y = Mult5(x); // y = 100

y = Mult5(-15); // y = -75

**6. Які є способи передачі параметрів у функцію? Приклад**

В C++ існує **3 способи** передачі параметрів у функцію:

* передача параметру **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);
* передача параметру **за адресою** змінної. У цьому випадку функції в якості параметрів передаються не копії змінних, а копії адрес змінних, тобто покажчик на змінну. Використовуючи цей покажчик функція здійснює доступ до потрібних комірок пам’яті де розташована передана змінна і може змінювати її значення. Загальні відомості про покажчики наведено [**тут**](https://www.bestprog.net/uk/2016/12/08/%D1%96%D0%B4%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B8-%D0%B7%D0%B0%D1%80%D0%B5%D0%B7%D0%B5%D1%80%D0%B2%D0%BE%D0%B2%D0%B0%D0%BD%D1%96-%D1%81%D0%BB%D0%BE%D0%B2%D0%B0/);
* передача параметру **за посиланням** (Call-By-Reference). Передається посилання (покажчик) на об’єкт (змінну), що дозволяє синтаксично використовувати це посилання як покажчик і як значення. Зміни, внесені в параметр, що переданий за посиланням, змінюють вихідну копію параметра викликаючої функції.

Приклад**.** Цей приклад демонструє відмінність між передачею параметрів за значенням, передачею параметрів за адресою та передачею параметрів за посиланням. Описується функція, що отримує три параметри. Перший параметр (x) передається за значенням. Другий параметр (y) передається за адресою (як покажчик). Третій параметр (z) передається за посиланням.

// функція MyFunction

// 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;

}

Демонстрація виклику функції MyFunction() з іншого програмного коду:

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;

Як видно з результату, значення змінної a не змінилось. Тому що, змінна a передавалась у функцію MyFunction() з передачею значення (перший параметр).

Однак, значення змінної b після виклику функції MyFunction() змінилось. Це пов’язано з тим, що в функцію MyFunction() передавалось значення адреси змінної b. Маючи адресу змінної b в пам’яті комп’ютера, всередині функції MyFunction() можна змінювати значення цієї змінної з допомогою покажчика y.

Також змінилось значення **c** після виклику функції. Посилання є адресою об’єкту в пам’яті. З допомогою цієї адреси можна мати доступ до значення об’єкта.

**Що таке формальні та фактичні параметри функції? Приклад**

*Формальні параметри* – це змінні, що приймають значення аргументів (параметрів) функції. Якщо функція має декілька аргументів (параметрів), їх тип та імена розділяються комою ‘ , ‘.

При виклику функції, що має аргументи, компілятор здійснює копіювання копій формальних аргументів в стек.

Приклад**.** Функція MyAbs(), що знаходить модуль числа має один формальний параметр x.

// 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)

// виклик функції з іншого програмного коду

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

При виклику функції з іншого програмного коду фігурує фактичний параметр. У даному прикладі фактичний параметр це змінна a та константа 23.

При виклику функції фактичні параметри копіюються в спеціальні комірки пам’яті в стеку (стек – частина пам’яті). Ці комірки пам’яті відведені для формальних параметрів. Таким чином, формальні параметри (через використання стеку) отримують значення фактичних параметрів.

Оскільки, фактичні параметри копіюються в стек, то зміна значень формальних параметрів в тілі функції не змінить значень фактичних параметрів (тому що це є копія фактичних параметрів).

##### Область видимості формальних параметрів функції? Приклад

Область видимості формальних параметрів функції визначається межами тіла функції, в якій вони описані. У наведеному нижче прикладі формальний параметр n цілого типу має область видимості в межах фігурних дужок { }.

**Приклад.** Функція, що знаходить факторіал цілого числа n.

// функція, що знаходить n!

unsigned long int MyFact(int n) // початок області видимості формального параметру n

{

int i;

unsigned long int f = 1; // результат

for (i=1; i<=n; i++)

f = f\*i;

return f; // кінець області видимості формального параметру n

}

Виклик функції з іншого програмного коду (іншої функції):

// виклик функції з іншого програмного коду

int k;

unsigned long int fact;

k = 6;

fact = MyFact(k); // fact = 6! = 720

**Що таке прототип функції? Приклади**

*Прототип функції* – це повідомлення компілятору та іншим функціям про те, що така функція існує. Іншими словами, це повідомлення компілятору та іншим функція про те, що у програмі існує функція з заданою сигнатурою. Прототип функції складається з скороченого опису без тіла функції.

Прототип функції містить:

* ім’я функції;
* параметри функції або типи цих параметрів;
* тип значення, яке повертається функцією.

Загальна форма оголошення прототипу функції наступна:

return\_type FuncName(parameters);

де

* *return\_type* – тип, що повертається функцією;
* FuncName – ім’я функції;
* *parameters* – параметри функції.

При вказанні прототипу функції необов’язково вказувати імена параметрів. Наприклад, для функції

double Average(int A[], int size);

можна задати наступний прототип

double Average(int[], int);

У випадках, коли функція, яку викликають, оголошена перед викликаючою її функцією можна обійтись без використання прототипу. В інших випадках прототип функції необхідний.

Якщо функція описується в класі і викликається з методів цього класу, тоді подібних помилок не буде. Це пов’язане з тим, що в класі прототип функції відомий усім методам класу.

**Приклад опису функції, що отримує масив цілих чисел**

Для передачі [масиву](https://www.bestprog.net/uk/2017/03/01/%d0%bc%d0%b0%d1%81%d0%b8%d0%b2%d0%b8-%d1%87%d0%b0%d1%81%d1%82%d0%b8%d0%bd%d0%b0-1-%d0%b2%d0%b8%d0%b7%d0%bd%d0%b0%d1%87%d0%b5%d0%bd%d0%bd%d1%8f-%d0%bc%d0%b0%d1%81%d0%b8%d0%b2%d1%83-%d0%be%d0%b4/) чисел в функцію потрібно передати [покажчик](https://www.bestprog.net/uk/2017/03/21/%d0%bf%d0%be%d0%ba%d0%b0%d0%b6%d1%87%d0%b8%d0%ba%d0%b8-%d1%87%d0%b0%d1%81%d1%82%d0%b8%d0%bd%d0%b0-1-%d0%b7%d0%b0%d0%b3%d0%b0%d0%bb%d1%8c%d0%bd%d1%96-%d0%bf%d0%be%d0%bd%d1%8f%d1%82%d1%82%d1%8f/) на цей масив чисел.

Покажчиком на масив чисел є:

* ім’я масиву;
* адреса першого елементу масиву.

Приклад**.** Нехай потрібно описати функцію SumArrayInts(), що отримує масив з цілих чисел і знаходить суму елементів цього масиву. Функція отримує 2 параметри:

* масив цілих чисел;
* кількість елементів у масиві.

Спосіб 1. Передача масиву як int A[].

// Функція, що підраховує суму елементів масиву цілих чисел

// Функція отримує 2 параметри:

// n - кількість елементів масиву,

// A - масив цілих чисел

int SumArrayInts(int n, int A[])

{

int i;

int sum = 0; // сума

for (i=0; i<n; i++)

sum = sum + A[i];

return sum;

}

Спосіб 2. Передача масиву як int \*A.

int SumArrayInts(int n, int \*A) // масив передаєтсья як \*A

{

int i;

int sum = 0; // сума

for (i=0; i<n; i++)

sum = sum + A[i];

return sum;

}

Виклик функції SumArrayInts() з іншого програмного коду.

// передача масиву у функцію

int M[5] = { 23, -2, -1, -8, 4 };

int n = 5;

int summa;

summa = SumArrayInts(n, M); // summa = 16

summa = SumArrayInts(n, &M[0]); // теж добре, summa = 16

Слід зауважити, що при такій передачі масиву у функцію, елементи масиву можна змінювати в тілі функції.

##### Приклад передачі двовимірного масиву як параметру функції

У даному прикладі описана функція, що знаходить суму елементів двовимірного масиву. Функція, яка отримує двовимірний масив дійсних чисел в якості параметру.

// функція обчислення суми елементів двовимірного масиву

double Sum2(double A[][3])

{

double s = 0;

int i,j;

for (i=0; i<2; i++)

for (j=0; j<3; j++)

s = s + A[i][j];

return s;

}

Виклик функції з програмного коду. У функцію передається двовимірний масив з іменем M розміром 2\*3.

// Передача двовимірного масиву як параметру функції

// двовимірний масив чисел

double M[][3] = { { 2.78, -3.18, 9.4 },

{ -3.4, 8.8, 0.5 } };

double summa;

summa = Sum2(M);

##### Що таке рекурсія? Що називається рекурсією?

Будь-яка функція (метод) у своєму тілі може викликати сама себе. Рекурсія – це такий спосіб задавання функції, при якому результат повернення з функції для даного значення аргументу визначається на основі результату повернення з тієї ж функції для попереднього (меншого або більшого) значення аргументу.

Якщо функція (метод) викликає сама себе, то такий виклик називається рекурсивний виклик функції. При кожному рекурсивному виклику запам’ятовуються попередні значення внутрішніх локальних змінних та переданих у функцію параметрів. Щоб наступний крок рекурсивного виклику відрізнявся від попереднього, значення як мінімум одного з параметрів функції повинно бути змінене. Припинення процесу рекурсивних викликів функції відбувається, коли змінюваний параметр досягнув деякого кінцевого значення, наприклад, оброблено останній елемент в масиві.

**Як правильно організувати рекурсивний виклик функції?**

Рекурсивне звернення до функції може бути здійснене, якщо алгоритм визначений рекурсивно.

Щоб циклічний процес перетворити в рекурсивний, потрібно вміти визначити (виділити) три важливі моменти:

* умову припинення послідовності рекурсивних викликів функції. Наприклад, якщо лічильник або ітератор з іменем k змінюється від 1 до 10 у зростаючому порядку, то умова припинення є досягнення значення k=10. Умова припинення вказується в операторі return;
* формулу наступного елементу або ітератору, що використовується в рекурсивному процесі. Ця формула вказується в операторі return;
* перелік параметрів, що передаються в рекурсивну функцію. Один з параметрів обов’язково є ітератор (лічильник), який змінює своє значення. Інші параметри є додатковими, наприклад, посилання на масив, над яким здійснюється обробка.

##### Пошук суми елементів масиву. Приклад

За подібним зразком можна створювати власні рекурсивні функції, які визначають будь-які суми елементів будь-яких масивів.

**Задача.** Задано масив чисел A. Розробити рекурсивну функцію, яка знаходить суму елементів масиву:

* S **=** A**[0] +** A**[1] + … +** A**[**n**]**,

де n – кількість елементів масиву. Програмний код функції наступний:

// сума елементів масиву  
int Sum(int i, int A[], int n)  
{  
 if (i==n-1)  
 return A[i];  
 else  
 return A[i]+Sum(i+1,A,n);  
}

як видно з прикладу, у рекурсивну функцію Sum() передається 3 параметри:

* поточне значення ітератора *i*;
* масив *A*;
* кількість елементів масиву *n*.

Вихід з функції здійснюється, коли буде оброблено останній елемент масиву. Умова припинення рекурсивного процесу має вигляд:

if (i==n-1)  
 return A[i];

Для сумування поточного значення A[i] з наступними вказується рядок:

return A[i]+Sum(i+1,A,n);

де рекурсивний виклик Sum(i+1, A, n) означає наступне значення масиву A. Параметр i+1 вказує, що на наступному рівні виклику рекурсивної функції буде взято наступний за даним елемент масиву.

Використання функції Sum() в іншому програмному коді може бути таким:

int A[] = { 5, 7, 2, -1 };  
int n = 4;  
int sum;  
sum = Sum(0,A,n); // sum = 13

**Приклад знаходження факторіалу числа – n!**

Обчислення факторіалу числа методом рекурсії є майже в кожному підручнику. Факторіал числа обчислюється за формулою

f = 1 · 2 · … · (n-1) · n

де n>=1.

Рекурсивний виклик можна організувати двома способами:

* в порядку зростання 1, 2, …, *n*;
* в порядку спадання *n*, *n*-1, …, 2, 1.

У даному прикладі розроблено дві функції, які знаходять факторіал обома способами. Програмний код функцій наступний:

// обчислення факторіалу - спосіб 1  
int Fact1(int n, int i)  
{  
 if (i==n)  
 return n; // умова завершення рекурсивного процесу  
 else  
 return i\*Fact1(n, i+1); // перехід до наступного числа i+1  
}  
  
// обчислення факторіалу - спосіб 2

int Fact2(int n)  
{  
 if (n==1)  
 return 1; // умова завершення рекурсивного процесу  
 else  
 return n\*Fact2(n-1); // перехід до попереднього числа n-1  
}

У першу функцію Fact1() передаються 2 параметри. Перший параметр визначає максимально можливе значення n, яке може бути помножене. Другий параметр визначає поточне значення **i** яке приймає участь у множенні.

У другу функцію Fact2() передається 1 параметр – максимально можливе значення n, що бере участь у рекурсивному множенні. Другого параметру тут не потрібно, оскільки межа припинення рекурсивного процесу є відома і рівна 1. У функції Fact2() рекурсивний процес завершується коли n=1.

## 7.2.Обробка файлів C++: як відкривати, писати, читати, закривати файли на C++

Що таке обробка файлів у C++?

Файли постійно зберігають дані на пристрої зберігання. За допомогою обробки файлів вихідні дані програми можна зберігати у файлі. У файлі над даними можна виконувати різні операції.

Потік — це абстракція пристрою, де виконуються операції введення/виведення. Ви можете представити потік або як пункт призначення, або як джерело символів невизначеної довжини. Це визначатиметься їх використанням. C++ надає вам бібліотеку з методами обробки файлів. Давайте обговоримо це.

**Бібліотека fstream**

Бібліотека fstream надає програмістам C++ три класи для роботи з файлами. Ці класи включають:

* **ofstream** – цей клас представляє вихідний потік. Він використовується для створення файлів і запису інформації у файли.
* **ifstream** – цей клас представляє вхідний потік. Він використовується для читання інформації з файлів даних.
* **fstream** – цей клас зазвичай представляє потік файлів. Він поставляється з можливостями ofstream/ifstream. Це означає, що він здатний створювати файли, записувати файли, читати файли даних.

Щоб використовувати наведені вище класи бібліотеки fstream, ви повинні включити її у свою програму як файл заголовка. Звичайно, ви будете використовувати директиву препроцесора #include. Ви також повинні включити файл заголовка iostream.

**Як відкривати файли**

Перш ніж виконувати будь-яку операцію з файлом, його необхідно відкрити. Якщо вам потрібно записати файл, відкрийте його за допомогою об’єктів fstream або ofstream. Якщо вам потрібно лише прочитати файл, відкрийте його за допомогою об’єкта ifstream.

Три об’єкти, тобто fstream, ofstream і ifstream, мають визначену функцію open(). Синтаксис функції:

open (file\_name, mode);

* Параметр file\_name позначає назву файлу, який потрібно відкрити.
* Параметр mode необов'язковий. Він може приймати будь-яке з наступних значень:

| **Значення** | **опис** |
| --- | --- |
| ios:: app | Режим додавання. Вихідні дані, надіслані до файлу, додаються до нього. |
| ios::ate | Він відкриває файл для виведення, а потім переміщує елементи керування читанням і записом у кінець файлу. |
| ios::in | Він відкриває файл для читання. |
| ios::out | Він відкриває файл для запису. |
| ios::trunc | Якщо файл існує, елементи файлу слід скоротити перед його відкриттям. |

Є можливість використовувати два режими одночасно. Ви об’єднуєте їх за допомогою | (OR) оператор.

**приклад 1:**

#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

fstream my\_file;

my\_file.open("my\_file", ios::out);

if (!my\_file) {

cout << "Файл не створено!";

}

else {

cout << "Файл створено успішно!";

my\_file.close();

}

return 0;

}

**Вихід:**

Файл створено успішно!

Ось скріншот коду:

**Пояснення коду:**

1. Включіть файл заголовка iostream у програму, щоб використовувати її функції.
2. Включіть файл заголовка fstream у програму, щоб використовувати його класи.
3. Включіть простір імен std у наш код, щоб використовувати його класи без його виклику.
4. Виклик функції main(). Логіка програми повинна знаходитися в її тілі.
5. Створіть об’єкт класу fstream і назвіть його my\_file.
6. Застосуйте функцію open() до вищевказаного об’єкта, щоб створити новий файл. Режим виходу дозволяє нам записувати у файл.
7. Використовуйте оператор if, щоб перевірити, чи не вдалося створити файл.
8. Повідомлення для друку на консолі, якщо файл не було створено.
9. Кінець тіла оператора if.
10. Використовуйте оператор else, щоб вказати, що робити, якщо файл було створено.
11. Повідомлення для друку на консолі, якщо файл було створено.
12. Застосуйте функцію close() до об’єкта, щоб закрити файл.
13. Кінець тіла оператора else.
14. У разі успішного завершення програма повинна повернути значення.
15. Кінець тіла функції main().

**Як закрити файли**

Щойно програма C++ завершує роботу, вона автоматично

* спорожнює потоки
* звільняє виділену пам'ять
* закриває відкриті файли.

Однак, як програміст, ви повинні навчитися закривати відкриті файли до завершення програми.

Об’єкти fstream, ofstream і ifstream мають функцію close() для закриття файлів. Синтаксис функції:

## void close();

**Як писати у файли**

Ви можете писати у файл прямо з програми C++. Для цього використовується оператор вставки потоку (<<). Текст, який потрібно записати у файл, потрібно взяти в подвійні лапки.

Давайте продемонструємо це.

### приклад:

#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

fstream my\_file;

my\_file.open("my\_file", ios::out);

if (!my\_file) {

cout << "Файл не створено!";

}

else {

cout << "Файл успішно створено!";

my\_file << "Привіт, світ!";

my\_file.close();

}

return 0;

}

**Вивід:**

Файл успішно створено!

Ось скріншот коду:

**Пояснення коду:**

1. Включіть файл заголовка iostream у програму, щоб використовувати її функції.
2. Включіть файл заголовка fstream у програму, щоб використовувати його класи.
3. Включіть простір імен std у програму, щоб використовувати її класи без її виклику.
4. Виклик функції main(). Логіку програми слід додати в тіло цієї функції.
5. Створіть екземпляр класу fstream і дайте йому назву my\_file.
6. Використовуйте функцію open(), щоб створити новий файл під назвою my\_file.txt. Файл буде відкритий у вихідному режимі для запису в нього.
7. Використовуйте оператор if, щоб перевірити, чи не було відкрито файл.
8. Текст для друку на консолі, якщо файл не відкрито.
9. Кінець тіла оператора if.
10. Використовуйте оператор else, щоб вказати, що робити, якщо файл було створено.
11. Текст для друку на консолі, якщо файл було створено.
12. Напишіть текст у створений файл.
13. Використовуйте функцію close(), щоб закрити файл.
14. Кінець тіла оператора else.
15. Після успішного завершення програма повинна повернути значення.
16. Кінець тіла функції main().

**Як читати з файлів**

Ви можете зчитувати інформацію з файлів у програму C++. Це можливо за допомогою оператора вилучення потоку (>>). Ви використовуєте оператор так само, як і для читання введених користувачем даних із клавіатури. Однак замість використання об’єкта cin ви використовуєте об’єкт ifstream/fstream.

**приклад 3:**

#include <iostream>

#include <fstream>

#include <Windows.h>

using namespace std;

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

fstream my\_file;

my\_file.open("my\_file.txt", ios::in);

if (!my\_file) {

cout << "Немає такого файлу";

}

else {

char ch;

while (1) {

my\_file >> ch;

if (my\_file.eof())

break;

cout << ch;

}

}

my\_file.close();

return 0;

}

**Вихід:**

Немає такого файлу

Ось скріншот коду:

# 7.3. Контрольні запитання

1. Коли виникає необхідність застосування функцій?
2. Який загальний вигляд функцій?
3. Як функція повертає значення?
4. Які завдання виконує прототип функції?
5. Як виглядає функція типу **void**?
6. Як реалізується виклик функції з передачею значень?
7. Як реалізується виклик функції з передачею адреси змінних?
8. Як реалізується виклик функції з масивом?
9. Як реалізується виклик функції зі змінним числом параметрів?
10. Що таке рекурсія?
11. Які Ви знаєте параметри функції **main()**?

# 7.4. Лабораторне завдання

1. Ознайомитися із особливостями застосування функцій у мові С.
2. Одержати індивідуальне завдання.
3. Побудувати блок-схеми алгоритмів відповідно до завдання.
4. Скласти програми на алгоритмічній мові C згідно завдання.
5. Відлагодити програми, виконати обчислення, проаналізувати отримані результати.

**7.5. Зміст звіту** 1. Титульний аркуш.

1. Мета роботи.
2. Індивідуальне завдання.
3. Блок-схема алгоритмів у відповідності до завдання.
4. Тексти програм у відповідності до завдання.
5. Результати обчислень.
6. Аналіз результатів, висновки.

**7.6. Індивідуальні завдання**

Завдання 1 

Виконати завдання наведені нижче. Ввід-вивід даних та виконання інших окремих логічних дій необхідно реалізувати в окремих функціях. У головній функції необхідно виконувати лише їх виклик. Використання глобальних змінних не допускається. Інформація повинна передаватися у функції лише за допомогою параметрів.

1. Використовуючи метод половинного ділення знайти корінь рівняння: . Пошук здійснити на інтервалі [1,3]. Точність вводити з клавіатури.
2. Використовуючи метод лінійного пошуку знайти корінь рівняння: . Розв’язок лежить в інтервалі [0.4,0.8]. Точність вводити з клавіатури.
3. Використовуючи метод половинного ділення знайти корінь рівняння: . Розв’язок лежить в інтервалі [–0.5,0.5]. Точність вводити з клавіатури.
4. Використовуючи метод лінійного пошуку знайти корінь рівняння: . Розв’язок лежить в інтервалі [–0.5,0.5]. Точність вводити з клавіатури.
5. Обчислити означений інтеграл функції: 1 + cos2x, на інтервалі [0,π], використовуючи формулу лівих прямокутників. Крок вводити з клавіатури.
6. Обчислити означений інтеграл функції: sin3x – 3x, на інтервалі [0,π], використовуючи формулу правих прямокутників. Крок вводити з клавіатури.
7. Обчислити означений інтеграл функції: , на інтервалі [2,3], використовуючи формулу середніх прямокутників. Крок вводити з клавіатури.
8. Обчислити означений інтеграл функції: , на інтервалі [0,1], використовуючи формулу лівих прямокутників. Крок вводити з клавіатури.
9. Обчислити означений інтеграл функції: , на інтервалі [1,6], використовуючи формулу правих прямокутників. Крок вводити з клавіатури.
10. Обчислити означений інтеграл функції: , на інтервалі [0.1,0.2], використовуючи формулу середніх прямокутників. Крок вводити з клавіатури.
11. Протабулювати суму  обчислену з точністю  на інтервалі  з кроком h. На основі отриманих результатів обчислити означений інтеграл цієї функції. Крок і точність обчислення ряду вводити з клавіатури.
12. Протабулювати суму  обчислену з точністю на інтервалі  з кроком. На основі отриманих результатів обчислити означений інтеграл цієї функції.
13. Протабулювати суму  обчислену з точністю  на інтервалі  з кроком h. На основі отриманих результатів обчислити означений інтеграл цієї функції. Крок і точність обчислення ряду вводити з клавіатури..
14. Протабулювати суму  обчислену з точністю  на інтервалі  з кроком h. На основі отриманих результатів обчислити означений інтеграл цієї функції. Крок і точність обчислення ряду вводити з клавіатури..
15. Протабулювати добуток  обчислений з точністюна інтервалі  з кроком h. На основі отриманих результатів обчислити означений інтеграл цієї функції. Крок і точність обчислення ряду вводити з клавіатури.
16. Протабулювати добуток  обчислений з точністю на інтервалі  з кроком h. На основі отриманих результатів обчислити означений інтеграл цієї функції. Крок і точність обчислення ряду вводити з клавіатури.

Завдання 2

Виконати аналіз текстового файлу (текст довільний). Ввід-вивід даних та виконання інших окремих логічних дій необхідно реалізувати в окремих функціях. У головній функції необхідно виконувати лише їх виклик. Використання глобальних змінних не допускається. Інформація повинна передаватися у функції лише за допомогою параметрів. Назва текстового файлу та інші вхідні дані задаються в тексті програми.

Вихідні дані виводяться на консоль.

1. Обчислити частоту розділових знаків (для кожного кількість та %).
2. Обчислити частоту малих літер (для кожної кількість та %).
3. Обчислити частоту великих літер (для кожної кількість та %).
4. Обчислити частоту цифр (для кожної кількість та %).
5. Визначити кількість знаків з пробілами і без пробілів.
6. Визначити кількість слів.
7. Визначити кількість рядків та абзаців.
8. Визначити, скільки разів входить задане користувачем слово.
9. Визначити довжину найдовшого слова кожного рядка, вивести слова та їх довжини.
10. Визначити довжину найкоротшого слова кожного рядка, вивести слова та їх довжини.
11. Обчислити частоту слів (для кожного кількість та %).
12. Обчислити кількість слів по абзацах.
13. Вивести всі слова, що починаються з літер, які вказані користувачем (слова не дублювати, вивести кожне окреме слово та кількість його входжень).
14. Поміняти місцями перший та другий абзаци.
15. Замінити усі входження слова, вказаного користувачем на інше слово, що також вказане користувачем.
16. Вивести всі слова тексту відсортовані за алфавітом (слова не дублювати, вивести кожне окреме слово та кількість його входжень).