**Практична робота №9. Написання програм з використанням бінарних файлів для введення/виведення даних**

**Мета**: навчитися писати програми з використанням з використанням файлів для введення/виведення даних сивів **на мові С++** в **консольному режимі.**

**Завдання.**

1. Прочитати (повторити) основні теоретичні відомості. Переглянути лекції №№10-11
2. Запустити середовище програмування С++ . Записати програму, що виконує 2 завдання (пп.3,4) шляхом виклику відповідної функції. В функціях повинен проводитися аналіз результатів звернення до файлу з обробленням помилок. В першому рядку програми записати

*// Група № Прізвище*

вказавши номер своєї групи та своє прізвище

1. Перша функція виконує такі дії:

* Запитує фразу;
* Записує її до бінарного файлу;
* Для перевірки запису файл відкривається для читання, зчитується інформація та виводиться на консоль;
* Запитує номер позиції для читання та кількість літер;
* Через команди прямого доступу зчитується запитаний фрагмент та виводиться на консоль.

1. Друга функція виконує такі дії:

- записує в бінарний файл дані про сканер із приведеної нижче структури scan\_info. Структура файлу: у перших двох байтах розміщується значення типу int, що визначає кількість зроблених у файл записів; далі без пропусків розміщаються записи про сканери. Дані про сканер можна занести ініціалізацією (відповідно, створивши масив структур) або ввести з клавіатури - 3–4 записи. Інформація по кожному сканеру перед записом до файлу виводиться на консоль;

- запитує номер потрібного запису, зчитує потрібну інформацію та виводить на консоль.

Для збереження даних про планшетні сканери описати структуру виду:

struct scan\_info{

char model[25]; // найменування моделі

int price; // ціна

double x\_size; // гориз. розмір обл. сканування

double y\_size; // вертик. розмір обл. сканування

int optr; // оптичне розрішення

int grey; // число градацій сірого

};

1. Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді cpp-файлу з іменем у форматі

**<Номер групи><Номер лабораторної><Прізвище англійською>**

Наприклад, 31-01Ivanov.cpp.

Іншим рішенням є надсилання поштою посилання на текст програми за URL адреси, яку надає C++Shell, вказавши в темі листа, номер групи прізвище студента та номер ПР.

В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№8".

**Строк відсилки ПР для МІВТ/ЕТ-41 17.11.2020**

**МНТ/ЕТ- 41 17.11.2020.**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, В темі листа вказати, номер групи, прізвище студента та номер ПР як "ПР№8" В темі листа вказати, номер групи, прізвище студента, номер ПР та фразу "Запитання".

**Увага!** C++Shell не підтримує роботу з файлами.

Для роботи з файлами можна використати безкоштовний компілятор Dev-C++, який потрібно встановити на ваш комп’ютер. Скачати його можна за посиланням:

**https://sourceforge.net/projects/orwelldevcpp/**

Інструкція по роботі з компілятором надається за посиланням (але там забагато реклами та іншого сміття):

**https://studopedia.ru/18\_64672\_pokrokove-vikonannya-programi.html**

Якщо у Вас нема можливості встановити цю програму, то просто запишіть код виконання операцій роботи з файлами та надішліть на перевірку.

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Структуровані файли**

Структурований файл – це окремий випадок двійкового файлу. В якості порції даних в таких файлах використовують структури. Для ініціалізації, відкриття, зчитування та запису даних до структурованих файлів застосовуються ті ж самі функції,які використовуються при роботі з двійковими файлами. Структуровані файли дозволяють за одне звернення записати/зчитати всі поля структури, що значно пришвидшує роботу з файлом.

Приклад запису до файлу структури, що містить 2 поля – ціле (b.n - 4 байта) і дійсне (b.r - 4 байта):

**#include<stdio.h>**

**#include<windows.h>**

**#include"math.h"**

**int main()**

**{ system("color F0");**

**FILE\*f1;** // оголошення вказівника на файл

**struct{** // опис структури даних

**int n;**

**float r;} b;**

// відкриття файлу для запису

**f1 = fopen("file1","wb");**

**printf("Write to file");**

**for (int j = 1; j < 6; j++)**

**{**// заповнення полів структури

**b.n = j;b.r = sqrt(float(j));**

**//** запис елементу структури до файлу

**fwrite(&b, sizeof(b), 1, f1);**

**printf("\n%d %f", b.n, b.r);**

**}**

**fclose(f1); //** закрити файл

**printf("\n");**

**//** відкриття файлу для зчитування

**f1 = fopen("file1","rb");**

**printf("Read from file\n");**

**int k = 0; //** 0–для 1-го блоку

**while(!feof(f1)) //** доки не досягли кінця файлу

**{//** встановити вказівник перед зчитуваним блоком

**fseek(f1, (k)\*sizeof(b), SEEK\_SET);**

**//** зчитати блок даних розміром sizeof(b) – 1 елемент структури

**if(fread(&b, sizeof(b), 1, f1))**

**//** вивести результат на екран

**printf("%d %f\n", b.n, b.r);**

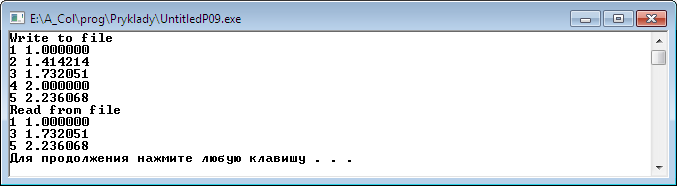
**k+=2; //** пропустити запис

**}**

**system("pause");**

**return 0;}**

Результат



***Пригадаємо.***

Обробка інформації, що зберігається у вигляді файлу передбачає наступні дії:

* визначення змінної – файлового покажчика;
* відкривання та закривання потоку;
* введення-виведення (символів, рядків, форматованих даних, порцій даних певної довжини);
* аналіз можливих помилок операцій введення-виведення;
* керування буферізацією потоку (розміром буферу);
* керування буферним покажчиком.

Потоки бувають двох типів: текстові та двійкові.

Кожний потік має керівну структуру типу FILE , що містить усю необхідну інформацію для роботи з ним. Змінна, що буде представляти потік визначається як покажчик на структуру типу FILE.

Наприклад:  
 FILE \*fp;

Змінна fp зображує потік у подальшій роботі з файлом.

Опис типу FILE , а також прототипи більшості функцій, макросів та констант файлової системи містяться у заголовному файлі <cstdio> (та <stdio.h>).

Потік можна відкрити для читання або/та запису в текстовому або двійковому режимі. Функція відкриття потоку має формат:

FILE \*fopen(const char \*filename, const char \*mode);

Якщо відкривання було успішним, функція повертає *покажчик файлу*, що містить всю необхідну для роботи з потоком інформацію; інакше функція повертає NULL.

Параметр filename – задає ім`я файлу у вигляді рядка в *стилі С*.

Параметр mode – визначає режим відкривання файлу.

* “r” – відкриття існуючого файлу для читання;
* “w” – створення нового файлу для запису (якщо файл з таким ім`ям існує – він перезаписується);
* “a” – відкриття існуючого файлу для додавання в його кінець нової інформації;
* “r+” – відкриття існуючого файлу для читання й запису;
* “w+” – створення нового файлу для читання й запису (перезаписується файл з таким ім`ям існує, якщо існує);
* “a+” – відкриття існуючого файлу для читання та додавання в його кінець нової інформації.

Режим може містити символи t – текстовий, або b – двійковий. По умовчанню передбачається текстовий режим (t).

Приклади

FILE \*fp;

fp = fopen(“file1.txt”, “a+”);

FILE \*f = fopen(“c:\\temp\\data.dat”, “rb+”);

FILE \*fp;

if (fp = fopen(“file2.txt”, “r”) == NULL)

{perror(“ERROR open file2.txt”); return 1;}

FILE \*fp = fopen(“.\\data.dat”, “w”);

Потік закривається або при завершенні програми, або явно функцією fclose:  
int fclose(FILE \* );

Якщо потік, заданий параметром-покажчиком файлу, був відкритий для запису, то перед закриттям у файл записуються данні, що містяться у буферах потоку. У випадку успішного закриття функція повертає 0, інакше – EOF.

Рекомендується завжди явно закривати потоки, що відкриті для запису, щоб уникнути втрати даних.  
 int fflush(FILE \* ); - примусово скидає буфер у файл.

Максимальна кількість одночасно відкритих файлів визначається макросом FOPEN\_MAX.

Операції введення/виведення завжди виконуються починаючи з поточної позиції потоку, що визначається *покажчиком потоку*. *Покажчик потоку* встановлюється при відкритті на початок, або кінець (в залежності від режиму відкриття) й змінюється автоматично при виконанні операцій введення/виведення. Поточне положення *покажчика потоку* можна отримати функціями ftell, fgetpos й задати функціями fseek, fsetpos . Ці функції не можна використовувати для *стандартних потоків*.

**Запис потоку байтів**

fwrite – записує об`єкт у файл, відкритий у двійковому режимі. Функція повертає кількість записаних об`єктів. *Формат*:  
size\_t fwrite(const void \*Buf, size\_t Size, size\_t Count,   
FILE \*fp);

Buf – покажчик на об`єкт;

Size – розмір об`єкту у байтах;

Count – кількість об`єктів, що записуються;

fp – файловий-покажчик.

#include <cstdio>

#include <iostream>

using namespace std;

int main() {

FILE \*fp = fopen("d:\\Tmp\\data1.dat", "wb");

struct Point {

int x, y;

} point;

point.x = 10; point.y = 20;

int x = fwrite(&point, sizeof(Point), 1, fp);

cout << x << endl;

system("pause"); return 0;

}

**Читання потоку байтів**

fread – зчитує об`єкт з файлу, відкритого у двійковому режимі. Функція повертає кількість зчитаних об`єктів. *Формат*:  
size\_t fread(void \*Buf, size\_t Size, size\_t Count, FILE \*fp);

Buf – покажчик на об`єкт;

Size – розмір об`єкту у байтах;

Count – кількість об`єктів, що записуються;

fp – файловий-покажчик.

**Обробка помилок**

Функції роботи з потоком повертають значення, які потрібно аналізувати й обробляти помилкові ситуації. Крім того використовують функції:

int feof(FILE \*) – повертає ненульове значення, якщо досягнутий кінець файлу й 0 у протилежному випадку.

int ferror(FILE \*) – повертає ненульове значення, якщо виявлена помилка введення/виведення й 0 у протилежному випадку.

До функцій потрібно звертатись відразу після виконання операцій з файлом.

**Функції для роботи з буферним покажчиком**

Дозволяють здійснювати читання та запис у файл з довільної позиції. Використовуються в основному для файлів відкритих у двійковому режимі.

long ftell(FILE \*fp) – повертає поточну позицію покажчика (при помилці ).

int fgetpos(FILE \*fp, fpos\_t \*pos) – записує поточну позицію покажчика у другий параметр, повертає 0, якщо помилок не має, інакше ненульове значення.

void rewind(FILE \*fp) – встановлює покажчик на початок файлу.

int fsetpos(FILE \*fp, const fpos\_t \*pos) – встановлює покажчик за параметром pos , повертає 0, якщо помилок не має, інакше ненульове значення.

int fseek(FILE \*fp, long offset, int origin) – встановлює покажчик згідно зміщення offset відносно позиції origin.

Для параметру origin можуть бути вказані макроси:  
SEEK\_SET – початок файлу;  
SEEK\_CUR – поточна позиція покажчика;  
SEEK\_END – кінець файлу.