Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение   
высшего образования   
«Владимирский государственный университет   
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Институт прикладной математики, физики и информатики

Кафедра физики и прикладной математики

ЛАБОРАТОРНАЯ РАБОТА №5  
по дисциплине  
«Основы программирования»  
на тему:  
«Работы с файлами»

Выполнил:  
ст. гр. ПМИ-123

Рушев А.М.  
  
  
Принял:  
ст. преподаватель   
каф. ФиПМ  
Шишкина М.В.

Владимир,2024

**Цель работы**

Изучение способов работы с файлами, чтение и запись данных из файлов с помощью функции языка С++.

**Постановка задачи**

1. Реализовать сортировку слиянием двух одномерных упорядоченных целочисленных массивов. Упорядоченные массивы даны в файлах, необходимо записать их в третий файл с сохранением упорядоченности.
2. Дан текстовый файл, подсчитать и вывести на экран количество символов в файле до первого пробела.

**Теоретическая часть**

Под файлом понимают поименованную последовательность данных, расположенных на внешнем носителе, завершающуюся маркером конца файла EOF.

По способу доступа файлы можно разделить на последовательные, в этом случае чтение и запись производятся последовательно байт за байтом, и файлы с производным доступом, допускающие чтение и запись в указанную позицию.

При работе с файлами используют термин поток. Под потоком понимают логическую абстракцию, представляющую собой последовательность байтов. Такой подход позволяет привести к единообразию работу по чтению и записи данных. При этом стандартные устройства ввода/вывода, такие как дисплей и клавиатура, рассматриваются как потоки.

Работа с потоком может вестись в двух режимах: текстовом и бинарном. В случае бинарного потока данные рассматриваются так, как они хранятся в памяти, т. е. как последовательность битов. В случае текстовых файлов данные рассматриваются как строки, заканчивающиеся символом '\n'. В текстовом режиме прочитанная из потока комбинация символов CR (возврат каретки) и LF (перевод строки) преобразуется в один символ конца строки '\n'. При записи в поток в текстовом режиме осуществляется обратное преобразование, т. е. символ '\n' преобразуется в комбинацию CR и LF. Если файл хранит не текстовую, а произвольную двоичную информацию, то такие преобразования не нужны.

Режим потока определяется при установлении связи с файлом. В языке программирования С++ работа с файлами организована с помощью методов стандартных классов. Работа с файлами через стандартные классы языка программирования С++ в данном пособии не рассматривается, этот вопрос будет изучаться позднее в курсе объектно-ориентированного программирования. Поэтому далее будут рассмотрены функции для работы с файлами, унаследованные от языка программирования С. Именно эти функции следует использовать при выполнении лабораторной работы № 5.

Информация о файле, необходимая для работы с ним, содержится в стандартной структуре типа FILE, описанной в stdio.h, там же определены функции для работы с файлами. Поэтому если в программе планируется работа с файлами, необходимо подключить соответствующий заголовочный файл, используя директиву include #include <stdio.h>;

Для организации работы с файлами необходимо объявить указатель на файл следующим образом:

FILE \* идентификатор;

Например, так: FILE\* F;

Функция открытия потока получает на вход два параметра: имя файла и режим доступа к данным и имеет следующий формат:

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

При успешном открытии функция возвращает указатель на структуру типа FILE. В противном случае функция возвращает NULL. Более безопасной считается функция fopen\_s, имеющая следующую сигнатуру:

errno\_t fopen\_s(FILE\*\* pFile, const char \*filename,

const char \*mode);

Функция fopen\_s принимает три параметра: указатель на файл имя файла и режим работы с файлом. Возвращает нуль в случае удачного завершения работы или код ошибки – в противном случае.

Функции с постфиксом '\_s', считаются более безопасными в вопросе защиты данных; эти функции выполняют дополнительные проверки параметров и возвращают коды ошибок. Однако ошибки не могут быть исправлены автоматически, поэтому необходимо анализировать возвращаемое функцией значение.

Функции, использование которых может привести к ошибкам безопасности, так как их работа не предотвращает операции, которые могут перезаписывать значения в памяти, считаются устаревшими. Их использовать не рекомендуется. По умолчанию компилятор выдаёт предупреждение об устаревании функции, если в коде присутствует такая функция, и предлагает заменить её перегруженными шаблонами C++, которые помогают упростить переход к более безопасным вариантам. Так, вместо функции fopen предпочтительнее использовать функцию fopen\_s. Ниже приведён пример работы с этой функцией:

FILE\* pf;

errno\_t errFile3;

if ((errFile3=fopen\_s(&pf, "c:\\fRezult.txt", " r

")) != 0) {printf("Не удалось открыть файл"); return 0;

}

В приведённом примере объявлен указатель pf, далее осуществлена попытка его связи с файлом fRezult.txt, расположенным на диске С, и открытие потока для чтения. Если файл открыть не удалось, на экране будет отображено сообщение "Не удалось открыть файл".

Режим открытия файла:

"r" – открытие файла для чтения;

"w" – открытие пустого файла для записи;

"a" – открытие файла для добавления элементов в его конец;

"r+" – открытие файла для чтения и записи, при этом файл должен существовать;

"w+" – открытие пустого файла для чтения и записи; если файл с указанным именем уже существует, его информация стирается;

"a+" – открытие файла для чтения и добавления информации в его конец.

Режим открытия файла может содержать ещё один символ:

t – текстовый файл или b – двоичный файл.

По умолчанию файл открывается в текстовом режиме. При вводе в поток информация накапливается в буфере до его заполнения или до закрытия потока.

После завершения работы с файлом поток должен быть закрыт. Для этого используют функцию int fclose(FILE\* pf).

Для корректного завершения предыдущего примера поток необходимо закрыть следующим образом:

fclose(pf);

45

Рассмотрим основные функции для работы с потоками ввода/вывода, унаследованные от языка С.

Чтение/запись потока байтов fread()/fwrite()

size\_t fread(void\* буфер, size\_t число\_байту size\_t

объем, FILE \*fp);

size\_t fwrite(const void \*буфер, size\_t число\_байт,

size\_t объем, FILE \*fp);

Пример использования функции fread:

FILE \*pf;

char content[100];

pf = fopen("the\_file.txt", "rb");

fread(&content, sizeof(char), 100,pf);

Функция fread\_s считывает данные из потока. Это версия функции fread с усовершенствованиями, касающимися безопасности работы с данными

size\_t fread\_s(void \*buffer, size\_t bufferSize,

size\_t elementSize, size\_t count, FILE \*pf);

Функция fread\_s возвращает то количество элементов, которое было считано в буфер. Возвращаемое значение может быть меньше количества, указанного для чтения элементов. Это может произойти в случае, если возникла ошибка чтения или конец файла был обнаружен раньше достижения заданного для чтения количества элементов.

Чтобы отличить ошибку, возникшую при чтении из файла от условия конца файла, следует использовать функцию feof или ferror. Пример использования функции fread\_s:

FILE \*pf;

int numread = fread\_s( list, BUFFERSIZE,

ELEMENTSIZE, ELEMENTCOUNT, stream );

Функция int ferror(FILE \*pf) проверяет, имеются ли файловые ошибки в потоке pf. Если возвращаемое значение равно нулю – ошибки отсутствуют; отличная от нуля величина указывает на наличие ошибки.

46Пример использования функции fwrite:

FILE \*stream;

char list[30];

. . .

int numwritten = fwrite( list, sizeof( char ),

DATASIZE, stream );

fclose( stream );

Для более наглядной иллюстрации работы рассмотренных функций приведём пример их работы в связке

char ch[10];

for (int i = 0; i < 10; i++) ch[i] = i + 33;

FILE\* pf;

errno\_t errF1\_1;

if(errF1\_1=open\_s(&pf,"c:\\file\_new.txt","w") !=0)

{

printf("Not File"); return 0;

}

else

{

printf("open 'W'\n");

fwrite(ch, sizeof(char), 10, pf);

fclose(pf);

}

if(errF1\_1=fopen\_s(&pf,"c:\\file\_new.txt","r")!=0)

{printf("Not File"); return 0;

}

else {

printf("open 'r'\n");

char \*c=new char[10];

if (!ferror(pf)) { fread\_s(c, 10, 1, 10, pf); }

}

fclose(pf);

}

В приведённом выше примере объявлен указатель на массив ch, состоящий из десяти символов. Массив проинициализирован последовательностью из символов, начиная с символа под номером 33. Под номером 33 в таблице ASCII (прил. 7) находится символ '!'. Далее объявлен указатель на файл pf. Осуществляется попытка создания файла

file\_new.txt, связывание его с указателем pf и открытие потока на запись. Если создание и открытие произвести не удалось, на экране будет отображена надпись "Not File" и произведён выход из функции.

В случае успеха на экране будет отображена надпись "open 'W'", говорящая пользователю о том, что файл успешно открыт для записи. Таким образом, подготовительный этап для записи данных в файл успешно завершён, и следующее действие – запись данных в файл при помощи функции fwrite. В файл, связанный с указателем pf, будут записаны десять элементов из массива ch размером sizeof(char), после чего поток будет закрыт и вновь открыт на чтение. Если открытие прошло успешно, в массив символов с из потока pf будет считана последовательность из десяти элементов размером один байт, после чего поток будет вновь закрыт.

Реализуем рассмотренный выше пример при помощи функций чтения и записи символа из потока.

Чтение символа из потока реализуется с помощью функции fgetc. Запись символа в поток реализуется с помощью функций fputc.

Функция int (FILE\* stream) возвращает следующий за текущей позицией символ во входном потоке stream и даёт приращение указателю положения в файле. Символ считывается как данные типа unsigned char, преобразованные к переменной целого типа.

При достижении конца файла fgetc() возвращает константу – EOF, но поскольку EOF имеет значение целого типа, при работе с двоичными файлами для контроля достижения конца файла необходимо использовать feof(). Если fgetc() обнаруживает ошибку, то также возвращается значение EOF

FILE\* pf;

errno\_t errF;

if (errF = fopen\_s(&pf, "c:\\file\_new.txt","w")!=0)

48

{

printf("Not F1"); return 0;

}

else

{

printf("open 'W'\n");

for (int i=0; i<10; i++) { fputc(i + 33, pf); }

fclose(pf);

}

if (errF=fopen\_s(&pf,"c:\\file\_new.txt","r") != 0)

{

printf("NoT F1"); return 0;

}

else {

printf("open 'r'\n");

while (!feof(pf)) {

int c = fgetc(pf);

if (c!=EOF ) printf\_s("%c\t", c);

}

fclose(pf);

}

В приведённом примере стоит обратить внимание на дополнительную проверку на конец файла перед выводом считанного значения. Проверка обусловлена тем, что функция fgetc(pf) возвращает значение, следующее за значением текущей позиции в файле, поэтому

условия продолжения цикла недостаточно. На рис. 2 приведён результат работы рассмотренного фрагмента программного кода. К такому же результату при выводе данных приводит предыдущий пример и два разобранных ниже.

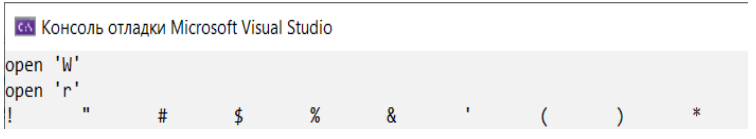


Рис. 1. Результат примера использования функций

Рассмотрим функции чтения/записи строки в файл. Функция fgets() реализует чтение строки из указанного потока. Функция fputs() реализует запись строки в указанный поток char \*fgets(char \*str, int длина, FILE \*fp); int fputs(const char \*str, FILE \*fp);

Далее приведён пример использования этих функций для рассмотренной выше задачи. Для корректной работы функции чтения строки из файла в исходный массив добавлен символ завершения строки.

Результат работы приведённого ниже фрагмента кода такой же, как на рис. 3. char ch[11]; /\*объявление массива символов, заполнение объявленного строкой выше массива\*/

for (int i = 0; i < 10; i++) ch[i] = i + 33;

ch[10] = '\0';

FILE\* pf; //объявление указателя на файл

errno\_t errF;

//открытие файла для записи

if (errF=fopen\_s(&pf,"c:\\file\_new.txt", "w") != 0)

{//вывод информативного сообщения в случае неудачи открытия

printf("Not F1"); return 0;

}

else

{//вывод информативного сообщения в случае удачного открытия

printf("open 'W'\n");

fputs(ch, pf); //запись строки

fclose(pf); //закрытие потока

}

//открытие файла для чтения

if (errF=fopen\_s(&pf, "c:\\file\_new.txt", "r")!= 0)

{//вывод информативного сообщения в случае неудачи открытия

printf("NoT F1"); return 0;

}

else

{//создание массива, в который будет записана строка

char ch[126];

/\*вывод информативного сообщения в случае удачного открытия\*/

printf("open 'r'\n");

//запись строки в массив и вывод строки из массива на экран

if (fgets(ch, 126, pf)) printf\_s("%s\t", ch);

}

fclose(pf);//закрытие потока

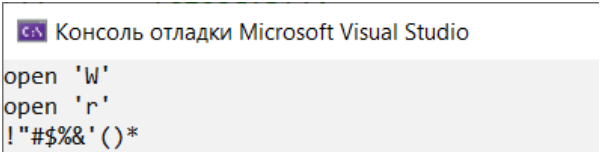


Рис. 3. Результат примера использования функций fgets() и fputs

Из рис. 3 видно, что на экране отображены те же символы, что и на рис. 2, но без пробелов. Это объясняется тем, что данные выведены как строка, а в предыдущем примере данные выводились посимвольнои были разделены управляющим символом табуляции \t.

В ряде случаев удобнее использовать функции форматированного ввода/вывода из потока.

Форматированный ввод fscanf(), Форматированный вывод fprintf().

int fscanf(FILE \*fp, char \*format, ...);

int fprintf(FILE \*fp, char \*format, ...);

Реализуем задачу, рассмотренную выше, с помощью этих функций

char ch[11];

for (int i = 0; i < 10; i++) ch[i] = i + 33;

ch[10] = '\0';

FILE\* pf;

errno\_t errF;

if (errF = fopen\_s(&pf, "c:\\file\_new.txt", "w") !=

0) {

printf("Not F1"); return 0;

}

51

else {

printf("open 'W'\n");

{if( fprintf(pf, ch)); }

fclose(pf);

}

if (errF=fopen\_s(&pf,"c:\\file\_new.txt","r") != 0)

{

printf("NoT F1"); return 0;

}

else

{char ch[126];

printf("open 'r'\n");

if (fscanf\_s(pf, "%s", ch, \_countof(ch)))

printf\_s("%s", ch);

}

fclose(pf);

\_countof возвращает количество элементов массива.

Из разобранных выше примеров очевидно, что одну и ту же задачу можно решить, используя различные функции языка программирования С++. При выборе функции для решения конкретной практической задачи следует руководствоваться спецификой каждой из функций и личными предпочтениями разработчика.

Значения, возвращаемые функциями работы с потоком, необходимо анализировать в программе на корректность работы, для того чтобы бвовремя отследить возможные ошибки, например, при открытии файла.

При чтении данных из файла работа часто ведётся до его завершения. Для анализа файла на завершение используют функцию feof. int feof(FILE\*) возвращает нуль, если достигнут конец файла, и отличное от нуля значение – в противном случае.

**Практическая часть**

**Задание 1.**

**Листинг кода:**

//инициализация массива

int InputMass(int \*arr, int n) {

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

arr[i] = 1 + rand() % 50;

}

return \*arr;

}

//вывод массива

void PrintMass(int \*arr, int n) {

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

cout << arr[i] << "\t";

}

cout << endl;

}

//сортировка массива

void SortMass(int\* arr, int n) {

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

for (int j = 0; j < n - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

//запись массива в файл

void WriteToFile(int\* arr, int n, char\* filePath) {

FILE\* \_writeToFile;

errno\_t err = fopen\_s(&\_writeToFile, filePath, "w");

if (\_writeToFile != NULL || err != 0) {

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

fprintf(\_writeToFile, "%d\n", arr[i]);

}

fclose(\_writeToFile);

cout << "Массив записан в файл успешно!" << endl;

}

else {

cout << "Не удалось открыть файл!" << endl;

}

}

void WriteToFileForTask(FILE\* file, FILE\* outFile) {

int num;

while (fscanf\_s(file, "%d", &num) != EOF) {

fprintf\_s(outFile, "%d", num);

}

}

//запись слияния массива в файл

void MergeWriteToFile(char\* filePath1, char\* filePath2, char\* filePath3) {

FILE\* file1, \* file2, \* outFile;

fopen\_s(&file1, filePath1, "r");

fopen\_s(&file2, filePath2, "r");

if (file1 == NULL || file2 == NULL) {

cout << "Ошибка открытия отдельных файлов!" << endl;

return;

}

fopen\_s(&outFile, filePath3, "w");

int num1, num2;

if (fscanf\_s(file1, "%d", &num1) == EOF) {

fclose(file2);

fopen\_s(&file2, filePath2, "r");

while (fscanf\_s(file2, "%d", &num2) != EOF) {

fprintf\_s(outFile, "%d\t", num2);

}

cout << "Данные из 2 файла записаны в общий файл!" << endl;

return;

}

else if (fscanf\_s(file2, "%d", &num2) == EOF) {

fclose(file1);

fopen\_s(&file1, filePath1, "r");

while (fscanf\_s(file1, "%d", &num1) != EOF) {

fprintf\_s(outFile, "%d\t", num1);

}

cout << "Данные из 1 файла записаны в общий файл!" << endl;

return;

}

else {

while (true) {

if (num1 <= num2) {

fprintf(outFile, "%d ", num1);

if (fscanf\_s(file1, "%d", &num1) == EOF) {

fprintf(outFile, "%d\t ", num2);

break;

}

}

else {

fprintf(outFile, "%d ", num2);

if (fscanf\_s(file2, "%d", &num2) == EOF) {

fprintf(outFile, "%d\t", num1);

break;

}

}

}

}

fclose(file1);

fclose(file2);

fclose(outFile);

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №5" << endl;

//Задание 1

const int n1 = 5;

const int n2 = 4;

int arr1[n1];

int arr2[n2];

cout << "Первый массив: ";

InputMass(arr1, n1);

PrintMass(arr1, n1);

cout << endl;

cout << "Второй массив: ";

InputMass(arr2, n2);

PrintMass(arr2, n2);

cout << endl;

cout << "Сортировка первого массива: ";

SortMass(arr1, n1);

PrintMass(arr1, n1);

cout << endl;

cout << "Сортировка второго массива: ";

SortMass(arr2, n2);

PrintMass(arr2, n2);

cout << endl;

//Задание с файлами

char filePath1[] = "Array1.txt";

char filePath2[] = "Array2.txt";

char filePath3[] = "ArrayAll.txt";

WriteToFile(arr1, n1, filePath1);

WriteToFile(arr2, n2, filePath2);

MergeWriteToFile(filePath1, filePath2, filePath3, n1, n2);

system("pause");

return 0;

}

**Результат выполнения кода программы:**

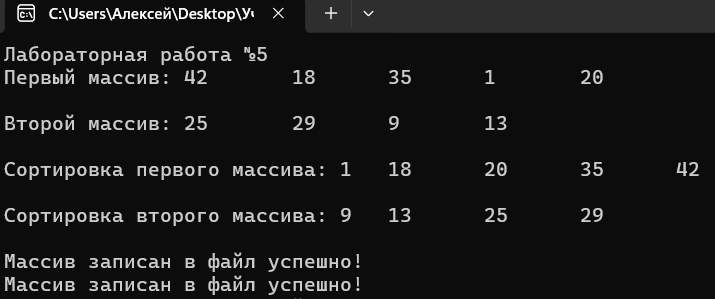
****

Рисунок 1 – Результат выполнения кода задания 1. Часть1

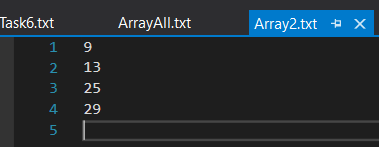


Рисунок 2 – Результат выполнения кода задания 1. Часть 2

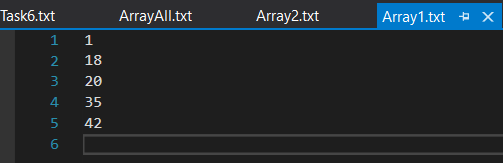


Рисунок 3 – Результат выполнения кода задания 2. Часть 3

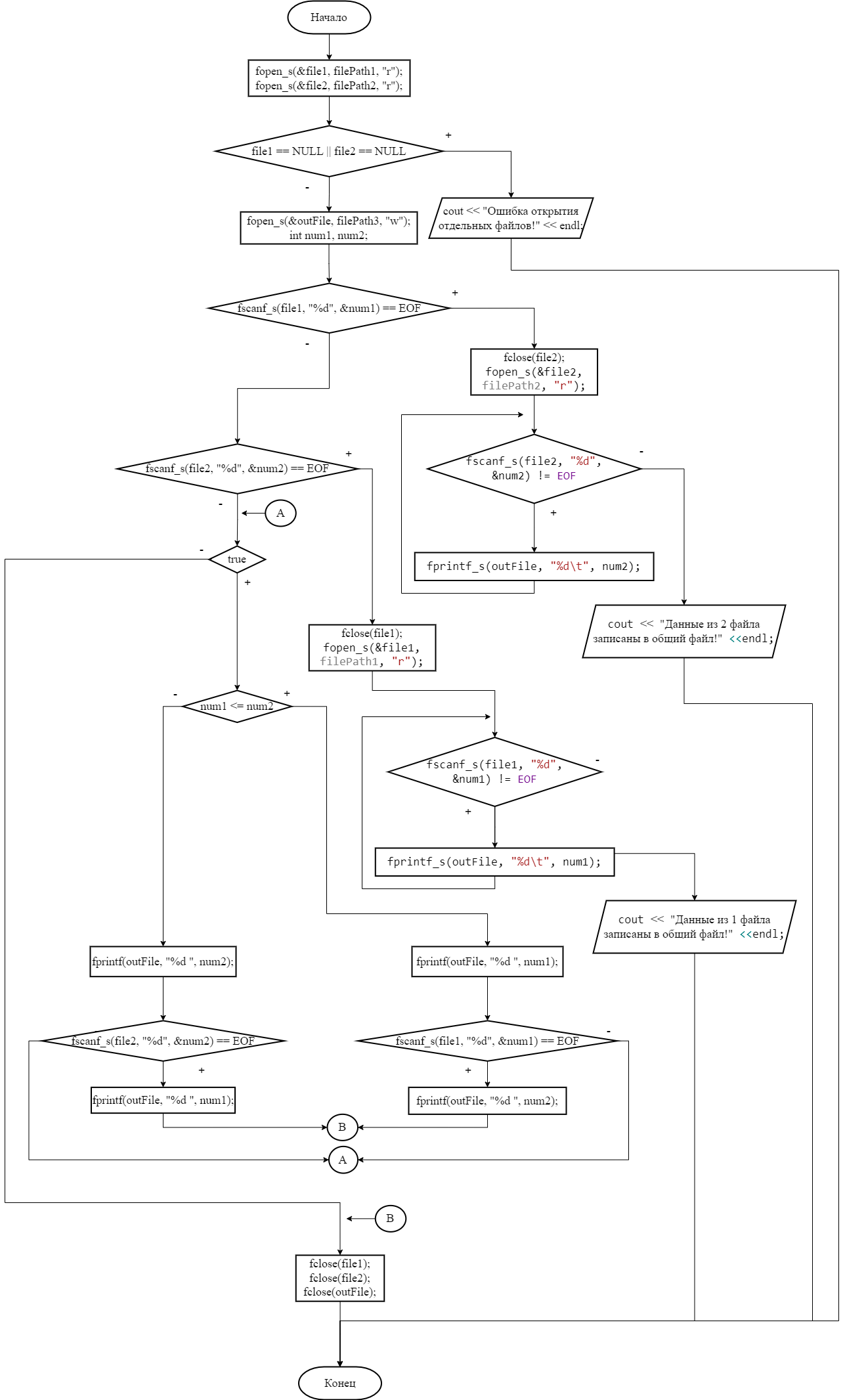


Рисунок 4 – Блок-схема задания 1

**Задание 2.**

**Листинг кода:**

int Task6\_WriteToFile(char\* filePath) {

FILE\* task6;

fopen\_s(&task6, filePath, "r");

if (task6 == NULL) {

cout << "Ошибка открытия файла, он пустой" << endl;

return -1;

}

int \_count = 0;

char ch;

while ((ch = fgetc(task6)) != EOF && !isspace(ch)) {

\_count++;

}

fclose(task6);

cout << "Количество символов в файле до первого пробела: " << \_count << endl;

return \_count;

}

int main()

{

setlocale(LC\_ALL, "rus");

cout << "Лабораторная работа №5" << endl;

//Задание 1

const int n1 = 5;

const int n2 = 4;

int arr1[n1];

int arr2[n2];

cout << "Первый массив: ";

InputMass(arr1, n1);

PrintMass(arr1, n1);

cout << endl;

cout << "Второй массив: ";

InputMass(arr2, n2);

PrintMass(arr2, n2);

cout << endl;

cout << "Сортировка первого массива: ";

SortMass(arr1, n1);

PrintMass(arr1, n1);

cout << endl;

cout << "Сортировка второго массива: ";

SortMass(arr2, n2);

PrintMass(arr2, n2);

cout << endl;

//Задание с файлами

char filePath1[] = "Array1.txt";

char filePath2[] = "Array2.txt";

char filePath3[] = "ArrayAll.txt";

WriteToFile(arr1, n1, filePath1);

WriteToFile(arr2, n2, filePath2);

MergeWriteToFile(filePath1, filePath2, filePath3, n1, n2);

system("pause");

return 0;

}

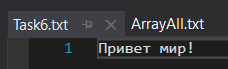


Рисунок 5 – Текст задания 2.

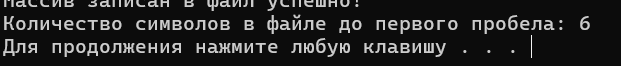


Рисунок 6 – Результат выполнения кода задания 2

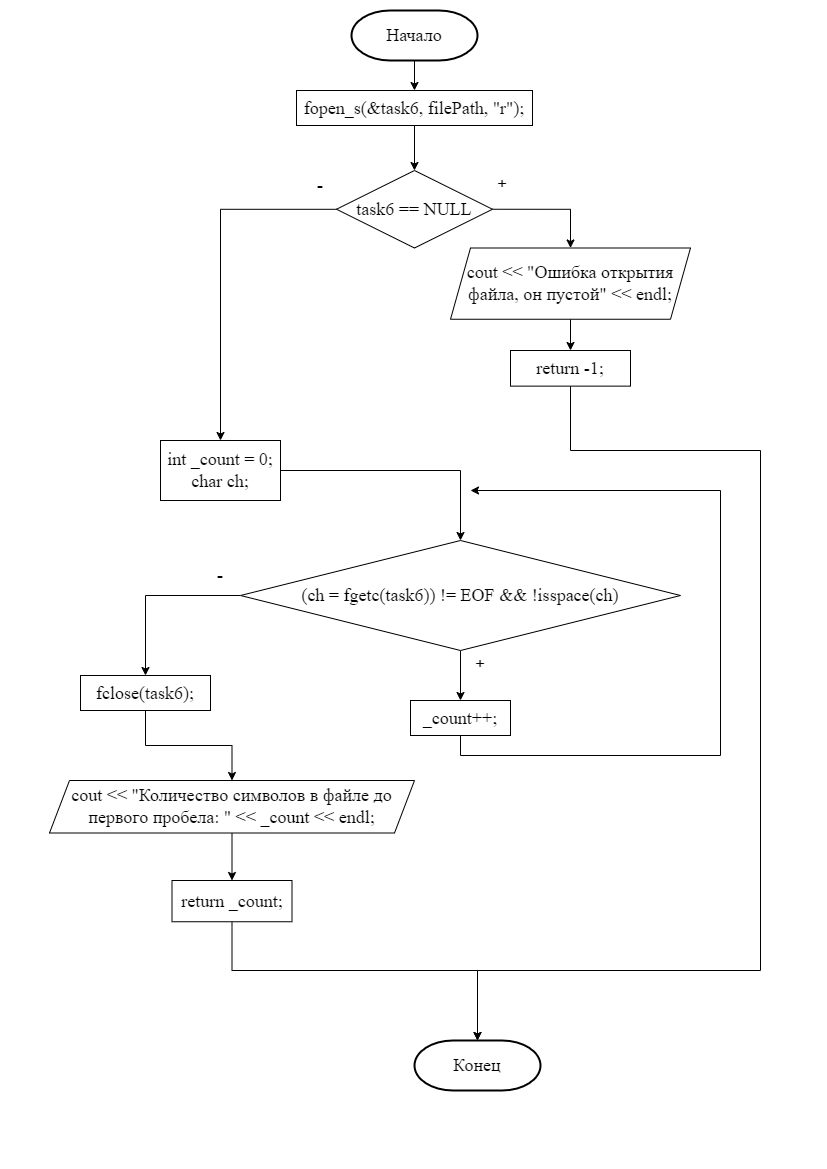


Рисунок 7 – Блок-схема задания 2

**Вывод:** в ходе выполнения лабораторной работы по работе с файлами на C++ были освоены основные операции с файлами: открытие, закрытие, чтение и запись данных.

Преимущества при работе с файлами: Файлы позволяют хранить данные даже после завершения работы программы, что позволяет сохранять информацию между запусками программы. Файлы позволяют обмениваться данными между различными программами или устройствами. Файлы позволяют сохранять информацию на долгий срок, что полезно для архивирования данных.

Недостатки при работе с файлами: Чтение и запись данных в файл может быть медленнее по сравнению с работой с данными в памяти. Файлы могут иметь ограничения по размеру, что может быть проблемой при работе с большими объемами данных. Зависимость от файловой системы: Работа с файлами требует доступа к файловой системе операционной системы, что может создавать проблемы при переносе программы на другие платформы. При работе с файлами всегда существует риск потери данных из-за ошибок ввода-вывода или повреждения файла.