Лабораторная работа 3

Работа с текстовыми файлами

Работа с текстовым файлом похожа работу с консолью: с помощью функций форматированного ввода мы сохраняем данные в файл, с помощью функций форматированного вывода считываем данные из файла. Есть множество нюансов, которые мы позже рассмотрим.

Основные операции, которые необходимо проделать, это:

1. Открыть файл, для того, чтобы к нему можно было обращаться. Соответственно, открывать можно для чтения, записи, чтения и записи, переписывания или записи в конец файла и т.п. Когда вы открываете файл, может также произойти куча ошибок – файла может не существовать, это может быть файл не того типа, у вас может не быть прав на работу с файлом и т.д. Всё это необходимо учитывать.

2. Непосредственно работа с файлом - запись и чтение. Здесь также нужно помнить, что мы работаем не с памятью с произвольным доступом, а с буферизированным потоком, что добавляет свою специфику.

3. Закрыть файл. Так как файл является внешним по отношению к программе ресурсом, то если его не закрыть, то он продолжит висеть в памяти, возможно, даже после закрытия программы (например, нельзя будет удалить открытый файл или внести изменения и т.п.). Кроме того, иногда необходимо не закрывать, а "переоткрывать" файл для того, чтобы, например, изменить режим доступа.

Функция fopen сама выделяет память под объект, очистка проводится функцией fclose.

Функция fopen может открывать файл в текстовом или бинарном режиме. По умолчанию используется текстовый.

Если необходимо открыть файл в бинарном режиме, то в конец строки добавляется буква b, например “rb”, “wb”, “ab”, или, для смешанного режима

“ab+”, “wb+”, “ab+”. Вместо b можно добавлять букву t, тогда файл будет открываться в текстовом режиме.

Самый распространенный связан со структурой FILE (это не класс, потому что сущность языка С). Эта структура определена в заголовочном файле стандартной библиотеки <stdio.h>. Размер этой структуры и ее поля зависят от ОС и от версии компилятора. Поэтому никто не пользуется структурой FILE. Обычно пользуются указателем на эту структуру: FILE*. Например:

```
FILE *file = fopen("file1.txt", "r");
```

fopen -- функция из стандартной библиотеки. Первый параметр - имя файла (в текущем каталоге). Второй параметр задает режим открытия файла; в данном случае "r" означает, что файл будет открыт только для чтения. Эта функция возвращает ненулевой указатель, если открытие прошло успешно; и возвращает NULL, если произошла ошибка. Ошибка может возникать в следующих ситуациях:

- • не существует файла;
- • у программы недостаточно прав доступа для работы с файлом;

Для дальнейшей корректной работы следует писать примерно такой код:

```
if (file == NULL)
{
	// файл не удалось открыть
}
else
{
	// Работа с файлом
}
```

Допустим, что нам удалось открыть файл, т.е. f != NULL. Тогда для того, чтобы считывать файл, можно использовать функцию:

```
fscanf(file, "%s", ptr);
```

Эта функция работает аналогично функции scanf.

Закрытие файла

Это можно сделать с помощью функции:

```
fclose(file);
```

Ввиду механического устройства жесткого диска, данные в файл попадают не сразу. Сначала данные записываются в так называемый буфер (область оперативной памяти), и когда он переполнится, то данные из буфера будут записаны в файл. Такая схема придумана для ускорения работы с файлами. На самом деле, буфер - это поле структуры FILE: указатель на массив char'ов.

Если мы напишем fprintf(...), то запись произведется в буфер. И только тогда, когда буфер будет заполнен до конца, он будет сразу весь записан на жесткий диск. По этой причине, если мы не закроем файл функцией fclose(f), то последние данные из буфера не запишутся в файл. Отсутствие этой команды может привести к потере данных в файле, который был открыт для записи (дозаписи).

Если не закрывать файлы (которые открыты даже для чтения), то это может привести к ограничению доступа к файлу для других программ. Какие именно ограничения наложатся - это зависит от ОС.

Но в ОС Windows если файл открыт на чтение и не закрывается, то из другой программы его нельзя удалить.

В любой ОС есть ограничение на количество одновременно открытых файлов. И это еще одна причина для закрытия файлов.

Текстовые и бинарные файлы

Рассмотрим строку:

```
fopen(f, "file1.txt", "w");
```

Существует несколько способов прочитать/записать файл. Например:

fopen("file1.txt", "wt") - откроет файл как текстовый файл;
fopen("file1.txt", "wb") - откроет файл как бинарный файл.

Разница заключается лишь в том, что символы переноса строк запишутся по разному.

Рассмотрим пример в Windows:

Исходная строка кода выглядит так: fprintf("Hello\n");

Откроем в Windows файл на запись с параметром "wb" (как бинарный файл). Это означает, что в него запишется в точности то, что мы передали в функции fprintf. Тогда в файл запишутся ровно 6 байт: *Hello\10*

А теперь мы откроем в Windows файл на запись с параметром "wt" (как текстовый файл). Тогда в файл запишутся ровно 7 байт: *Hello\10\13*

Тут \10\13 означает символы перевода строки в ОС Windows.

Чтение данных из файла

Хороший способ чтения из файла дает функция fgets() (от "get string"):

```
char *fgets(char *buffer, size_t length, FILE *file);
```

Тут
buffer - это указатель на буфер, в который мы читаем;
length - это размер буфера;

file - это файл, из которого мы читаем (если читаем с клавиатуры, то разумно использовать stdin).

Функция возвращает строку

Эта функция делает примерно следующее. Она читает из файла file в буфер buffer не больше length-1 символов. Функция может прочитать не все length-1 символов в том случае, если она встретит конец строки, либо конец файла. Функция читает length-1 символ потому, что последний символ функция добавляет сама - '\0'

Налицо быстрота и безопасность. Главное отличие от scanf'a заключается в том, что функция перестанет читать в тот момент, когда закончится буфер. Быстрота обусловлена тем, что функция scanf должна в момент выполнения разобрать форматную строку, в то время как fgets просто читает строку.

В то время как фунция fgets читеат обычную строку, функция scanf может читать и различные другие типы (целые, вещественные числа).

В языке C есть семейство функций ~ atoi (a -- ASCII, i -- integer):

N = atoi(string);

Функция принимает единственный параметр строку и пытается ее привести в типу int. Надо заметить, что функция atoi безопасная, но не очень удобная. Безопасная в том смысле, что не сломается: atoi("25a") == 25. "Неудобства" заключаются в том, то если мы передаем в качестве параметра строку, в которой есть не только числа, нужно быть очень внимательным и знать, как работает эта функция. Функция atoi никак не проинформирует нас, если преобразование прошло неудачно.

Рассмотрим следующий пример:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	FILE *file;
	char buffer[128];
	file = fopen("test.txt", "w");
	if(file == NULL)
	{
		fprintf(stdout,"Error open file!");
		exit(-1);
	}
	fprintf(file, "Hello, World!\n");
	fprintf(file, "It`s a second line.\n");
	fprintf(file, "Third line");
	fclose(file);
	file = fopen("test.txt", "r");
	if(file == NULL)
	{
		fprintf(stdout,"Error open file!");
		exit(-1);
	}
	int num = 1;
	while(!feof(file))
	{
		fgets(buffer, 127, file);
		printf("%s", buffer);
	}
	fclose(file);
}
```

Признак конца файла

Функция int feof (FILE * stream); возвращает истину, если конец файла достигнут. Функцию удобно использовать, когда необходимо пройти весь файл от начала до конца. Пусть есть файл с текстовым содержимым text.txt. Считаем посимвольно файл и выведем на экран.

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

```
int main()
{
	FILE *input = NULL;
	char c;
	input = fopen("test.txt", "rt");
	if (input == NULL)
	{
		printf("Error opening file");
		exit(0);
	}
	while (!feof(input))
	{
		c = fgetc(input);
		printf("%c", c);
	}
	fclose(input);
}
```

Примеры:

В файле записаны целые числа. Найти максимальное из них. Воспользуемся тем, что функция fscanf возвращает число верно прочитанных и сопоставленных объектов. Каждый раз должно возвращаться число 1.

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
int main()
{
	FILE *input = NULL;
	int num, maxn, hasRead;
	input = fopen("nums.txt", "r");
	if (input == NULL)
	{
		printf("Error opening file");
		exit(0);
	}
	maxn = INT_MIN;
	while (!feof(input))
	{
		fscanf(input, "%d", &num);
		printf("%d\n", num);
		if (num > maxn)
		{
			maxn = num;
		}
```

```
        }
        printf("max number = %d", maxn);
        fclose(input);
}
```

Задания на ЛБ.

Общая структура заданий:

Первая программа:

1. С клавиатуры пользователем вводится число элементов массива.
2. Выделяется память заданного размера.
3. Массив заполняется случайными числами.
4. Выводится на экран заполненный массив.
5. Полученный массив записывается в файл.
6. Закрывается файл

Вторая программа

1. Открывается файл
2. Читаются хранящиеся в файле числа в массив.
3. Выполняется обработка массива.
4. В случае изменения его размера выполняется перераспределение памяти.
5. Выводится на экран содержимое измененного массива.
6. Полученный массив записывается в новый файл.
7. Освобождается память, закрываются файлы.

Задача.

1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-40,30]. Удалить из него все элементы, которые состоят из одинаковых цифр (включая однозначные числа).

2. Вставить число *K* перед всеми элементами, в которых есть цифра 1.

3. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра четная, а само число делится на нее.

4. Вставить элемент со значением *K* до и после всех элементов, заканчивающихся на цифру *K*.

5. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-35,75]. Удалить из него все элементы, первая цифра которых четная.

6. Вставить число *K1* после всех элементов, больших заданного числа, а число *K2* – перед всеми элементами, кратными трем.

7. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-45,95]. Удалить из него все элементы кратные 7 и принадлежащие промежутку [*a*, *b*].

8. Вставить число *K* между всеми соседними элементами, которые имеют одинаковые знаки.

9. Переставить в обратном порядке часть массива между элементами с номерами *K1* и *K2*, включая их.

10. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, в записи которых есть цифра 5.

12. Поменять местами первый положительный и последний отрицательный элементы.

13. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-40,30]. Удалить из него все четные элементы, у которых последняя цифра 2.

14. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра четная, а само число делится на нее.

15. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, первая цифра которых четная.

16. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-30,30]. Удалить из него все элементы кратные 7 и принадлежащие промежутку [a, b].

17. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [10,100]. Удалить из него все элементы, в записи которых есть цифра 3.

18. Вставить число K перед всеми элементами, в которых есть цифра 2.

Задача 2.

1. Вставить максимальный элемент массива после всех элементов, в которых есть цифра 1.

2. Переставить первые k и последние k элементов местами, сохраняя порядок их следования.

3. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра нечетная, а само число кратно 3.

4. Вставить элемент со значением K после всех четных элементов, начинающихся на цифру K.

5. Вставить число $K1$ после всех элементов, больших заданного числа, а число $K2$ – после всех элементов, кратных пяти.

6. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-135,175]. Удалить из него все элементы, первая и последняя цифра которых четная.

7. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-95,95]. Удалить из него все отрицательные элементы кратные 5 и принадлежащие промежутку [a,b].

8. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-50,50]. Удалить из него все элементы, в записи которых последняя цифра равна 0.

9. Вставить значение минимального элемента массива после всех четных элементов.

10. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-140,140]. Удалить из него все элементы, у которых первая и вторая цифры одинаковые.

11. Переставить первые два и средние два элемента местами, сохраняя порядок их следования (количество элементов – четное).

12. Вставить максимальное значение элементов массива перед всеми элементами, в записи которых есть цифра 1.

13. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-200,500]. Удалить из него все элементы, в записи которых есть цифра 0.

14. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, в которых последняя цифра нечетная, а само число кратно 2.

15. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [100, 275]. Удалить из него все элементы, первая и последняя цифра которых нечетная.

16. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-65,65]. Удалить из него все отрицательные элементы кратные 3 и принадлежащие промежутку $[a,b]$.

17. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-100,100]. Удалить из него все элементы, у которых первая и вторая цифры разные.

18. Вставить значение минимального элемента массива после всех нечетных элементов.