ПЗ-04: **Работа с текстовыми файлами** (4 ч.)

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 рассмотрим ниже.

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.

**Кроме того**, существует ряд задач, когда нам не нужно обращаться к содержимому файла: **переименование, перемещение, копирование** и т.д.

В стандарте **Си** нет описания функций для этих нужд.

Они, безусловно, имеются для каждой из реализаций компилятора.

Считывание содержимого каталога (папки, директории) – это тоже обращение к файлу, потому что каталог сам по себе является файлом с метаинформацией.

Иногда необходимо выполнять некоторые *вспомогательные операции*: **переместиться в нужное место файла**, **запомнить текущее положение**, **определить длину файла** и т.д.

Для работы с файлом необходим объект **FILE**. Этот объект хранит ***идентификатор*** *файлового потока* и информацию, которая нужна, чтобы им управлять, включая указатель на его буфер, индикатор позиции в файле и индикаторы состояния.

Объект **FILE** сам по себе является **структурой**, но к его полям не должно быть доступа. Переносимая программа должна работать с файлом как с абстрактным объектом, позволяющим получить доступ до файлового потока.

Создание и выделение памяти под объект типа **FILE** осуществляется с помощью функции **fopen** или **tmpfile** (есть и другие, но мы остановимся только на этих).

|  |  |
| --- | --- |
| Функция **fopen** открывает **файл**. Она получает **два аргумента**:  – строку с адресом файла  – строку с режимом доступа к файлу.  Имя файла может быть (м.б.), как абсолютным, так и относительным.  **fopen** возвращает **указатель** на объект **FILE**, с помощью которого далее можно осуществлять доступ к файлу.  **FILE\* fopen(const char\* filename,**  **const char\* mode);**  *Схожая функция* ***freopen*** *библиотеки* ***Си*** *выполняет аналогичную операцию после первого закрытия любого открытого потока, связанного с её параметрами*:  **FILE\* freopen(const char \*path, const char \*mode, FILE \*fp);**  Функция **fopen** сама выделяет память под объект, очистка проводится функцией **fclose**. Закрывать файл обязательно, самостоятельно он не закроется.  Функция **fopen** может открывать файл в текстовом или бинарном режиме.  *По умолчанию* используется **текстовый**. | Например, откроем файл и запишем в него *Hello World*:  #include <stdio.h>  int main()  {  FILE \*fp; // *Declaring a variable of the FILE type*  char name[ ] = "test.txt";  // *Checking for an error when opening a stream:*  if ((fp = fopen(name, "w")) == NULL)  {  printf("Could not open the file");  getchar();  return 0;  }  // *The file was opened successfully:*  printf("Managed to open the file");  fprintf(fp, "Hello, World!");  fclose(fp);  getchar();  return 0;  } |

Режим доступа может быть следующим:

|  |  |
| --- | --- |
| **Тип** | **Описание** |
| **r** | Чтение.  Файл должен существовать. |
| **w** | Запись нового файла.  Если файл с таким именем уже существует, то его содержимое будет потеряно. |
| **a** | Запись в конец файла. Операции позиционирования (**fseek**, **fsetpos**, **frewind**) игнорируются.  Файл создаётся, если не существовал. |
| **r+** | Чтение и обновление. Можно как читать, так и писать.  Файл должен существовать. |
| **w+** | Запись и обновление. Создаётся новый файл. Если файл с таким именем уже существует, то его содержимое будет потеряно. Можно как писать, так и читать. |
| **a+** | Запись в конец и обновление. Операции позиционирования работают только для чтения, для записи игнорируются. Если файл не существовал, то будет создан новый. |

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. Это зависит от реализации. В новом стандарте **си** (2011) буква **x** означает, что функция **fopen** должна завершиться с ошибкой, если файл уже существует. Дополним нашу старую программу: заново откроем файл и считаем, что мы туда записали.

Вместо функции **fgets** можно было использовать **fscanf**, но нужно помнить, что она может считать строку только до первого пробела: **fscanf(file, "%127s", buffer);**

|  |  |
| --- | --- |
| #include <stdio.h>  void main()  {  FILE \*fp;  char buffer[128];  char name[ ] = "test.txt";  if ((fp = fopen(name, "w")) == NULL)  {  printf("Could not open the file");  getchar();  return 0;  }  // The file was opened successfully:  printf("Managed to open the file");  fp = fopen(name, "w");  fprintf(fp, "Hello, World!");  fclose(fp);  // Reading from a file  fp = fopen(name, "r");  fgets(buffer, 127, fp);  fclose(fp);  printf("\n%s", buffer);  getchar();  } | Также, вместо того, чтобы **открывать** и **закрывать** **файл** можно воспользоваться функцией **freopen**, которая «**переоткрывает**» файл с новыми правами доступа:  \* \* \*  fp = fopen(name, "w");  fprintf(file, "Hello, World!");  freopen(name, "r", fp);  fgets(buffer, 127, fp);  printf("%s", buffer);  fclose(fp);  \* \* \*  Функции **fprintf** и **fscanf** отличаются от **printf** и **scanf** только тем, что принимают в качестве первого аргумента указатель на **FILE**, в который они будут выводить или из которого они будут читать данные.  Здесь стоит сразу же добавить, что функции **printf** и **scanf** могут быть без проблем заменены функциями **fprintf** и **fscanf**.  ***Ошибка открытия файла***  *Если вызов функции* ***fopen*** *прошёл неудачно, то она возвратит* ***NULL****. Ошибки во время работы с файлами встречаются достаточно часто, поэтому каждый раз, когда мы открываем файл, необходимо проверять результат работы.* |
| В ОС (рассматриваем самые распространённые ОС) существует три стандартных потока:  1. стандартный поток вывода stdout,  2. стандартный поток ввода stdin  3. стандартный поток вывода ошибок stderr.  Они автоматически открываются во время запуска приложения и связаны с консолью. | void main()  {  int a, b;  fprintf(stdout, "Enter two numbers\n");  fscanf(stdin, "%d", &a);  fscanf(stdin, "%d", &b);  if (b == 0)  {  fprintf(stderr, "Error: divide by zero");  } else  {  fprintf(stdout, "%.3f", (float) a / (float) b);  }  getchar();  } |

**Буферизация данных**

Как уже говорилось ранее, когда мы выводим данные, они сначала помещаются в буфер.

Очистка буфера осуществляется:

1) Если он заполнен

2) Если поток закрывается

3) Если мы явно указываем, что необходимо очистить буфер (здесь тоже есть исключения:)).

4) Также очищается, если программа завершилась удачно. Вместе с этим закрываются и все файлы. В случае ошибки выполнения этого может не произойти.

**Форсировать выгрузку буфера** можно с помощью вызова функции **fflush(File \*)**.

Рассмотрим два примера – с очисткой и без.

|  |  |
| --- | --- |
| Раскомментируйте вызов **fflush**.  Во время выполнения откройте текстовый файл и посмотрите на поведение.  Буфер файла можно назначить самостоятельно, задав свой размер.  Делается это при помощи функции:  **void setbuf (FILE \* stream, char \* buffer);**  которая принимает уже открытый **FILE** и указатель на новый буфер.  Размер нового буфера должен быть не меньше чем **BUFSIZ** (к примеру, на текущей рабочей станции **BUFSIZ** равен **512** байт).  Если передать в качестве буфера **NULL**, то поток станет небуферизированным.  Можно также воспользоваться функцией:  **int setvbuf ( FILE \* stream, char \* buffer, int mode, size\_t size );**  которая принимает буфер произвольного размера **size**. | #include <stdio.h>  void main()  {  FILE \*fp;  char c;  char name[ ] = "test.txt";  fp = fopen(name, "w");  fprintf(stdout, "Enter any characters (Enter);  q = exit!!!\n");  do {  c = getchar();  fprintf(fp, "%c", c);  fprintf(stdout, "%c", c);  **//fflush(fp);**  } while(c != 'q');  fclose(fp);  getchar();  } |
| Функция **fflush** «сбрасывает» накопленные данные в поток данных.  **fflush** – «проталкивание» накопленных данных в поток данных. | Синтаксис **fflush**:  #include < stdio.h >  int fflush (FILE \*stream); |

*Аргументы*: **stream** – указатель на управляющую таблицу потока данных.

Возвращаемое значение: 0 – при успешном завершении работы функции.

EOF (обычно -1) – в случаи ошибки.

Описание: Функция **fflush** сбрасывает в связанный с потоком данных файл данные, находящиеся в буфере.

Режим **mode** может принимать следующие значения

**\_IOFBF** - полная буферизация. Данные записываются в файл, когда он заполняется. На считывание, буфер считается заполненным, когда запрашивается операция ввода и буфер пуст.

**\_IOLBF** - линейная буферизация. Данные записываются в файл, когда он заполняется, либо, когда встречается символ новой строки. На считывание, буфер заполняется до символа новой строки, когда запрашивается операция ввода и буфер пуст.

**\_IONBF** – без буферизации. В этом случае параметры **size** и **buffer** игнорируются.

В случае удачного выполнения функция возвращает 0.

|  |  |  |  |
| --- | --- | --- | --- |
| **Пример**:  Зададим свой буфер и посмотрим, как осуществляется чтение из файла.  Пусть файл короткий (что-нибудь, типа Hello, World!), и считываем мы его посимвольно.  Видно, что данные уже находятся в буфере.  Считывание посимвольно производится уже из буфера. | | #include <stdio.h>  void main()  {  FILE \*input = NULL;  char c;  char buffer[BUFSIZ \* 2] = {0};  input = fopen("test.txt", "rt"); // файл короткий (Hello, World!)  setbuf(input, buffer);  while (!feof(input))  {  c = fgetc(input);  printf("%c\n", c);  printf("%s\n", buffer);  getchar();  }  fclose(input);  } | |
|  | |  | |
| **feof**  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 **test.txt**.  Считаем посимвольно файл и выведем на экран.  Всё бы ничего, только функция **feof** работает неправильно...  Это связано с тем, что понятие "конец файла" не определено.  При использовании **feof** часто возникает ошибка, когда последние считанные данные выводятся два раза.  Это связано с тем, что данные записывается в буфер ввода, последнее считывание происходит с ошибкой и функция возвращает старое считанное значение. | | | #include <stdio.h>  void main()  {  FILE \*input = NULL;  char c;  input = fopen("test.txt", "rt");  if (input == NULL)  {  printf("Error opening file");  getchar();  exit(0);  }  while (!feof(input))  {  c = fgetc(input);  fprintf(stdout, "%c", c);  }  fclose(input);  getchar();  } |
| Этот пример (справа-ниже) сработает с ошибкой (скорее всего) и выведет последний символ файла два раза. | | | |
| #include <stdio.h>  void main()  {  FILE \*input = NULL;  char c;  input = fopen("test.txt", "rt");  if (input == NULL)  {  printf("Error opening file");  getchar();  exit(0);  }  while (fscanf(input, "%c", &c) == 1)  {  fprintf(stdout, "%c", c);  }  fclose(input);  getchar();  } | Решение (слева) –  не использовать **feof**.  Например, хранить общее количество записей или использовать тот факт, что функции **fscanf** и пр. обычно возвращают число верно считанных и сопоставленных значений. | | #include <stdio.h>  void main()  {  FILE \*input = NULL;  char c;  input = fopen("test.txt", "rt");  if (input == NULL)  {  printf("Error opening file");  getchar();  exit(0);  }  while (!feof(input))  {  fscanf(input, "%c", &c);  fprintf(stdout, "%c", c);  }  fclose(input);  getchar();  } |

*Чтение символа из файла:*

**char fgetc(поток);**

Аргументом функции является указатель на поток типа **FILE**. Функция возвращает код считанного символа.

Если достигнут конец файла или возникла ошибка, возвращается константа **EOF**.

*Запись символа в файл:*

**fputc(символ, поток);**

Аргументами функции являются символ и указатель на поток типа FILE.

Функция возвращает код считанного символа.

*Функции* **fscanf()** и **fprintf()** аналогичны функциям **scanf()** и **print**f**()**, но работают с файлами данных, и имеют первый аргумент — указатель на файл.

**fscanf(поток, "ФорматВвода", аргументы);**

**fprintf(поток, "ФорматВывода", аргументы);**

*Функции* fgets() и fputs() предназначены для ввода-вывода строк, они являются аналогами функций gets() и puts() для работы с файлами.

**fgets(УказательНаСтроку, КоличествоСимволов, поток);**

Символы читаются из потока до тех пор, пока не будет прочитан символ новой строки ‘**\n**’, который включается в строку, или пока не наступит конец потока **EOF** или не будет прочитано максимальное количество символов.

Результат помещается в указатель на строку и заканчивается нуль- символом ‘**\0**’.

Функция возвращает адрес строки.

|  |  |
| --- | --- |
| *Функция*  **fputs(УказательНаСтроку, поток);**  Копирует строку в поток с текущей позиции.  Завершающий нуль-символ не копируется.  **Пример**:  Ввести число (например: 3) и сохранить его в файле **s1.txt**.  Считать число из файла **s1.txt**, увеличить его на **3** и сохранить в файле **s2.txt**.  Результат выполнения — 2 файла: | #include <stdio.h>  int main()  {  FILE \*S1, \*S2;  int x, y;  printf("Enter a number: ");  scanf("%d", &x);  S1 = fopen("S1.txt", "w+t");  fprintf(S1, "%d", x);  fclose(S1);  S1 = fopen("S1.txt", "r");  S2 = fopen("S2.txt", "w");  fscanf(S1, "%d", &y);  y += 3;  fclose(S1);  fprintf(S2, "%d\n", y);  fclose(S2);  return 0;  } |

**NB**:

* ***Файл*** – *именованная область внешней памяти, выделенная для хранения массива данных. Данные, содержащиеся в файлах, имеют самый разнообразный характер: программы на алгоритмическом или машинном языке; исходные данные для работы программ или результаты выполнения программ; произвольные тексты; графические изображения и т. п*.

* ***Каталог*** (***директория***) – *именованная совокупность байтов на носителе информации, содержащая название подкаталогов и файлов, используется в файловой системе для упрощения организации файлов*.

* ***Файловой системой*** (**ФС**) *называется функциональная часть операционной системы, обеспечивающая выполнение операций над файлами. Примерами файловых систем являются FAT (FAT – File Allocation Table, таблица размещения файлов), NTFS, UDF (используется на компакт-дисках)*.

Например, файловая система FAT32 имеет следующую структуру:

