Файловый ввод / вывод

Файловые системы языков C и C++

- Язык C++ полностью поддерживает файловую систему языка C
- В языке C++ определена собственная объектно-ориентированная система ввода-вывода, включающая функции и операторы, которые полностью дублируют функциональные возможности средств ввода-вывода языка C

Потоки и файлы

В системе ввода/вывода С для программ поддерживается единый интерфейс, не зависящий от того, к какому конкретному устройству осуществляется доступ.

Система ввода-вывода создает между программистом и устройством абстрактное средство связи

Обобщенное устройство ввода или вывода или устройство более высокого уровня абстракции называется потоком

Конкретное физическое устройство называется файлом

Под файлом в языке С/С++ понимается все – это может быть файл на диске, консоль, дисплей, терминал, дисковод и принтер

Несмотря на разнообразие физических устройств, файловая система преобразовывает каждое из них в некое логическое устройство – поток.

Все потоки ведут себя похожим образом. И так как они в основном не зависят от физических устройств, то та же функция, которая выполняет запись в дисковый файл, может ту же операцию выполнять и на другом устройстве, например, на консоли.

Виды потоков

- Текстовый поток – это последовательность символов. В стандарте С считается, что текстовый поток организован в виде строк, каждая из которых заканчивается символом новой строки. Однако в конце последней строки этот символ не является обязательным. В текстовом потоке по требованию базовой среды могут происходить определенные преобразования символов. Например, символ новой строки может быть заменен парой символов — возврата каретки и перевода строки. Поэтому может и не быть однозначного соответствия между символами, которые пишутся (читаются), и теми, которые хранятся во внешнем устройстве. Кроме того, количество тех символов, которые пишутся (читаются), и тех, которые хранятся во внешнем устройстве, может также не совпадать из-за возможных преобразований.
- Бинарный поток – это последовательность байтов, которая взаимно однозначно соответствует байтам на внешнем устройстве, причем никакого преобразования символов не происходит. Кроме того, количество тех байтов, которые пишутся (читаются), и тех, которые хранятся на внешнем устройстве, одинаково. Однако в конце двоичного потока может добавляться определяемое приложением количество нулевых байтов. Такие нулевые байты, например, могут использоваться для заполнения свободного места в блоке памяти незначащей информацией, чтобы она в точности заполнила

Файлы

- Поток связывают с определенным файлом, выполняя операцию открытия. Как только файл открыт, можно проводить обмен информацией между ним и программой.
- Но не у всех файлов одинаковые возможности. Например, к дисковому файлу прямой доступ возможен, в то время как к некоторым принтерам – нет.
- Все потоки одинаковы, а файлы – нет.
- Если файл может поддерживать запросы на местоположение (указатель текущей позиции), то при открытии такого файла указатель текущей позиции в файле устанавливается в начало. При чтении из файла (или записи в него) каждого символа указатель текущей позиции увеличивается, обеспечивая тем самым продвижение по файлу.
- Файл отсоединяется от определенного потока (т.е. разрывается связь между файлом и потоком) с помощью операции закрытия. При закрытии файла, открытого с целью вывода, содержимое (если оно есть) связанного с ним потока записывается на внешнее устройство. Этот процесс, который обычно называют дозаписью потока, гарантирует, что никакая информация случайно не останется в буфере диска. Если программа завершает работу нормально, т.е. либо main() возвращает управление операционной системе, либо вызывается exit(), то все файлы закрываются автоматически. В случае аварийного завершения работы программы, например, в случае краха или завершения путем вызова abort(), файлы не закрываются.
- У каждого потока, связанного с файлом, имеется управляющая структура, содержащая информацию о файле; она имеет тип FILE.

Файловая система языка С

- Файловая система языка С состоит из нескольких взаимосвязанных функций. Самые распространенные из них показаны в табл. 1. Для их работы требуется заголовок <stdio.h>.
- Заголовок <stdio.h> предоставляет прототипы функций ввода/вывода и определяет следующие три типа: size_t, fpos_t и FILE. size_t и fpos_t представляют собой определенные разновидности такого типа, как целое без знака.
- Кроме того, в <stdio.h> определяется несколько макросов: NULL, EOF, FOPEN_MAX, SEEK_SET, SEEK_CUR и SEEK_END. Макрос NULL определяет пустой (null) указатель. Макрос EOF, часто определяемый как -1, является значением, возвращаемым тогда, когда функция ввода пытается выполнить чтение после конца файла. FOPEN_MAX определяет целое значение, равное максимальному числу одновременно открытых файлов. Другие макросы используются вместе с fseek() — функцией, выполняющей операции прямого доступа к файлу.
- *Указатель файла* — это указатель на структуру типа FILE. Он указывает на структуру, содержащую различные сведения о файле, например, его имя, статус и указатель текущей позиции в начало файла. Чтобы объявить переменную-указатель файла, используйте такого рода оператор:

Файловая система языка С

Имя	Что делает
fopen()	Открывает файл
fclose()	Закрывает файл
putc()	Записывает символ в файл
fputc()	То же, что и putc()
getc()	Читает символ из файла
fgetc()	То же, что и getc()
fgets()	Читает строку из файла
fputs()	Записывает строку в файл
fseek()	Устанавливает указатель текущей позиции на определенный байт файла
ftell()	Возвращает текущее значение указателя текущей позиции в файле
fprintf()	Для файла то же, что printf() для консоли
fscanf()	Для файла то же, что scanf() для консоли
feof()	Возвращает значение true (истина), если достигнут конец файла
ferror()	Возвращает значение true, если произошла ошибка
rewind()	Устанавливает указатель текущей позиции в начало файла
remove()	Стирает файл

Открытие файла

Функция fopen() открывает поток, связывает с этим потоком определенный файл и возвращает указатель этого файла.

Никогда не следует изменять значение этого указателя в программе. Если при открытии файла происходит ошибка, то fopen() возвращает пустой (null) указатель.

Прототип функции fopen():

```
FILE *fopen(const char *имя_файла, const char *режим);
```

где имя_файла — это указатель на строку символов, представляющую собой допустимое имя файла, в которое также может входить спецификация пути к этому файлу. Строка, на которую указывает режим, определяет, каким образом файл будет открыт.

Допустимые значения режима

Режим	*Что означает*
r	Открыть текстовый файл для чтения. Если файл не существует, работа fopen() завершится отказом
w	Создать текстовый файл для записи. Если файл не существует, то он будет создан. А если он существует, то содержимое, которое хранилось в нем до открытия, будет утеряно, причем будет создан новый файл.
a	Добавить в конец текстового файла . Если файл не существует, то он просто будет создан. Если существует, то все новые данные, которые записываются в него, будут добавляться в конец файла. Содержимое, которое хранилось в нем до открытия не будет.
rb	Открыть двоичный файл для чтения
wb	Создать двоичный файл для записи
ab	Добавить в конец двоичного файла
r+	Открыть текстовый файл для чтения/записи. Если файл не существует, то в режиме открытия r+ он создан не будет. Если файл уже существует, то содержимое останется нетронутым.
w+	Создать текстовый файл для чтения/записи. Если файл не существует, то файл будет создан. Если файл уже существует, то открытие приведет к утрате его содержимого.
a+	Добавить в конец текстового файла или создать текстовый файл для чтения/записи
r+b	Открыть двоичный файл для чтения/записи
w+b	Создать двоичный файл для чтения/записи
a+b	Добавить в конец двоичного файла или создать двоичный файл для чтения/записи

Примеры открытия файла

В следующем коде функция fopen() используется для открытия файла по имени TEST для записи.

```
FILE *fp;
fp = fopen("test", "w");
```

Следующий код при открытии файла проверяет наличие ошибки, например, защиту от записи или полный диск, причем проверка осуществляется еще до того, как программа попытается в этот файл что-либо записать.

```
FILE *fp;
 if ((fp = fopen("test","w"))==NULL) {
  printf("Ошибка при открытии файла.\n");
  exit(1);
}
```

Открытие файлов в различных режимах

- Файл можно открыть либо в одном из текстовых, либо в одном из двоичных режимов. В большинстве реализаций в текстовых режимах каждая комбинация кодов возврата каретки (ASCII 13) и конца строки (ASCII 10) преобразуется при вводе в символ новой строки. При выводе же происходит обратный процесс: символы новой строки преобразуются в комбинацию кодов возврата каретки (ASCII 13) и конца строки (ASCII 10). В двоичных режимах такие преобразования не выполняются.
- Максимальное число одновременно открытых файлов определяется FOPEN_MAX. Это значение не меньше 8, но чему оно точно равняется — это должно быть написано в документации по компилятору.

Закрытие файла

Функция fclose() закрывает поток, который был открыт с помощью вызова fopen()

Функция fclose() записывает в файл все данные, которые еще оставались в дисковом буфере, и проводит официальное закрытие файла на уровне операционной системы. Отказ при закрытии потока влечет всевозможные неприятности, включая потерю данных, испорченные файлы и возможные периодические ошибки в программе. Функция fclose() также освобождает блок управления файлом, связанный с этим потоком, давая возможность использовать этот блок снова.

Прототип функции fclose() такой:

```
int fclose(FILE *fp);
```

где *fp* — указатель файла, возвращенный в результате вызова fopen(). Возвращение нуля означает успешную операцию закрытия. В случае же ошибки возвращается EOF.

Запись символов

putc() и fputc()

Функция putc() записывает символы в файл, который с помощью fopen() уже открыт в режиме записи. Прототип этой функции следующий:

```
int putc(int ch, FILE *fp);
```

где *fp* — это указатель файла, возвращенный функцией fopen(), а *ch* — выводимый символ. Указатель файла сообщает putc(), в какой именно файл следует записывать символ. Хотя *ch* и определяется как int, однако записывается только младший байт.

Если функция putc() выполнилась успешно, то возвращается записанный символ. В противном же случае возвращается EOF.

Чтение символов

getc() и fgetc().

Функция getc() считывает символ из файла, открытого с помощью fopen()в режиме для чтения. Прототип этой функции следующий:

```
int getc(FILE *fp);
```

где *fp* — это указатель файла, имеющий тип FILE и возвращенный функцией fopen(). Функция getc() возвращает целое значение, но символ находится в младшем байте. Если не произошла ошибка, то старший байт (байты) будет обнулен.

Если достигнут конец файла, то функция getc() возвращает EOF.

Чтобы прочитать символы до конца текстового файла, можно использовать следующий код;

```
do {
  ch = getc(fp);
} while(ch!=EOF);
```

Однако getc() возвращает EOF и в случае ошибки. Для определения того, что же на самом деле произошло, можно использовать ferror().

```c
/*KTOD: программа ввода с клавиатуры на диск */
#include <stdio.h>
#include <stdlib.h>
 int main(int argc, char *argv[]) {
  FILE *fp;
  char ch;
   if(argc!=2) {
    printf("Вы забыли ввести имя
файла.\n");
    exit(1);
  }
   if((fp=fopen(argv[1], "w"))==NULL) {
    printf("Ошибка при открытии
файла.\n");
    exit(1);
  }
   do {
    ch = getchar();
    putc(ch, fp);
  } while (ch != '$');
   fclose(fp);
   return 0;
}
```

```c
/* DTOS: программа, которая читает файлы и
выводит их на экран. */
#include <stdio.h>
#include <stdlib.h>
 int main(int argc, char *argv[])
{
  FILE *fp;
  char ch;
   if(argc!=2) {
    printf("Вы забыли ввести имя
файла.\n");
    exit(1);
  }
   if((fp=fopen(argv[1], "r"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    exit(1);
  }
   ch = getc(fp);   /* чтение одного символа */
   while (ch!=EOF) {
    putchar(ch);  /* вывод на экран */
    ch = getc(fp);
  }
   fclose(fp);
   return 0;
}
```

Определение конца файла feof()

В языке С имеется функция feof(), которая определяет, достигнут ли конец файла.

Прототип функции feof() такой:

```
int feof(FILE *fp);
```

Если достигнут конец файла, то feof() возвращает true (истина); в противном же случае эта функция возвращает нуль.

Следующий код будет читать двоичный файл до тех пор, пока не будет достигнут конец файла:

```
while(!feof(fp)) ch = getc(fp);
```

Ясно, что этот метод можно применять как к двоичным, так и к текстовым файлам.

Чтение и запись строк

Функция fgets() читает строки символов из файла на диске

Функция fputs() записывает строки такого же типа в файл, тоже находящийся на диске

Эти функции работают почти как putc() и getc(), но читают и записывают не один символ, а целую строку.

Прототипы функций fgets() и fputs() следующие:

int fputs(const char *_cmp_, FILE *fp);

char *fgets(char *_cmp_, int _длина_, FILE *fp);

Функция fputs() пишет в определенный поток строку, на которую указывает *cmp*. В случае ошибки эта функция возвращает EOF.

Функция fgets() читает из определенного потока строку, и делает это до тех пор, пока не будет прочитан символ новой строки или количество прочитанных символов не станет равным *длина*-1. Если был прочитан разделитель строк, он записывается в строку, чем функция fgets() отличается от функции gets(). Полученная в результате строка будет оканчиваться символом конца строки ('0'). При успешном завершении работы функция возвращает *cmp*, а в случае ошибки — пустой указатель (null).

Пример чтения строк с клавиатуры и записи их в файл, который называется TEST

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void){
char str[80];  FILE *fp;
if((fp = fopen("TEST", "w"))==NULL) { printf("Ошибка при открытии файла.\n");
exit(1);  }
do {
printf("Введите строку (пустую - для выхода):\n");
gets(str);
strcat(str, "\n");  /* добавление разделителя строк */
 fputs(str, fp);  }
while(*str!='\n');
return 0;
}
```

Функция rewind()

Функция rewind() устанавливает указатель текущей позиции в файле на начало файла, указанного в качестве аргумента этой функции. Иными словами, функция rewind() выполняет "перемотку" (rewind) файла. Вот ее прототип:

```
void rewind(FILE *fp);
```

где *fp* — это допустимый указатель файла.

Функция rewind()

```
 int main(void)
{
  char str[80];   FILE *fp;
   if((fp = fopen("TEST", "w+"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    exit(1);
  }
   do {
    printf("Введите строку (пустую - для выхода):\n");
    gets(str);
    strcat(str, "\n");  /* ввод разделителя строк */
    fputs(str, fp);
  } while(*str!='\n');
   rewind(fp);  /* установить указатель  текущей позиции на начало файла. */
  while(!feof(fp)) {
    fgets(str, 79, fp);
    printf(str);
  }
   return 0;
}
```

Стирание файлов

Функция remove() стирает указанный файл. Вот ее прототип:

int remove(const char *имя_файла);

В случае успешного выполнения эта функция возвращает нуль, а в противном случае – ненулевое значение.

Дозапись потока

Для дозаписи содержимого выводного потока в файл применяется функция fflush(). Вот ее прототип:

```
int fflush(FILE *fp);
```

Эта функция записывает все данные, находящиеся в буфере в файл, который указан с помощью *fp*. При вызове функции fflush() с пустым (null) указателем файла *fp* будет выполнена дозапись во все файлы, открытые для вывода.

После своего успешного выполнения fflush() возвращает нуль, в противном случае – EOF.

Функции fread() и fwrite()

Для чтения и записи данных, тип которых может занимать более 1 байта, в файловой системе языка С имеется две функции: fread() и fwrite(). Эти функции позволяют читать и записывать блоки данных любого типа. Их прототипы следующие:

size_t fread(void *буфер, size_t колич_байт, size_t счетчик, FILE *fp);

size_t fwrite(const void *буфер, size_t колич_байт, size_t счетчик, FILE *fp);

Для fread() *буфер* – это указатель на область памяти, в которую будут прочитаны данные из файла. А для fwrite() *буфер* – это указатель на данные, которые будут записаны в файл. Значение *счетчик* определяет, сколько считывается или записывается элементов данных, причем длина каждого элемента в байтах равна *колич_байт*. (Вспомните, что тип size_t определяется как одна из разновидностей целого типа без знака.) И, наконец, *fp* – это указатель файла, то есть на уже открытый поток.

Функция fread() возвращает количество прочитанных элементов. Если достигнут конец файла или произошла ошибка, то возвращаемое значение может быть меньше, чем счетчик. А функция fwrite() возвращает количество записанных элементов. Если ошибка не произошла, то возвращаемый результат будет равен значению счетчик.

/* Запись не символьных данных в дисковый файл и последующее их чтение. */

```
int main(void) {
  FILE *fp;
  double d = 12.23;   int i = 101;   long l = 123023L;
   if((fp=fopen("test", "wb+"))==NULL) {  printf("Ошибка при открытии файла.\n");
    exit(1); }
   fwrite(&d, sizeof(double), 1, fp);
  fwrite(&i, sizeof(int), 1, fp);
  fwrite(&l, sizeof(long), 1, fp);
  rewind(fp);
  fread(&d, sizeof(double), 1, fp);
  fread(&i, sizeof(int), 1, fp);
  fread(&l, sizeof(long), 1, fp);
  printf("%f %d %ld", d, i, l);
  fclose(fp);
  return 0;
}
```

Пример использования **fread() и fwrite()** для чтения и записи данных пользовательского типа

Программа сохраняет адреса в файле. Адреса будут храниться в массиве структур следующего типа:

```
struct addr {
  char name[30];
  char street[40];
  char city[20];
  char state[3];
  unsigned long int zip;
} addr_list[MAX];
```

```
/* Сохранение списка. */
void save(void)
{
  FILE  *fp;
  register int i;
   if((fp=fopen("maillist", "wb"))==NULL) {
    printf("Ошибка при открытии файла.\n");
    return;
   }
   for(i=0; i<MAX; i++)
    if(*addr_list[i].name)
      if(fwrite(&addr_list[i], sizeof(struct addr), 1, fp)!=1)
          printf("Ошибка при записи файла.\n");
   fclose(fp);
}
```

Стандартные потоки

В начале выполнения программы автоматически открываются три потока:

stdin - стандартный поток ввода

stdout - стандартный поток вывода

stderr - стандартный поток ошибок

Обычно эти потоки направляются к консоли, но в средах, которые поддерживают перенаправление ввода/вывода, они могут быть перенаправлены операционной системой на другое устройство

Так как стандартные потоки являются указателями файлов, то они могут использоваться системой ввода/вывода языка С также для выполнения операций ввода/вывода на консоль.

Например, putchar() может быть определена таким образом:

```
int putchar(char c)
{
  return putc(c, stdout);
}
```

Вообще говоря, stdin используется для считывания с консоли, а stdout и stderr — для записи на консоль.

В роли указателей файлов потоки stdin, stdout и stderr можно применять в любой функции, где используется переменная типа FILE *.

Например, для ввода строки с консоли можно написать примерно такой вызов fgets():

```
char str[255];
fgets(str, 80, stdin);
```