Лекции по C/C++: работа с файлами (stdio.h)

В лекции рассмотрен классический способ работы с файлами в C/C++, основанный на библиотеке `stdio.h` и доступе к данным через структуру `FILE`. Альтернативный современный механизм работы с файлами в языке C++ на основе потоков и библиотек <fstream>, <ifstream>, <ofstream> будет изучен в следующей лекции.

Базовые функции для работы с файлами описаны в библиотеке `stdio.h`. Вся работа с файлом выполняется через *файловую переменную* - указатель на структуру типа `FILE`, определённую в стандартной библиотеке:

```
FILE *fp;
```

Открыть файл можно функцией `fopen`, имеющей 2 параметра:

```
FILE *fopen (char *имя_файла, char *режим_доступа)
```

Параметр `имя_файла` может содержать относительный или абсолютный путь к открываемому файлу:

1) `"data.txt"` - открывается файл `data.txt` из текущей папки

Важно: при запуске exe-файла "текущая папка" – та, где он находится; при отладке в IDE папка может быть иной, например, в Visual Studio при открытом консольном решении с именем Console файл следует разместить в папке `Console/Console`, а при запуске исполняемого файла не из IDE – в папке `Console/Debug`.

2) `"f:\\my.dat"` - открывается файл `my.dat` из головной папки диска `f:`

3) имя файла запрашивается у пользователя:

```
char buf[80];
printf ("\nведите имя файла:");
fflush (stdin);
gets (buf);
```

Параметр `режим_доступа` определяет, какие действия будут разрешены с открываемым файлом, примеры его возможных значений:

1) "rt" - открываем для чтения текстовый файл;

2) "r+b" - открываем для произвольного доступа (чтение и запись) бинарный файл;

3) "at" – открываем текстовый файл для добавления данных в конец файла;

4) "w" - открываем файл для записи без указания того, текстовый он или бинарный.

Фактически, указание "r" или "t" не накладывает каких-либо ограничений на методы, которые мы будем применять для чтения или записи данных.

После открытия файла следует обязательно проверить, удалась ли эта операция. Для этого есть 2 основных подхода:

1) стандартный обработчик ferror (см. пособие, п.8.7);

2) сравнить указатель, который вернула `fopen`, с константой `NULL` (`nullptr`) из стандартной библиотеки:

```
fp = fopen ("text.txt","r+b");
if (fp==NULL) {
 //Обработка ситуации "Не удалось открыть файл"
}
```

Пример. Приложение проверяет, удалось ли открыть файл из текущей папки, имя файла запрашивается у пользователя (Visual Studio)

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

int main(void) {
 setlocale(LC_ALL,"Rus"); SetConsoleCP(1251); SetConsoleOutputCP(1251);
 FILE *fp;
 char buf[80];
 printf ("\nвведите имя файла:");
 fflush (stdin);
 gets (buf);
 fp = fopen (buf,"r+b");
 if (fp==NULL) {
  printf ("\nне удалось открыть файл");
  getchar();
  exit (1); //Выйти с кодом завершения 1
 }

 fflush(stdin); getchar();  return 0;
}
```

Важно! Функции, возвращающие указатель, в том числе, `fopen`, считаются небезопасными в ряде новых компиляторов, например, Visual Studio 2015. Если их использование приводит не просто к предупреждению, а к генерации ошибок, есть 2 основных способа решения проблемы:

1) в соответствии с рекомендациями компилятора, заменить старые названия функций на их безопасные версии, например, `strcpy` на `strcpy_s` и `fopen` на `fopen_s`. При этом может измениться и способ вызова функций, например,

```
FILE *out; fopen_s(&out,"data.txt", "wt");
```

вместо

```
FILE *out = fopen_s("data.txt", "wt");
```

2) в начало файла (до всех `#include`) включить директиву

```
#define _CRT_SECURE_NO_WARNINGS
```

Если используется предкомпиляция, то можно определить этот макрос в заголовочном файле `stdafx.h`.

Выбор **способа чтения или записи данных** зависит от того, какой должна быть структура файла.

Если файл *форматированный*, то есть, является текстовым и состоит из лексем, разделённых стандартными разделителями (пробел, табуляция, перевод строки), обмен данными с ним можно выполнять методами:

- `fscanf` - для чтения
- `fprintf` - для записи

Первым параметром этих функций указывается файловая переменная, в остальном работа совпадает со стандартными `scanf` и `printf`.

Пример. Файл `text.txt` в текущей папке приложения имеет следующий вид:

```
1 1.5 -3.5
2 3.5
```

Прочитаем его как последовательность вещественных чисел.

```
FILE *fp = fopen ("text.txt","r");
if (fp==NULL) {
 printf ("\nне удалось открыть файл");  getchar(); exit (1);
}
float a;
while (1) {
 fscanf (fp,"%f",&a);
 if (feof(fp)) break; //Если файл кончился, выйти из цикла
 //здесь выполняется обработка очередного значения a, например:
 printf ("%.2f ",a);
}
fclose(fp);
```

Важно!

1. Функции семейства `scanf` возвращают целое число - количество значений, которые успешно прочитаны в соответствии с указанным форматом. В реальных приложениях эту величину следует проверять в коде:

```
int i=fscanf (fp,"%f",&a);
if (i!=1) {
 //не удалось получить 1 значение
}
```

2. На "восприятие" программой данных может влиять установленная в приложении локаль. Например, если до показанного кода выполнен оператор

```
setlocale(LC_ALL,"Rus");
```

результат работы кода может измениться (для русской локали разделителем целой и дробной части числа является запятая, а не точка).

3. Очередное чтение данных изменяет внутренний *файловый указатель*. Этот указатель в любой момент времени, пока файл открыт, показывает на следующее значение, которое будет прочитано. Благодаря этому наш код с "бесконечным" `while` не зациклился.

4. Код показывает, как читать из файла заранее неизвестное количество значений – это позволяет сделать стандартная функция `feof` (проверка, достигнут ли конец файла; вернёт не 0, если прочитано всё).

5. Распространённый в примерах из Сети код вида

```
 while (!feof(fp)) {
  fscanf (fp,"%f",&a);
  //обработка числа a
 }
```

в ряде компиляторов может породить неточности при интерпретации данных. Например, этот код может прочитать как последнее значение завершающий перевод строки в файле, благодаря чему последнее прочитанное значение "удвоится".

В качестве примера **форматной записи в файл** сохраним массив `a` из 10 целочисленных значений в файле с именем `result.txt` по 5 элементов в строке:

```
const int n=10;
int a[n],i;
FILE *fp=fopen ("result.txt","wt");
if (fp==NULL) {
 puts ("не удалось открыть файл на запись");
 getchar(); exit (1);
}
else {
 for (i=0; i<n; i++) a[i]=i+1;
 for (i=0; i<n; i++) {
  fprintf (fp,"%5d ",a[i]);
  if ((i+1)%5==0) fprintf (fp,"\n");
 }
 fclose (fp);
 //Закрыть файл, делать всегда, если в него писали!
}
```

Важно! Ввод/вывод функциями библиотеки `stdio.h` *буферизован*, то есть, данные "пропускаются" через область памяти заданного размера, обмен данными происходит не отдельными байтами, а "порциями". Поэтому перед чтением данных желательно очищать буфер от возможных "остатков" предыдущего чтения методом `fflush`, а после записи данных следует обязательно закрывать файл методом `fclose`, иначе данные могут быть потеряны. Заметим, что консольный ввод/вывод "обычными" методами `scanf` и `printf` также буферизован.

Теперь рассмотрим текстовый файл, состоящий из **неструктурированных строк** (абзацев) текста, разделённых символами перевода строки. При работе с такими данными могут потребоваться следующие функции:

- `fgetc` и `fputc` - для посимвольного чтения и посимвольной записи данных;
- `fgets` и `fputs` - для чтения и записи строк с указанным максимальным размером.

Как и в случае с функциями для чтения форматированных данных, у всех этих методов имеются аналоги для работы со стандартным вводом/выводом.

Пример. Читая файл, определить длину каждой строки в символах. Для решения задачи воспользуемся тем фактом, что строки завершаются символом "перевод строки" (`'\n'`). Предполагается, что файл уже открыт для чтения.

```
int c; int len=0,cnt=0;
while (1) {
 c=fgetc(fp);
 if (c=='\n') {
  printf ("\nString %d, length=%d",++cnt,len); len=0;
 }
 else len++;
 if (feof(fp)) break;
}
if (len) printf ("\nString %d, length=%d",++cnt,len);
```

Важно! Из-за особенностей реализации `fgetc`, без последней проверки за телом цикла код мог "не обратить внимания", например, на последнюю строку файла, состоящую только из пробелов и не завершающуюся переводом строки.

Пример. Читаем построчно файл с известной максимальной длиной строки. Предполагается, что файл уже открыт для чтения.

```
char buf[128];
while (1) {
 fgets(buf,127,fp);
 if (feof(fp)) break;
 int len = strlen(buf);
 if (buf[len-1]=='\n') buf[len-1]='\0';
 puts (buf); //Вывести прочитанные строки на экран
}
```

Важно! Без дополнительной обработки прочитанные из файла строки при выводе будут содержать лишние пустые строки между строками данных. Это происходит потому, что функция `fgets` читает строку файла вместе с символом перевода строки (точней, под Windows - с парой символов `\r\n`, интерпретируемых как один), а `puts` добавляет к выводимой строке ещё один перевод строки.

Если максимальная длина строки принципиально не ограничена, помочь может либо предварительное посимвольное чтение файла для её определения, либо работа с файлом как с бинарными данными. *Бинарный файл* отличается от текстового тем, что необязательно состоит из печатаемых символов со стандартными разделителями между ними. Соответственно, для него не имеет смысла понятие "строки данных", а основной способ работы с ним – чтение и запись наборов байт указанного размера. Основные функции для чтения и записи бинарных данных – `fread` и `fwrite` соответственно. В базовой реализации они имеют по 4 параметра:

- `void *buffer` - нетипизированный указатель на место хранения данных;
- `size_t (unsigned) size` - размер элемента данных в байтах.
- `size_t count` - максимальное количество элементов, которые требуется прочитать (записать);
- `FILE *stream` - указатель на структуру `FILE`

Пример. Целочисленный массив `a` запишем в двоичный файл.

```
FILE *fp=fopen ("data.dat","wb");
if (fp==NULL) {
 puts ("не удалось открыть файл");
 getchar(); exit (1);
}
const int n=10;
int a[n];
for(int i=0; i<n; i++) a[i]=i+1;
for (int i=0; i<10; i++) fwrite (&a[i],sizeof(int),1,fp);
 //Записали 10 эл-тов по одному
 //Если sizeof(int)=2, получим файл из 20 байт, если 4 - из 40
fclose (fp);
```

Учитывая, что данные массива хранятся в последовательно идущих адресах памяти, цикл `for` для записи мы могли заменить одним оператором:

```
fwrite (&a[0],sizeof(int),n,fp);
```

Подход к чтению данных с помощью `fread` аналогичен. Например, если файл уже открыт для чтения в режиме "rb":

```
unsigned char c;
//…
fread (&c,1,1,fp); //читаем по 1 байту
```

```
unsigned char buf[512];
//…
fread (&buf,1,512,fp);
 //читаем по 1 сектору - по 512 байт
```

Для файлов, открытых в режиме "r+b", разрешены и чтение, и запись (произвольный доступ). Поэтому при работе с такими файлами нужны функции позиционирования файлового указателя:

- функции `fgetpos` и `ftell` позволяют выполнить чтение текущей позиции указателя в файле;
- функции `fseek` и `fsetpos` позволяют осуществить переход к нужной позиции в файле.

Пример. Определить размер файла в байтах, предположим, что файл уже открыт в режиме чтения или произвольного доступа.

```
fseek (fp, 0, SEEK_END); //Встали на 0 байт от конца файла
long int pos;
pos = ftell (fp); //Получили текущую позицию в файле
if (pos<0) puts ("\nОшибка");
else if (!pos) puts ("\nФайл пуст");
else printf ("\nВ файле %ld байт",pos);
```

Материал для чтения из пособия: пп. 8.6-8.11. Обратите внимание на таблицы с описанными прототипами функций ввода/вывода.

Рекомендуемые задачи: базовое задание включает две задачи, первая из которых предполагает обработку файла как текстовых данных, вторая – как бинарных. В

качестве дополнительной третьей задачи может быть предложена реализация одной из задач 1, 2, содержащая консольный интерфейс и меню.

> Про conio.h и почему его не надо использовать:
>
> Для ввода/вывода через цветную консоль во многих источниках используются методы библиотеки `conio.h`. Следует учитывать, что её реализации в компиляторах от Borland и Microsoft значительно отличаются, а в компиляторах под Unix/Linux реализации `conio.h` могут отсутствовать.
>
> Как вариант, в компиляторах Visual Studio можно использовать аналоги `conio.h` от сторонних разработчиков, например, открытый проект coniow.h. Законченный пример кода, реализующего несложное консольное меню для Visual Studio, есть вот здесь. Предполагается, что к проекту подключены заголовочный файл `coniow.h` и файл исходного кода `coniow.c`.

Оглавление серии

теги: форматы c++ учебное

05.11.2015, 09:20; рейтинг: 34572