

## ФАЙЛОВЫЕ ОПЕРАЦИИ В/В В C++

*Файл* – это именованная область внешней памяти. Файл имеет следующие характерные особенности:

- 1) имеет имя на диске, что дает возможность программам работать с несколькими файлами;
- 2) длина файла ограничивается только емкостью диска.

## ФАЙЛОВЫЙ ВВОД-ВЫВОД

Для того, чтобы в C++ работать с файлами, необходимо подключить заголовочный файл *fstream*:

```
#include <fstream>
```

После этого можно объявлять объекты, привязанные к файлам: для чтения данных из файла используются объекты типа *ifstream* (аббревиатура от **input file stream**), для записи данных в файл используются объекты типа *ofstream* (**output file stream**).

Например:

```
ifstream in; // Поток in будет использоваться для чтения
ofstream out; // Поток out будет использоваться для записи
```

Чтобы привязать тот или иной поток к файлу (открыть файл для чтения или для записи) используется метод *open*, которому необходимо передать параметр – текстовую строку, содержащую имя открываемого файла.

```
in.open("input.txt");
out.open("output.txt");
```

После открытия файлов и привязки их к файловым потокам, работать с файлами можно так же, как со стандартными потоками ввода-вывода *cin* и *cout*.

Если указывается имя файла при объявлении объекта типа *ofstream*, C++ создаст новый файл на диске, используя указанное имя, или перезапишет файл с таким же именем, если он уже существует на диске.

Следующая программа OUT\_FILE.CPP создает объект типа *ofstream* и затем использует оператор вставки для вывода нескольких строк текста в файл BOOKINFO.DAT:

```
#include <iostream>
#include <fstream> //для работы с файлами
using namespace std;
void main()
{
    ofstream book_file;
    book_file.open("BOOKINFO.DAT");
    book_file << "Учимся программировать на языке C++, " << "Вторая
    редакция"<< endl;
    book_file << "Jamsa Press" << endl;
```

```

        book_file << "22.95" << endl;
    }

```

В данном случае программа открывает файл BOOKINFO.DAT, если он уже существует на диске, или создает новый файл на диске, используя указанное имя, и затем записывает три строки в файл, используя оператор вставки.

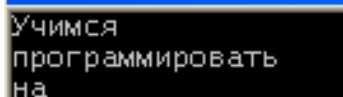
Следующая программа FILE\_IN.CPP открывает файл BOOKINFO.DAT, который создали с помощью предыдущей программы, и читает, а затем отображает первые три элемента файла:

```

#include <iostream>
#include <fstream> //для работы с файлами
#include <windows.h>
using namespace std;
void main(void)
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    ifstream input_file;
    input_file.open("BOOKINFO.DAT");
    char one[64], two[64], three[64];
    input_file >> one;
    input_file >> two;
    input_file >> three;
    cout << one << endl;
    cout << two << endl;
    cout << three << endl;
}

```

Подобно *cin*, входные файловые потоки используют пустые символы, чтобы определить, где заканчивается одно значение и начинается другое. В результате, при запуске предыдущей программы на дисплее появится следующий вывод:



```

Учимся
программировать
на

```

## ЧТЕНИЕ ЦЕЛОЙ СТРОКИ ФАЙЛОВОГО ВВОДА

Объекты типа *ifstream* могут использовать *getline* для чтения строки файлового ввода.

Следующая программа FILELINE.CPP использует функцию *getline* для чтения всех трех строк файла BOOKINFO.DAT:

```

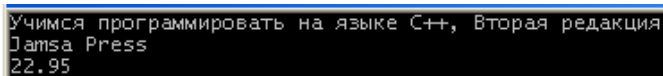
#include <iostream>
#include <fstream> //для работы с файлами
#include <windows.h>
using namespace std;

```

```

void main(void)
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    ifstream input_file;
    input_file.open("BOOKINFO.DAT");
    char one[64], two[64], three[64];
    input_file.getline(one, sizeof(one)) ;
    input_file.getline(two, sizeof(two));
    input_file.getline(three, sizeof(three)) ;
    cout << one << endl;
    cout << two << endl;
    cout << three << endl;
}

```



```

Учимся программировать на языке C++, Вторая редакция
Damsa Press
22.95

```

В данном случае программа успешно читает содержимое файла, потому что она знает, что файл содержит три строки. Однако, программа не всегда знает, сколько строк содержится в файле. В таких случаях программы будут продолжать чтение содержимого файла, пока не встретят конец файла.

## ОПРЕДЕЛЕНИЕ КОНЦА ФАЙЛА

Чтобы определить конец файла, программы могут использовать функцию *eof* потокового объекта. Эта функция возвращает значение 0, если конец файла еще не встретился, и 1, если встретился конец файла. Используя цикл *while*, программы могут непрерывно читать содержимое файла, пока не найдут конец файла, как показано ниже:

```

while (!input_file.eof())
{
    // Операторы
}

```

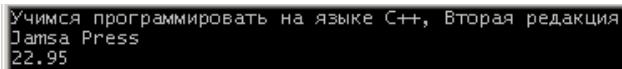
В данном случае программа будет продолжать выполнять цикл, пока функция *eof* возвращает ложь (0). Следующая программа TEST\_EOF.CPP использует функцию *eof* для чтения содержимого файла BOOKINFO.DAT, пока не достигнет конца файла:

```

#include <iostream>
#include <fstream> //для работы с файлами
#include <windows.h>
#include <string>
using namespace std;
void main(void)
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    ifstream input_file;
    input_file.open("BOOKINFO.DAT");
    char line[64];
    while (!input_file.eof())
    {
        input_file.getline(line, sizeof(line));
        cout << line << endl;
    }
}

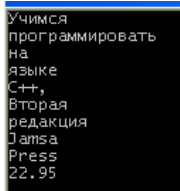
```

```
}  
}
```



Аналогично, следующая программа WORD\_EOF.CPP читает содержимое файла по одному слову за один раз, пока не встретится конец файла:

```
#include <iostream>  
#include <fstream> //для работы с файлами  
#include <windows.h>  
#include <string>  
using namespace std;  
void main(void)  
{  
    SetConsoleOutputCP(1251);  
    SetConsoleCP(1251);  
    ifstream input_file;  
    input_file.open("BOOKINFO.DAT");  
    char word[64];  
    while (!input_file.eof())  
    {  
        input_file >> word;  
        cout << word << endl;  
    }  
}
```



И наконец, следующая программа CHAR\_EOF.CPP читает содержимое файла по одному символу за один раз, используя функцию *get*, пока не встретит конец файла:

```
#include <iostream>  
#include <fstream> //для работы с файлами  
#include <windows.h>  
using namespace std;  
void main(void)  
{  
    SetConsoleOutputCP(1251);  
    SetConsoleCP(1251);  
    ifstream input_file;  
    input_file.open("BOOKINFO.DAT");  
    char letter;  
    while (!input_file.eof())  
    {  
        letter = input_file.get();  
        cout << letter;  
    }  
}
```

## ПРОВЕРКА ОШИБОК ПРИ ВЫПОЛНЕНИИ ФАЙЛОВЫХ ОПЕРАЦИЙ

Чтобы помочь программам следить за ошибками, можно использовать функцию *fail* файлового объекта. Если в процессе файловой операции ошибок не было, функция *fail* файлового объекта вернет ложь (0). Однако, если встретилась ошибка, функция *fail* вернет истину. Например, если программа открывает файл, ей следует использовать функцию *fail*, чтобы определить, произошла ли ошибка, как это показано ниже:

```
ifstream input_file;  
input_file.open("BOOKINFO1.DAT");  
if (input_file.fail())  
{  
    cerr << "Ошибка открытия BOOKINFO1.DAT" << endl;  
    getch();  
    exit(1);  
}
```

Таким образом, программы должны убедиться, что операции чтения и записи прошли успешно. Следующая программа TEST\_ALL.CPP использует функцию *fail* для проверки различных ошибочных ситуаций:

```
#include <iostream>  
#include <fstream> //для работы с файлами  
#include <windows.h>  
using namespace std;  
void main(void)  
{  
    SetConsoleOutputCP(1251);  
    SetConsoleCP(1251);  
    char line[256] ;  
    ifstream input_file;  
    input_file.open("BOOKINFO.DAT");  
    if (input_file.fail()) cerr << "Ошибка открытия BOOKINFO.DAT" << endl;  
    else  
    {  
        while ((! input_file.eof()) && (! input_file.fail()))  
        {  
            input_file.getline(line, sizeof(line)) ;  
            if (! input_file.fail()) cout << line << endl;  
        }  
    }  
}
```

## ЗАКРЫТИЕ ФАЙЛА

При завершении программы операционная система закроет открытые ею файлы. Однако, как правило, если программе файл больше не нужен, она должна его закрыть. Для закрытия файла программа должна использовать функцию *close*, как показано ниже:

```
input_file.close ();
```

Когда файл закрывается, все данные, которые программа писала в этот файл, сбрасываются на диск, и обновляется запись каталога для этого файла.

## УПРАВЛЕНИЕ ОТКРЫТИЕМ ФАЙЛА

В примерах представленных программ файловые операции ввода и вывода выполнялись с начала файла. Бывает необходимо добавлять информацию в конец существующего файла. Для открытия файла в режиме добавления нужно при его открытии указать второй параметр, как показано ниже:

```
ofstream book_file;  
book_file.open("BOOKINFO.DAT", ios::app);
```

В данном случае параметр `ios::app` указывает режим открытия файла. Значения режимов открытия файла перечислены в табл. 34.

Таблица 34. Значения режимов открытия.

Режим открытия	Назначение
<code>ios::app</code>	Открывает файл в режиме добавления, располагая файловый указатель в конце файла.
<code>ios::ate</code>	Располагает файловый указатель в конце файла.
<code>ios::in</code>	Указывает открыть файл для ввода.
<code>ios::binary</code>	Открывает файл в двоичном режиме.
<code>ios::out</code>	Указывает открыть файл для вывода.
<code>ios::trunc</code>	Сбрасывает (перезаписывает) содержимое существующего файла.

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции `open()`:

- `ofstream fout("cppstudio.txt", ios_base::app); /* открываем файл для добавления информации к концу файла*/`
- `fout.open("cppstudio.txt", ios_base::app); /* открываем файл для добавления информации к концу файла*/`

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции `или` `|`, например: `ios_base::out | ios_base::trunc` - открытие файла для записи, предварительно очистив его.

Объекты класса `ofstream`, при связке с файлами по умолчанию содержат режимы открытия файлов `ios_base::out | ios_base::trunc`. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи.

Объекты класса `ifstream` связываясь с файлом, имеют по умолчанию режим открытия файла `ios_base::in` - файл открыт только для чтения. Режим открытия файла ещё называют — флаг, для удобочитаемости в дальнейшем будем использовать именно этот термин. В таблице 34 перечислены не все флаги.

Флаги `ate` и `app` по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг `app` позволяет производить запись, только в конец файла, а флаг `ate` просто переставляет указатель в конец файла и не ограничивает места записи.

Разработаем программу, которая, используя операцию `sizeof()`, будет вычислять характеристики основных типов данных в C++ и записывать их в файл. Характеристики:

1. число байт, отводимое под тип данных
2. максимальное значение, которое может хранить определённый тип данных.

Запись в файл должна выполняться в таком формате:

```
/* data type    byte    max value
bool           = 1      255.00
char           = 1      255.00
short int      = 2      32767.00
unsigned short int = 2    65535.00
int            = 4      2147483647.00
unsigned int    = 4      4294967295.00
long int       = 4      2147483647.00
unsigned long int = 4    4294967295.00
float          = 4      2147483647.00
long float     = 8      9223372036854775800.00
double         = 8      9223372036854775800.00 */
```

В конце программы явно закрыли файл, хотя это и не обязательно, но считается хорошим тоном программирования. Стоит отметить, что все функции и манипуляторы используемые для форматирования стандартного ввода/вывода актуальны и для файлового ввода/вывода. Поэтому не возникло никаких ошибок, когда оператор ***cout*** был заменён объектом ***fout***.

## ВЫПОЛНЕНИЕ ОПЕРАЦИЙ ЧТЕНИЯ И ЗАПИСИ

Все представленные программы выполняли файловые операции над символьными строками. По мере усложнения программ понадобится читать и писать массивы и структуры. Для этого программы могут использовать функции ***read*** и ***write***. При использовании функций ***read*** и ***write*** необходимо указать буфер данных, в который данные будут читаться или из которого они будут записываться, а также длину буфера в байтах, как показано ниже:

```
input_file.read(buffer, sizeof(buffer)) ;
output_file.write(buffer, sizeof(buffer));
```

Например, следующая программа **STRU\_OUT.CPP** использует функцию ***write*** для вывода содержимого структуры в файл **EMPLOYEE.DAT**:

```
#include <iostream>
#include <fstream> //для работы с файлами
using namespace std;
void main(void)
{
    struct employee
    {
        char name[64];
        int age;
        float salary;
    } worker = { "Иванов Петр", 33, 25000.0 };
    ofstream emp_file("e:\\EMPLOYEE.DAT") ;
    emp_file.write((char *) &worker, sizeof(employee));
}
```

Функция ***write*** обычно получает указатель на символьную строку. Символы (***char*** \*) представляют собой оператор приведения типов, который информирует компилятор,

что передается указатель на другой тип. Подобным образом следующая программа **STRU\_IN.CPP** использует метод **read** для чтения из файла информации о служащем:

```
#include <iostream>
#include <fstream> // работа с файлами
using namespace std;
void main(void)
{
    setlocale(LC_ALL, "rus");
    struct employee
    {
        char name [64];
        int age;
        float salary;
    } worker;
    ifstream emp_file("e:\\EMPLOYEE.DAT");
    emp_file.read((char *) &worker, sizeof(employee));
    cout << worker.name << endl;
    cout << worker.age << endl;
    cout << worker.salary << endl;
}
```



```
Иванов Петр
33
25000
Для продолжения нажмите любую клавишу
```

## ЧТО НЕОБХОДИМО ЗНАТЬ

Заголовочный файл **fstream.h** определяет классы **ifstream** и **ofstream**, с помощью которых ваша программа может выполнять операции файлового ввода и вывода.

Для открытия файла на ввод или вывод вы должны объявить объект типа **ifstream** или **ofstream**, передавая конструктору этого объекта имя требуемого файла.

После того как ваша программа открыла файл для ввода или вывода, она может читать или писать данные, используя операторы извлечения (>>) и вставки (<<).

Ваши программы могут выполнять ввод или вывод символов в файл или из файла, используя функции **get** и **put**.

Ваши программы могут читать из файла целую строку, используя функцию **getline**.

Большинство программ читают содержимое файла, пока не встретится конец файла. Ваши программы могут определить конец файла с помощью функции **eof**.

Когда ваши программы выполняют файловые операции, они должны проверять состояние всех операций, чтобы убедиться, что операции выполнены успешно. Для проверки ошибок ваши программы могут использовать функцию **fail**.

Если вашим программам необходимо вводить или выводить такие данные, как структуры или массивы, они могут использовать методы **read** и **write**.

Если ваша программа завершила работу с файлом, его следует закрыть с помощью функции **close**.