

Работа с потоками

Для работы с данными программы необходимо грамотно организовать ввод/вывод данных. В языке C++ существует множество типов данных, в том числе, создаваемых пользователем самостоятельно, поэтому необходимо реализовать ввод/вывод данных для всех типов.

Система ввода/вывода создает между источником и приемником данных единый интерфейс, не зависящий от физических устройств. Это абстрактное средство связи называется *потоком*.

Основными операциями ввода и вывода являются перегруженные операторы сдвига влево << («прочитать из») и вправо >> («записать в»). Автором языка C++ Б. Страуструпом были выбраны эти операции из-за того, что, во-первых, они достаточно редко встречаются, во-вторых, имеют достаточно низкий приоритет, что позволяет не использовать скобки при выводе. Круглые скобки необходимо при выводе выражений, содержащих логические выражения и, естественно, операции сдвига влево и вправо.

Все данные, необходимые для работы со потоками, находятся в библиотеке <iostream>.

Как и в случае работы со строками, для потоков определен свой тип данных, используемый для работы с числами (количество позиций, текущая позиция и т. д.): `streamsize`

Потоки делятся на *стандартные* (для работы с потоками ввода с клавиатуры и потоками вывода на экран), *файловые* (для работы с файлами) и *строковые* (для работы со строками).

По своему действию потоки делятся на *входные*, *выходные* и *двунаправленные*.

1.1. Потоки ввода

При подключении библиотеки <iostream> генерируется два стандартных потока ввода: `cin` и `wcin` (для ввода символов типа `wchar_t`).

Операция >> пропускает пробельные символы, поэтому можно считывать данные в цикле.

Если идет считывание в строку типа `C`, то возможно переполнение, поэтому, если необходимо использовать именно `char*`, то можно ограничить число читаемых операций с помощью операции `width`:

```
char str[10];  
cin.width(10);
```

Будет прочитано максимум 9 символов и в конце строки добавлен ноль.

Каждый поток характеризуется своим состоянием. Существует четыре флага для проверки состояния потока ввода:

1. `good()` — следующая операция считывания может быть выполнена.
2. `eof()` — конец файла. Флаг окончания считывания данных.
3. `fail()` — следующая операция не может быть выполнена (потеряны символы, например). Считается, что поток не испорчен.
4. `bad()` — поток испорчен. Считывание прерывается, например, при считывании целых чисел встретилась десятичная точка.

Поскольку состояние потока возвращает `true` только тогда, когда состояние потока `good()`, можно использовать поток в качестве условия:

```
while(cin >> f)
```

Операция `>>` предназначена для форматированного ввода, то есть для чтения данных ожидаемого типа и формата. Для считывания символов существуют функции неформатированного ввода:

`cin.get()` считывание одного символа. Обычно используется для очищения потока (когда в нем содержатся оставшиеся с предыдущего считывания символы, например, переход на следующую строку).

`cin.get(c)` считывание одного символа в переменную с типа `char`.

`cin.get(c, n)` считывание либо `n` символов либо до конца строки в строку типа `char*`.

`cin.get(c, n, term)` считывание либо `n` символов либо до символа `term` типа `char` в строку типа `char*`.

`cin.getline(c, n)` считывание либо `n` символов либо до конца строки в строку типа `char*`.

`cin.getline(c, n, term)` считывание либо `n` символов либо до символа `term` типа `char` в строку типа `char*`.

`cin.ignore(n)` либо `n` символов либо до конца файла. Эти символы нигде не хранятся, то есть, служит для пропуска ровно `n` символов.

`cin.ignore(n, term)` считывание либо `n` символов либо до символа `term` типа `char`. Эти символы нигде не хранятся.

`cin.read(str, n)` считывание ровно `n` символов в массив `str` типа `char` без перевода в строку `C` (ноль в конец строки не добавляется).

`getline(cin, str)` считывание строки типа `string`.

Данные функции не пропускают пробельные символы, поэтому используются при работы со строками, когда пробел является таким же символом, как и все остальные.

Если функции не считывают ни одного символа, состояние потока становится `fail()`, и последующие попытки закончатся неудачей.

Поскольку функции `get()` не удаляют терминальные символы (символ окончания считывания), их нельзя использовать несколько раз:

```
while(cin){
    cin.get(str, 200);
    cout << str;
}
```

В таком случае лучше использовать `getline()`. Работает также как и `get()`, но удаляет терминальные символы из потока.

Пример 1.1. Примером работы потоков ввода:

Листинг 1.1.

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
```

```

6  int main (){
7
8      setlocale (LC_ALL, "russian");
9      char b[100], c;
10     cout << "ввод символа:\n";
11     cin.get(c); // ввод символа
12     cin.get(); // очистка буфера
13     cout << "с=" << c << endl;
14     cout << "ввод строки:\n";
15     cin.get(b, 100);
16     cout << "строка - " << b << endl;
17     cin.get();
18     cout << "ввод строки до #:\n";
19     cin.get(b, 100, '#');
20     cout << "строка - " << b << endl;
21     cout << "ввод строки:\n";
22     cin.getline(b, 100);
23     cout << "строка - " << b << endl;
24     cout << "ввод строки до #:\n";
25     cin.getline(b, 100, '#');
26     cout << "строка - " << b << endl;
27     string str;
28     cout << "ввод строки C++:\n";
29     getline (cin, str);
30     cout << "строка - " << str << endl;
31     cout << "ввод массива:\n";
32     int a[10], n = 10;
33     for(int i = 0; i < n; i++) //ввод массива
34         cin >> a[i];
35     for(int i = 0; i < n; i++) //вывод массива
36         cout << a[i] << " ";
37     cout << endl;
38     return 0;
39
40 }

```

Результат работы программы:

```
C:\WINDOWS\system32\cmd.exe
Ввод символа:
R
c=R
ввод строки:
Hello world
строка - Hello world
ввод строки до #:
Hello wo#rld
строка - Hello wo
ввод строки:
строка - #rld
ввод строки до #:
Hello wo#rld
строка - Hello wo
ввод строки C++:
строка - rld
ввод массива:
1 2 3 4 5.5 6 7 8 9 0
1 2 3 4 5 -858993460 -858993460 -858993460 -858993460 -858993460
Для продолжения нажмите любую клавишу . . .
```

Как видно из результатов работы программы, неиспользуемые данные остаются в буфере и считываются в следующую переменную (строки 8 и 11 рисунка, функция `getline` (строка 22 Листинга 1.5) не выполняется, терминальные символ `#` не отбрасывается и строки 13 и 16 рисунка, функция `getline` (строка 29 Листинга 1.5) не выполняется, терминальные символ `#` отбрасывается).

Ввод массива демонстрирует корректность ввода данных. Данные считываются до десятичной точки, после этого считывание невозможно, поэтому последние пять элементов массива заполнены мусором. □

1.2. Потоки вывода

При подключении библиотеки `<iostream>` создаются стандартные потоки вывода: `cout` (поток вывода на экран), `cerr` и `clog` (небуферизованный и буферизованный, соответственно, потоки вывода ошибок) для вывода символов типа `char` и `wcout` `wcerr` и `wclog` для вывода символов типа `wchar_t`.

Для вывода данных используются операция вывода (`<<`), а также функции `cout`, `put(c)` — для вывода символа и `cout.write(str, n)` — для вывода `n` символов строки `str`.

1.2.1. Форматирование

Достаточно часто необходимо применять форматирование для вывода на экран данных. Форматирование можно производить двумя способами: либо с помощью флагов, либо с помощью манипуляторов.

В таблице ниже приведены флаги, которые можно устанавливать для вывода данных.

left	выравнивание данных слева
right	выравнивание данных справа
internal	заполнитель между знаком и значением
dec	вывод в десятичной системе счисления
hex	вывод в шестнадцатеричной системе счисления
oct	вывод в восьмеричной системе счисления
scientific	вывод числа с плавающей запятой в виде: <i>d.ddddddEdd</i>
fixed	вывод числа с плавающей запятой в виде: <i>dddddd.dddd</i>
showbase	печать префикса: 0 — для восьмеричной, 0X — для шестнадцатеричной
showpoint	печать незначащих нулей
showpos	печать + перед положительным числом
boolalpha	использовать символьное представление для true и false
uppercase	выводить 'E', 'X'
adjustfield	флаги для выравнивания полей
basefield	флаги системы счисления
floatfield	флаги вывода чисел с плавающей запятой
unitbuf	очистка буфера после каждой операции вывода

Для работы с флагами используется функция `setf(f)` и `unsetf(f)`. Все флаги описаны в классе `ios_base`, поэтому нужно использовать операцию доступа `::`. Если необходимо изменить несколько значений флагов, используется «побитовое ИЛИ» (`|`).

Например,

```
cout.setf(ios_base::left|ios_base::fixed|ios_base::showpoint).
```

В случае выравнивания полей, вывода системы счисления или чисел с плавающих точкой необходимо использовать функцию `setf` с двумя параметрами: изменения флага и указатель на то, что именно хотим изменить:

```
cout.setf(ios_base::fixed, ios_base::floatfield).  
cout.setf(ios_base::oct, ios_base::basefield).
```

```
cout.setf(ios_base::left, ios_base::adjustfield).
```

Существует три функции для форматирования:

<code>cout.width(n)</code>	для вывода данных используется n позиций
<code>cout.fill(c)</code>	заполнение пустых позиций символами c
<code>cout.precision(n)</code>	вывод n цифр после запятой

Последние три функции действуют только для одной операции вывода, а флаги действительны до тех пор, пока их не сбросить с помощью функции `unsetf`.

Пример 1.2. Примером работы потоков вывода:

Листинг 1.2.

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main (){
7
8     setlocale (LC_ALL, "russian");
9     int a = 1245;
10    cout << "Системы счисления:\n";
11    cout.setf (ios_base::showbase|ios_base::uppercase); //печать базы и прописной буквы
12    cout.setf (ios_base::hex, ios_base::basefield); //16
13    cout << a << " ";
14    cout.setf (ios_base::showbase|ios_base::oct, ios_base::basefield); //8
15    cout << a << " ";
16    cout.setf (ios_base::showbase|ios_base::dec, ios_base::basefield); //10
17    cout << a << "\n";
18    a = 12;
19    cout.setf (ios_base::showpos); //вывод +
20    cout << "Температура воздуха " << a << " градусов\n";
21    cout << "Вывод вещественных чисел:\n";
22    double b = 0.00000015;
23    cout.setf (ios_base::fixed, ios_base::floatfield ); //0.01
24    cout.precision (7);
25    cout << b << endl;
26    cout.unsetf (ios_base::showpos); //очистка +
27    cout.setf (ios_base::scientific, ios_base::floatfield ); //0Е-01
```

```

28  cout << b << endl;
29  a = -123;
30  cout << "Заполнение и выравнивание:\n";
31  cout.width(8);
32  cout.fill('*');
33  cout.setf(ios_base::internal, ios_base::adjustfield);
34  cout << a << endl;
35  cout.width(8);
36  cout.fill('*');
37  cout.setf(ios_base::left, ios_base::adjustfield);
38  cout << a << endl;
39  cout.width(8);
40  cout.fill('*');
41  cout.setf(ios_base::right, ios_base::adjustfield);
42  cout << a << endl;
43  cout << "Вывод true словами:\n";
44  cout.setf(ios_base::boolalpha);
45  cout << (a < 12) << " " << (a > 100) << endl;
46  return 0;
47
48 }

```

Результат работы программы:

```

C:\WINDOWS\system32\cmd.exe
Системы счисления:
0X4DD 02335 1245
Температура воздуха +12 градусов
Вывод вещественных чисел:
+0.0000001
1.5000000E-007
Заполнение и выравнивание:
-****123
-123****
****-123
Вывод true словами:
true false
Для продолжения нажмите любую клавишу . . .

```

□

1.2.2. Манипуляторы

Использование флагов достаточно запутано и может привести к ошибкам. Поэтому достаточно часто используются манипуляторы. Для их использование необходимо

подключить библиотеку `<iomanip>`.

В таблице ниже приведены некоторые манипуляторы:

<code>left</code>	выравнивание данных слева
<code>right</code>	выравнивание данных справа
<code>internal</code>	заполнитель между знаком и значением
<code>dec</code>	вывод в десятичной системе счисления
<code>hex</code>	вывод в шестнадцатеричной системе счисления
<code>oct</code>	вывод в восьмеричной системе счисления
<code>scientific</code>	вывод числа с плавающей запятой в виде: <i>d.ddddddEdd</i>
<code>fixed</code>	вывод числа с плавающей запятой в виде: <i>dddddd.dddd</i>
<code>showbase</code> <code>noshowbase</code>	печать префикса: 0 — для восьмеричной, 0X — для шестнадцатеричной отмена печати префикса
<code>showpoint</code> <code>noshownpoint</code>	печать незначащих нулей отмена печати незначащих нулей
<code>showpos</code> <code>noshownpos</code>	печать + перед положительным числом отмена печати + перед положительным числом
<code>boolalpha</code> <code>noboolalpha</code>	использовать символьное представление для <code>true</code> и <code>false</code> вывод 0 и 1 для логических данных
<code>uppercase</code> <code>nouppercase</code>	выводить 'E', 'X' выводить 'e', 'x'
<code>setbase(b)</code>	вывод по основанию <code>b</code>
<code>setfill(c)</code>	заполнение данных символом <code>c</code>
<code>setprecision(n)</code>	вывод <code>n</code> цифр после запятой
<code>setw(n)</code>	для вывода данных используется <code>n</code> позиций

Большая часть манипуляторов работает пока не отключить их с помощью соответствующих функций, а вот функции `setw(n)` и `setprecision(n)` работают только для одной операции вывода.

Пример 1.3. Пример работы потоков вывода:

Листинг 1.3.

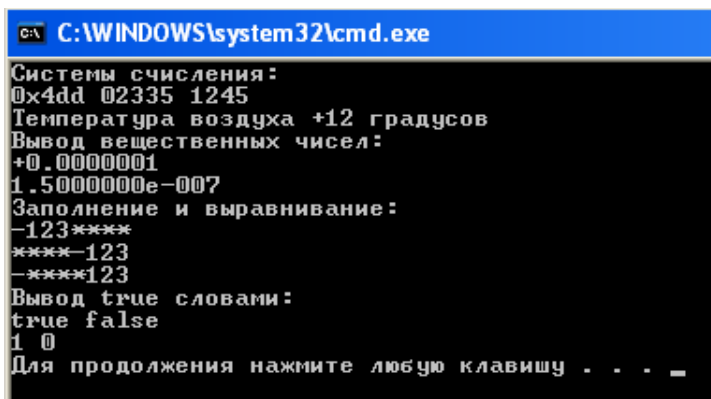
```
1 #include <iostream>
2 #include <string>
3 #include <iomanip>
4
```

```

5 using namespace std;
6
7 int main (){
8
9     setlocale (LC_ALL, "russian");
10    int a = 1245;
11    cout << "Системы счисления:\n";
12    cout << showbase << hex << a << " " << oct << a << " " << dec << a << endl;
13    a = 12;
14    cout << "Температура воздуха " << showpos << a << " градусов\n";
15    cout << "Вывод вещественных чисел:\n";
16    double b = 0.00000015;
17    cout << fixed << setprecision(7) << b << endl;
18    cout << noshowpos << scientific << b << endl;
19    a = -123;
20    cout << "Заполнение и выравнивание:\n";
21    cout << setw(8) << setfill('*') << left << a << "\n";
22    cout << setw(8) << right << a << "\n";
23    cout << setw(8) << internal << a << "\n";
24    cout << "Вывод true словами:\n";
25    cout << boolalpha << (a < 12) << " " << (a > 100) << endl;
26    cout << noboolalpha << (a < 12) << " " << (a > 100) << endl;
27    return 0;
28
29 }

```

Результат работы программы:



```

C:\WINDOWS\system32\cmd.exe
Системы счисления:
0x4dd 02335 1245
Температура воздуха +12 градусов
Вывод вещественных чисел:
+0.0000001
1.5000000e-007
Заполнение и выравнивание:
-123****
****-123
-****123
Вывод true словами:
true false
1 0
Для продолжения нажмите любую клавишу . . .

```

Видно, что программа выглядит короче и понятнее.

□

1.3. Файловые потоки

Под файлом в языке программирования понимается любое устройство откуда можно считывать данные либо куда можно записывать данные.

Работа с файлами может быть описана следующим алгоритмом:

1. Создание потока.
2. Связывание потока с файлом.
3. Считывание из файла / запись в файл.
4. Закрытие файла.

Как и в случае со стандартными потоками, файловые потоки бывают входными (чтение из файла), выходными (запись в файл) и двунаправленными.

Для работы с файловым потоком необходимо подключить библиотеку `<fstream>`.

Поскольку файловые потоки могут использоваться в разных функциях, чаще всего файловый поток создается в качестве глобальной переменной.

В библиотеке `<fstream>` определены классы `ifstream` (входной поток), `ofstream` (выходной поток) и `fstream` (двунаправленный поток).

Создание потока означает создать объект соответствующего класса:

```
ifstream in; //файловый поток ввода
ofstream out; //файловый поток вывода
fstream io; //файловый поток ввода-вывода
```

Имя потока может быть любое, желательно понятное. Можно создать несколько потоков ввода или вывода:

```
ifstream in0, in1, in2; //файловый поток ввода
```

Связывание потока с файлом с помощью операции `open()`.

Файл для чтения должен существовать до начала работы с файлом. Файл для записи создается автоматически. Если открывается для записи уже существующий файл, то все имеющиеся данные уничтожаются.

Файл связывается с потоком следующим образом:

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

Для того, чтобы не писать полный путь к файлу, этот файл должен быть расположен в той же директории, где расположен файл с расширением `.cpp`. В противном случае необходимо писать полный путь к файлу.

Можно также объединить создание потока и связывание его с файлом:

```
ifstream in("input.txt"); //создать поток и связать с файлом input.txt
```

Функция `open` может иметь два параметра: первый — это имя файла, второй — способ открытия файла.

В таблице ниже приведены возможные параметры открытия файла:

app	добавление данных в конец файла
ate	открыть и перейти к концу файла
binary	работа в бинарном режиме
in	открыть для чтения
out	открыть для записи
trunc	урезать файл до нулевой длины

Также как с флагами для работы с потоками, эти параметры определены в классе `ios_base` и применять их необходимо с помощью оператора доступа `::`. Если необходимо использовать несколько параметров, используется «побитовое ИЛИ».

Например, открыть файл для добавления в конец файла:

```
ofstream out("input.txt", ios_base::app);
```

Открытие файла для чтения и записи:

```
fstream file("input.txt", ios_base::in | ios_base::out);
```

Поскольку файл, открываемый для чтения, должен существовать, необходимо проверить правильность открытия файла. Для этого можно использовать функцию `is_open()`, которая возвращает `true`, если файл открыт корректно:

```
ifstream in("input.txt");
if(!in.is_open())
    cout << "Ошибка открытия файла"
```

Считывание из файла / запись в файл происходит с помощью тех же операторов и функций, что и работа со стандартными потоками:

- для чтения — оператор ввода (`>>`), функции `get()`, `getline()`, `read()`.

Только вместо имени стандартного потока `cin` используется имя созданного пользователем потока:

```
in >> x;
in.get(x);
in.getline(x,10,'#'); //для строк типа char
getline(in,str); //для строк типа string
```

- для записи — оператор вывода (`<<`), функции `put()`, `write()`. Только вместо имени стандартного потока `cout` используется имя созданного пользователем потока:

```
out << x;
out.put(x);
```

Заккрытие файла происходит с помощью функции `close()`:

```
out.close();
```

Чтобы определить, достигнут ли конец файла, можно воспользоваться функцией `peek()`. Эта функция возвращает значение текущего элемента, но не считывает его. Признаком конца файла является EOF:

```
while(in.peek() != EOF)
```

или воспользоваться функцией `eof()`:

```
while(in.eof())
```

Пример 1.4. Даны два файла. Поменять местами их содержимое, используя дополнительный файл.

Листинг 1.4.

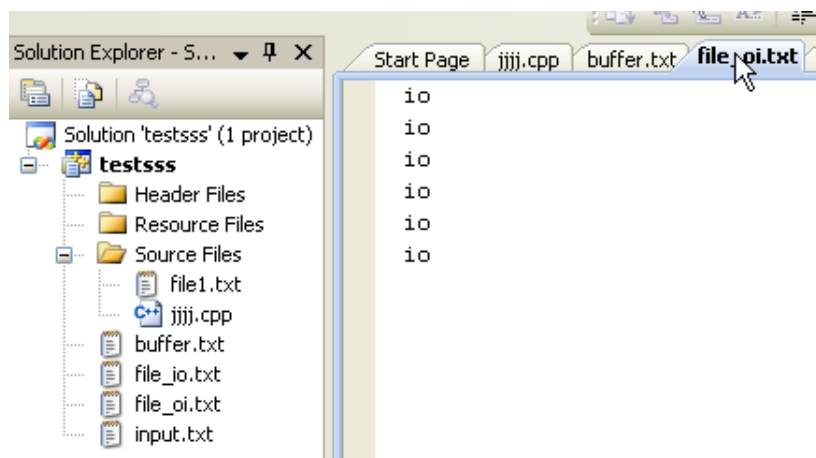
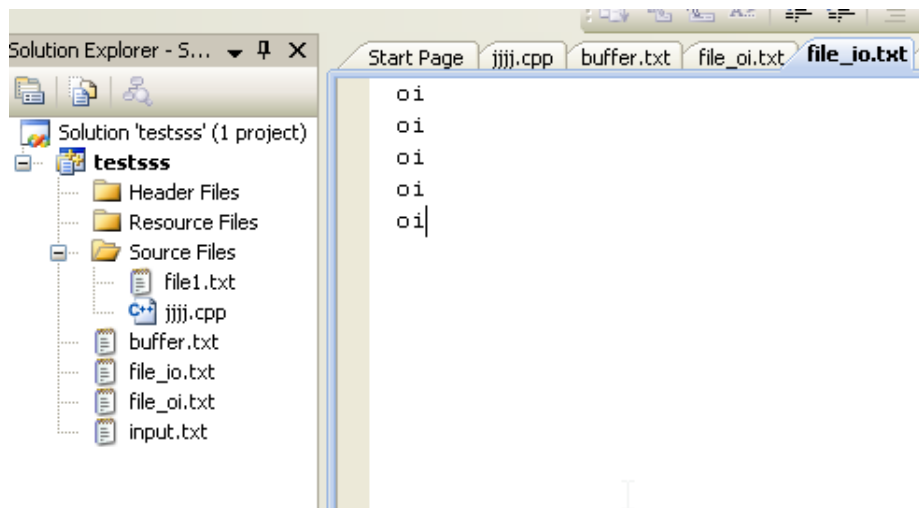
```
1 #include <iostream>
2 #include <string>
3 #include <iomanip>
4 #include <fstream>
5
6 using namespace std;
```

```

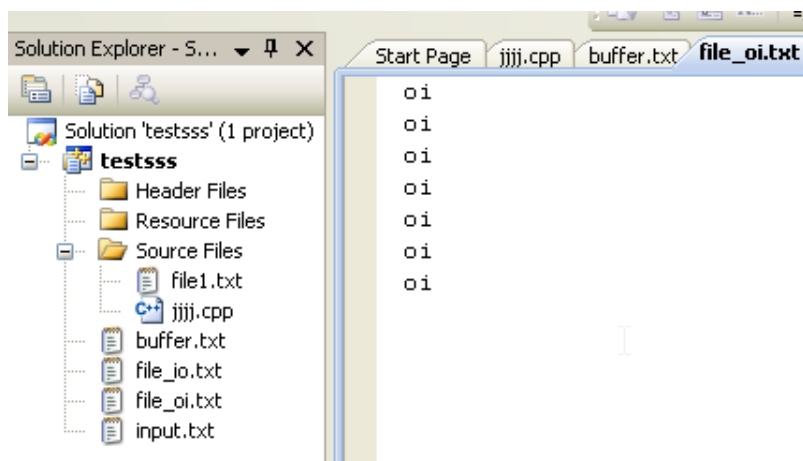
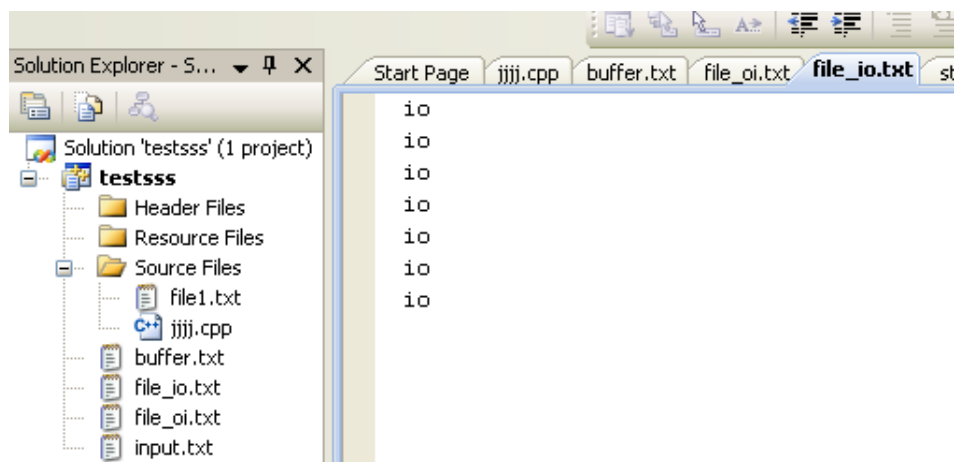
7
8
9 int main (){
10
11     setlocale (LC_ALL, "russian");
12     string x;
13     ifstream in("file_io.txt"); //для чтения
14     ofstream out("buffer.txt"); //для записи
15     while(in.peek() != EOF){ //пока не дошли до конца файла
16         in >> x;                //считываем из 1 файла
17         out << x << endl;        //записываем в буферный файл
18     }
19     in.close();                 //закрываем файлы
20     out.close();
21     ifstream in1("file_oi.txt"); //для чтения
22     ofstream out1("file_io.txt"); //для записи
23     while(in1.peek() != EOF){ //пока не дошли до конца файла
24         in1 >> x;                //считываем из 1 файла
25         out1 << x << endl;        //записываем в буферный файл
26     }
27     in1.close();
28     out1.close();
29     ifstream in2("buffer.txt"); //для чтения
30     ofstream out2("file_oi.txt"); //для записи
31     while(in2.peek() != EOF){ //пока не дошли до конца файла
32         in2 >> x;                //считываем из 1 файла
33         out2 << x << endl;        //записываем в буферный файл
34     }
35     in2.close();
36     out2.close();
37     return 0;
38
39 }

```

Первоначальные файлы:



Результат работы программы:



□

1.4. Работа с бинарными файлами

В примере, приведенном выше, видна одна из ошибок работы с текстовыми файлами — если в конце файла стоит какой-нибудь пробельный символ, то последний элемент будет считан два раза. В данном случае в начальных файлах пробельных символов нет, но при записи в новые файлы (строки 17 и 25) добавляется переход на новую строку. Именно этот символ и увеличивает количество данных в перезаписанных файлах.

Поэтому иногда полезнее использовать бинарные файлы. В отличие от текстового файла, бинарный файл представлен двоичными кодами данных, что приводит к меньшим

потерям, чем в случае текстовых файлов, так как двоичные коды определяются компьютером однозначно, в отличие от текстовых файлов. Иногда такие файлы называют типизированными, так как они содержат данные только одного типа.

Также бинарные файлы называют файлами с произвольным доступом — для того, чтобы считать, допустим пятый элемент, надо просто сдвинуть указатель на `5*sizeof(x)` позиций файла.

Считывание и запись в бинарные файлы происходит только программным путем.

Для открытия файла необходимо использовать в качестве второго параметра `ios_base::binary`:

```
ifstream in("myfile.dat", ios_base::binary)
```

Для считывания из файла используется только функция `read`:

```
in.read((char*)&x, sizeof(x));
```

Для записи в файл используется только функция `write`:

```
in.write((char*)&x, sizeof(x));
```

Для определения позиции в файле используются функции `seekg(n)` (для чтения из файла) и `seekp(n)` (для записи в файл). Функция имеет два параметра, первый — сдвиг на количество позиций, второй — относительно какой позиции сдвигать. По умолчанию сдвиг происходит на `n` позиций относительно начала файла, но существуют и другие параметры:

```
seekg(sizeof(x), ios_base::beg) //сдвиг отн. начала
```

```
seekg(sizeof(x), ios_base::end) //сдвиг отн. конца
```

```
seekg(sizeof(x), ios_base::cur) //сдвиг отн. текущей позиции
```

Для определения текущей позиции в файле используются функции `tellg()` (для файла для чтения) и `tellp()` (для файла для записи).

Пример 1.5. Даны два файла. Поменять местами их содержимое, используя дополнительный файл.

Листинг 1.5.

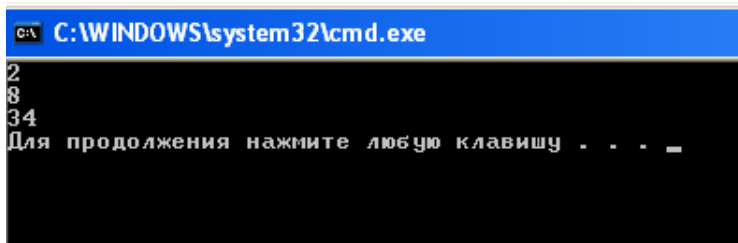
```
1 #include <iostream>
2 #include <string>
3 #include <iomanip>
4 #include <fstream>
5
6 using namespace std;
```

```

7
8
9 int main (){
10
11     setlocale (LC_ALL, "russian");
12     int a1 = 1, a2 = 1, a, n = 10;
13     ofstream out("data.dat", ios_base::binary); //для записи
14     out.write((char*)&a1, sizeof(a1)); //запись первых чисел
15     out.write((char*)&a2, sizeof(a2)); //Фибоначчи
16     for (int i = 2; i < n; i++){//запись остальных
17         a = a1 + a2;
18         a1 = a2;
19         a2 = a;
20         out.write((char*)&a, sizeof(a));
21     }
22     out.close(); //закрываем файл
23     ifstream in("data.dat", ios_base::binary); //для чтения
24     in.seekg(2*sizeof(a)); //сдвиг на третью позицию
25     while(in.peek() != EOF){
26         in.read((char*)&a, sizeof(a)); //считываем число
27         cout << a << endl;
28         in.seekg(2*sizeof(a), ios_base::cur); //сдвиг на третью позицию отн. текущей
29     }
30     return 0;
31 }

```

Результат работы программы:



```

C:\WINDOWS\system32\cmd.exe
2
8
34
Для продолжения нажмите любую клавишу . . . _

```

□