Лабораторная работа № 5.
Файловые и строковые потоки. Строки класса string

Файловые потоки

Для поддержки файлового ввода и вывода стандартная библиотека C++ содержит классы, указанные в табл. 5.1.

Таблица 5.1. Классы файловых потоков

Класс	Инстанцирован из шаблона	Базовый шаблонный класс	Назначение
`ifstream`	`basic_ifstream`	`basic_istream`	`Входной файловый поток`
`ofstream`	`basic_ofstream`	`basic_ostream`	`Выходной файловый поток`
`fstream`	`basic_fstream`	`basic_iostream`	`Двунаправленный файловый поток`

Так как классы файловых потоков являются производными от классов `istream`, `ostream` и `iostream` соответственно, то они наследуют все методы указанных классов, перегруженные операции вставки и извлечения, манипуляторы, состояние потоков и т.д. Как и стандартные потоки, файловые потоки обеспечивают гораздо более надежный ввод-вывод, чем старые функции библиотеки C. Для использования файловых потоков необходимо подключить заголовок `<fstream>`.

Работа с файлом обычно предполагает следующие операции:

- создание потока (потокового объекта);
- открытие потока и связывание его с файлом;
- обмен с потоком (ввод-вывод);
- закрытие файла.

Классы файловых потоков содержат несколько конструкторов, позволяющих варьировать способы создания потоковых объектов. *Конструкторы с параметрами* создают объект соответствующего класса, открывают файл с указанным именем и связывают файл с объектом:

```
ifstream( const char* name, int mode = ios::in );
ofstream( const char* name, int mode = ios::out | ios::trunc );
fstream ( const char* name, int mode = ios::in | ios::out );
```

Второй параметр конструктора задает режим открытия файла. Если значение по умолчанию вас не устраивает, можно указать другое, выбрав одно или несколько значений (объединенных операцией `|`) из указанных в табл. 5.2.

Таблица 5.2. Значения аргумента mode

Флаг	Назначение
ios::in	Открыть файл для ввода
ios::out	Открыть файл для вывода
ios::ate	Установить указатель на конец файла
ios::app	Открыть в режиме добавления в конец файла
ios::trunc	Если файл существует, обрезать его до нулевой длины
ios::binary	Открыть в двоичном режиме (по умолчанию используется текстовый)

Конструкторы без параметров создают объект соответствующего класса, не связывая его с файлом. В этом случае связь потока с конкретным файлом осуществляется позже — вызовом метода open, который имеет параметры, аналогичные параметрам рассмотренных выше конструкторов. Приведем примеры создания потоковых объектов и связывания их с конкретными файлами:

```
ofstream flog( "flog.txt" );            // <---- Файлы для вывода
ofstream fout1, fout2;                  // <----
fout1.open( "test1", ios::app );        // <----
fout2.open( "test2", ios::binary );     // <----
ifstream finp1( "data.txt" );           //  Файл для ввода
fstream myfile;                         //  Файл для ввода и вывода
myfile.open( "mf.dat" );
```

Если в качестве параметра name задано *краткое имя файла* (без указания полного пути), подразумевается, что файл открывается в текущем каталоге, в противном случае требуется задавать *полное имя файла*, например:

```
ifstream finp1( "D:\\VCwork\\Task1\\data.txt" );
```

ВНИМАНИЕ

Если файл не удается открыть (например, входной файл в указанном каталоге не найден или нет свободного места на диске для выходного файла), то независимо от способа его открытия — конструктором или методом open — потоковый объект принимает значение, равное нулю. Поэтому рекомендуется всегда проверять, чем завершилась попытка открытия файла, например:

```
ifstream finp1( "data.txt" );
if ( !finp1 ) { cerr << "Файл data.txt не найден." << endl;
                throw "Ошибка открытия файла "; }
```

После того как файловый поток открыт, работа с ним чрезвычайно проста: с входным потоком можно обращаться так же, как со стандартным объектом cin, а с выходным так же, как со стандартным объектом cout.

Если при чтении данных требуется контролировать, был ли достигнут *конец файла* после очередной операции ввода, используется метод eof, возвращающий нулевое значение, если конец файла еще не достигнут, и

ненулевое значение — если уже достигнут. Учтите, что в C++ после чтения из файла *последнего элемента* условие конца файла *не возникает*! Оно возникает при следующем чтении, когда программа пытается считать данные за последним элементом в файле.

Если при выполнении операций ввода-вывода фиксируется некоторая ошибочная ситуация, потоковый объект также принимает значение, равное нулю. Рекомендуется особо следить за состоянием потокового объекта во время выполнения операций вывода, так как диски «не резиновые» и имеют тенденцию переполняться.

Когда программа покидает область видимости потокового объекта, он уничтожается. При этом перестает существовать связь между потоковым объектом и физическим файлом, а физический файл закрывается. Если алгоритм требует более раннего закрытия файла, можно воспользоваться методом `close`.

Для примера работы с файловыми потоками приведем программу копирования одного файла в другой (листинг 5.1) и программу вывода на экран содержимого текстового файла (листинг 5.2).

Листинг 5.1. Копирование файлов

```
#include <iostream>
#include <fstream>
using namespace std;
void error(const char* text1, const char* text2 = "") {
    cerr << text1 << ' ' << text2 << endl;
    cin.get();  exit(1);
}
int main(int argc, char* argv[]) { // Имена файлов берутся из
командной строки
    //if (argc != 3) error("Неверное число аргументов");
     ifstream from("data.txt");                    // открываем входной
файл
    if (!from) error("Входной файл не найден:", "data.txt");
     ofstream to("data2.txt");                     // открываем выходной
файл
    if (!to) error("Выходной файл не открыт:", "data2.txt");
    char ch;
    while ( from.get(ch) ) {
  to.put(ch);
        if (!to) error("Ошибка записи (диск переполнен).");
    }
    cout << "Копирование из " << "data.txt" << " в " << "data2.txt" <<
" завершено.\n";
    cin.get();
}
```

Листинг 5.2. Вывод на экран содержимого текстового файла

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
// ...    <----- Здесь определение функции error() — из листинга 5.1

int main(int argc, char* argv[]) { // имя файла задается в командной
строке
    //if (argc != 2) error("Неверное число аргументов");
     ifstream tfile("data.txt");                    // открываем входной
файл
    if (!tfile) error("Входной файл не найден:", "data.txt");
    char buf[1024];
    while (!tfile.eof()) {
        tfile.getline(buf, sizeof(buf));
        cout << buf << endl;
    }
    cin.get();
}
```

В программе предполагается, что длина строки в файле не превышает 1024 символов.

При необходимости можно увеличить размер буфера `buf` до требуемой величины.

Строковые потоки

Работу со строковыми потоками обеспечивают классы `istringstream`, `ostringstream` и `stringstream`, которые являются производными от классов `istream`, `ostream` и `iostream` соответственно. Для использования строковых потоков необходимо подключить к программе заголовочный файл `<sstream>`.

Применение строковых потоков аналогично применению файловых потоков, но информация потока физически размещается в оперативной памяти, а не на диске. Кроме того, классы строковых потоков содержат метод `str`, возвращающий копию строки типа `string` или присваивающий потоку значение такой строки:

```
string str() const;
void str( const string& s );
```

Строковые потоки являются некоторыми аналогами функций `sscanf` и `sprintf` библиотеки C, которые также работают со строками в памяти, имитируя консольный ввод-вывод. Например, с помощью `sprintf` можно сформировать в памяти некоторую символьную строку, которую затем отобразить на экране. Эта же проблема легко решается с помощью объекта типа `ostringstream`.

В качестве примера приведем модифицированную версию листинга 5.2, которая выводит содержимое текстового файла на экран, предваряя каждую строку текстовой меткой «`Line N`», где $_N$ — номер строки (листинг 5.3).

Листинг5.3. Вывод на экран пронумерованного содержимого текстового файла

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;

//<Здесь определение функции error() — из листинга 5.1

int main(int argc, char* argv[]) {
    //if (argc != 2) error("Неверное число аргументов.");
    ifstream tfile( "data.txt" );
    if (!tfile) error("Входной файл не найден:", "data.txt");
    int n = 0;
    char buf[1024];
    while ( !tfile.eof() ) {
        n++;
        tfile.getline(buf, sizeof(buf));
        ostringstream line;
        line << "Line " << setw(3) << n << ": " << buf << endl;
        cout << line.str();
    }
    cin.get();
}
```

Строки класса string

Мы уже пользовались объектами класса `string`, начиная со второго семинара, и успели оценить удобства, обеспечиваемые этим классом в сравнении с традиционными С-строками (то есть массивами символов типа `char`, завершаемыми нулевым байтом). Сейчас мы рассмотрим строки типа `string` более подробно.

Для использования строк типа `string` необходимо подключить к программе заголовочный файл <string>. Важнейшей особенностью класса `string` является управление памятью как при размещении строки, так и при ее модификациях, изменяющих длину строки. Поэтому вы можете «забыть» об операциях `new` и `delete`, неаккуратное обращение с которыми является источником труднодиагностируемых ошибок.

Кроме этого, строки типа `string` защищены от ошибочных обращений к памяти, связанных с выходом за их границы. Но за все надо платить: строки типа `string` значительно проигрывают С-строкам в эффективности. Поэтому, если от программы требуется максимальное быстродействие, иногда лучше воспользоваться С-строками. В большинстве же программ на C++ строки типа `string` обеспечивают необходимую скорость обработки, поэтому их применение предпочтительней.

Для понимания определений методов класса string необходимо знать назначение некоторых имен. Так, в пространстве std определен идентификатор size_type, являющийся синонимом типа unsigned int. В классе string определена константа npos, задающая максимально возможное число, которое в зависимости от контекста означает либо «все элементы» строки, либо отрицательный результат поиска. Так как максимально возможное число имеет вид 0xFFFF...FFFF, то в случае присваивания его переменной типа int получится значение –1.

В классе string имеется несколько конструкторов, вот самые простые:

```
string();                    // создает пустой объект класса string
string( const char* );       //создает объект, инициализируя его
                                значением С-строки
```

Класс содержит три операции присваивания:

```
string& operator=( const string& str );// присвоить значение другой
                                          строки string
string& operator=( const char* s );// присвоить значение С-строки
string& operator=( char c );// присвоить значение символа
```

В табл. 5.3 приведены допустимые для объектов класса string операции.

Таблица 5.3. Операции класса string

Операция	Действие	Операция	Действие
=	Присваивание	>	Больше
+	Конкатенация	>=	Больше или равно
==	Равенство	[ ]	Индексация
!=	Неравенство	<<	Вывод
<	Меньше	>>	Ввод
<=	Меньше или равно	+=	Добавление

Использование операций очевидно. Размеры строк устанавливаются автоматически так, чтобы объект мог содержать присваиваемое ему значение. Нумерация элементов строки начинается с нуля. Кроме операции индексации, для доступа к элементу строки определен метод at(size_type n), который можно использовать как для чтения, так и для записи $_n$-го элемента строки:

```
cout << s.at(2);      // Будет выведен 2-й символ строки s
s.at(5) = 'W';        // 5-й символ заменяется символом W
```

Заметим, что в операции индексации не проверяется выход за диапазон строки. Метод at, напротив, такую проверку содержит, и, если индекс превышает длину строки, порождается исключение out_of_range.

В табл. 5.4 приведены некоторые употребительные методы класса string.

Таблица 5.4. Методы класса string

Метод	Назначение

`size_type size() const;`	Возвращает размер строки
`size_type length() const;`	То же, что и `size`
`insert(size_type pos1, const string& str);`	Вставляет строку `str` в вызывающую строку, начиная с позиции `pos1`
`replace(size_type pos1, size_type n1, const string& str);`	Заменяет `n1` элементов, начиная с позиции `pos1` вызывающей строки, элементами строки `str`
`string substr(size_type pos=0, size_type n=npos) const;`	Возвращает подстроку длиной n, начиная с позиции `pos`
`size_type find(const string& str, size_type pos=0) const;`	Ищет самое левое вхождение строки `str` в вызывающую строку, начиная с позиции `pos`. Возвращает позицию вхождения, или `npos`, если вхождение не найдено
`size_type find(char c, size_type pos=0) const;`	Ищет самое левое вхождение символа `c`, начиная с позиции `pos`. Возвращает позицию вхождения, или `npos`, если вхождение не найдено

Метод	**Назначение**
`size_type rfind(const string& str, size_type pos=0) const;`	Ищет самое правое вхождение строки `str`, начиная с позиции `pos`[1]
`size_type rfind(char c, size_type pos=0) const;`	Ищет самое правое вхождение символа `c`, начиная с позиции `pos`
`size_type find_first_of(const string& str, size_type pos=0) const;`	Ищет самое левое вхождение любого символа строки `str`, начиная с позиции `pos`
`size_type find_last_of(const string& str, size_type pos=0) const;`	Ищет самое правое вхождение любого символа строки `str`, начиная с позиции `pos`
`swap(string& str);`	Обменивает содержимого вызывающей строки и строки `str`
`erase(size_type pos=0, size_type n=npos);`	Удаляет `n` элементов, начиная с позиции `pos`
`clear();`	Очищает всю строку
`const char* c_str() const;`	Возвращает указатель на С-строку, содержащую копию вызывающей строки. Полученную С-строку нельзя пытаться изменить

[1] Возвращаемые значения такие же, как и у метода find.

`Size_type copy(char* s, size_type n, size_type pos=0) const;`	Копирует в массив `s` `n` элементов вызывающей строки, начиная с позиции `pos`. Нуль-символ в результирующий массив не заносится. Метод возвращает количество скопированных элементов

Поясним применение метода `find`. Допустим, что вы работаете над программой для игры в шахматы с компьютером, а в данный момент пишете функцию для ввода обозначения колонки шахматной доски. Эти колонки, как известно, обозначаются символами латинского алфавита от A до H. Желательно, чтобы ваша функция не допускала ввод некорректных символов. Приведем одно из возможных решений:

```
char GetColumn() {
    string goodChar = "ABCDEFGH";
    char symb;
    cout << "Введите обозначение колонки: ";
    while (1) {
        cin >> symb;
        if ( -1 == goodChar.find( symb ) ) {
             cout << "Ошибка. Введите корректный символ:\n"; continue;
}
        return symb;
    }
}
```

Метод `find` здесь используется для проверки, принадлежит ли введенный с клавиатуры символ множеству символов, заданному с помощью строки `goodChar`.

Задача 5.1. Подсчет количества вхождений слова в текст

Написать программу, которая определяет, сколько раз встретилось заданное слово в текстовом файле. Текст не содержит переносов слов. Максимальная длина строки в файле неизвестна[2].

Определим *слово* как последовательность алфавитно-цифровых символов, после которых следует либо знак пунктуации (., ,, ?, !), либо разделитель. В качестве разделителей могут выступать один или несколько пробелов, один или несколько символов табуляции `'\t'` и символ конца строки `'\n'`. Для хранения заданного слова (оно вводится с клавиатуры) определим переменную `word` типа `string`.

Поскольку максимальная длина строки в файле неизвестна, будем читать файл не построчно, а пословно, размещая очередное прочитанное слово в переменной `curword` типа string. Это можно реализовать с помощью

[2] Аналогичная задача решалась в первой части практикума (задача 5.2), но с упрощающим ограничением: длина строки в файле не более 80 символов.

операции >>, которая в случае операнда типа string игнорирует все разделители, предваряющие текущее слово, и считывает символы текущего слова в переменную curword, пока не встретится очередной разделитель.

Очевидно, что «опознание» текущего слова должно осуществляться с учетом возможного наличия после него одного из знаков пунктуации. Для решения этой задачи определим глобальную функцию

```
bool equal( const string& cw, const string& w );
```

которая возвращает значение true, если текущее слово cw совпадает с заданным словом w с точностью до знака пунктуации, или false — в противном случае. Имея такую функцию, очень просто составить алгоритм основного цикла:

- прочитать очередное слово;
- если оно совпадает с заданным словом w (с точностью до знака пунктуации), то увеличить на единицу значение счетчика count.

Текст решения приведен в листинге 5.4.

Листинг 5.4. Подсчет количества вхождений слова в текст

```
#include <windows.h>                // здесь прототип функции OemToChar
#include <iostream>
#include <fstream>
#include <string>
#include <clocale>                 // здесь прототип функции setlocale
using namespace std;
bool equal( const string& cw, const string& w ) {
    char punct[] = { '.', ',', '?', '!' };
    if ( cw == w ) return true;
    for ( int i = 0; i < sizeof( punct ); ++i )
        if ( cw == w + punct[i] ) return true;
    return false;
}
int main() {
    string word, curword;
    setlocale( LC_ALL, "Russian" ); // Для работы с русскими символами
    cout << " Введите слово для поиска: ";
    cin >> word;
    char pattern[30];
      OemToChar( word.c_str(), pattern ); // Для работы с русскими
символами
    ifstream fin("infile.txt");
    if ( !fin ) { cout << "Ошибка открытия файла." << endl; cin.get();
return 1; }
    int count = 0;
    while ( !fin.eof() ) {
        fin >> curword;
        if ( equal( curword, string( pattern ) ) ) count++;
    }
    cout << "Количество вхождений слова: " << count << endl;
```

```
    cin.get();  cin.get();
}
```

Обратите внимание на реализацию функции `equal` и, в частности, на использование операции сложения для добавления в конец строки `w` одного из знаков пунктуации.

Задача 5.2. Вывод вопросительных предложений

Написать программу, которая считывает текст из файла и выводит на экран только вопросительные предложения из этого текста[3].

Итак, имеется текстовый файл неизвестного размера, состоящий из неизвестного количества предложений. Предложение может занимать несколько строк, поэтому читать файл построчно неудобно. При решении аналогичной задачи в первой части практикума было принято решение выделить буфер, в который поместится *весь* файл. Такое решение тоже нельзя признать идеальным — ведь файл может иметь сколь угодно большие размеры, и тогда программа окажется неработоспособной.

Поищем более удачное решение, используя новые средства языка C++, с которыми мы познакомились на этом семинаре. Попробуем читать файл пословно, как и в предыдущей программе, в переменную `word` типа `string`, и отправлять каждое прочитанное слово в строковый поток sentence типа `ostringstream`, который, как вы уже догадались, будет хранилищем очередного предложения.

При таком подходе, однако, есть проблема, связанная с потерей разделителей при чтении файла операцией `fin >> word`. Чтобы ее решить, будем «заглядывать» в следующую позицию файлового потока `fin` с помощью метода `peek`. При обнаружении символа-разделителя его нужно отправить в поток `sentence` и переместиться на следующую позицию в потоке `fin`, используя метод `seekg`. Подробности обнаружения символа-разделителя инкапсулируем в глобальную функцию `isLimit`.

Осталось решить подзадачи:

- обнаружить конец предложения, то есть один из символов `'.'`, `'!'`, `'?'`;
- если это вопросительное предложение, вывести его в поток `cout`, в противном случае очистить поток `sentence` для накопления следующего предложения.

Рассмотренный алгоритм реализуется в листинге 5.5.

Листинг 5.5. Вывод вопросительных предложений

```
#include <iostream>
```

[3] Аналогичная задача рассмотрена в первой части практикума (задача 5.3).

```
#include <fstream>
#include <sstream>
#include <string>
using namespace std;
bool isLimit( char c ) {
    char lim[] = { ' ', '\t', '\n' };
      for ( int i = 0; i < sizeof( lim ); ++i ) if ( c == lim[i] )
return true;
    return false;
}
int main() {
    ifstream fin( "infile.txt" );
    if ( !fin ) { cout << "Ошибка открытия файла." << endl; cin.get();
return 1; }
    int count = 0;
    string word;
    ostringstream sentence;
    while( !fin.eof() ) {
        char symb;
        while( isLimit( symb = fin.peek() ) ) {
            sentence << symb;
            if ( symb == '\n' ) break;
            fin.seekg( 1, ios::cur );
        }
        fin >> word;
        sentence << word;
        char last = word[word.size() - 1];
        if ( ( last == '.' ) || ( last == '!' ) ) {
            sentence.str("");                          // очистка потока
            continue;
        }
         if ( last == '?' ) { cout << sentence.str(); sentence.str( ""
); count++; }
    }
     if ( !count ) cout << "Вопросительных предложений нет." << endl;
      cin.get();
}
```

Протестируйте приведенные программы. Не забудьте поместить в один каталог с программой текстовый файл `infile.txt`. Если программа запускается из среды `Visual Studio`, файл должен находиться в каталоге с исходными текстами проекта, если же из папки `Debug` (например, в режиме командной строки), то в этой папке.

Итоги

1. Для поддержки файлового ввода и вывода стандартная библиотека C++ содержит классы `ifstream`, `ofstream`, `fstream`.

2. Работа с файлом предполагает следующие операции: создание потока, открытие потока и связывание его с файлом, обмен с потоком

(ввод-вывод), закрытие файла. Рекомендуется всегда проверять, чем завершилось открытие файла.

3. Если в процессе ввода-вывода фиксируется ошибочная ситуация, потоковый объект принимает значение, равное нулю.

4. Следите за состоянием выходного потока после каждой операции вывода, так как на диске может не оказаться свободного места.

5. Работу со строковыми потоками обеспечивают классы `istringstream`, `ostringstream`, `stringstream`. Использование строковых потоков аналогично применению файловых потоков. Различие в том, что физически информация потока размещается в оперативной памяти, а не в файле на диске.

6. Класс `string` стандартной библиотеки C++ предоставляет программисту очень удобные средства работы со строками. Класс берет на себя управление памятью, как при размещении строки, так и при всех ее модификациях.

Задания

Выполнить задания семинара 5, используя потоковые классы.

1 Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.

2 Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.

3 Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

4 Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.

5 Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

6 Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

7 Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих из не более чем четырех буквами.

8 Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.

9 Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

10 . Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся гласными букв.

11 . Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.

12 . Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

13 . Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

14 . Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один», …, «девять», начиная каждое предложение с новой строки.

15 . Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

16 . Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.

17 . Написать программу, которая считывает текст из файла и выводит его на экран, добавляя после каждого предложения, сколько раз встретилось в нем введенное с клавиатуры слово.

18 . Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.

19 . Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.

20 . Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.