Язык программирования Си++

Иванов А.П., Князева О.С.

Семинар 7. Строковые классы Си++. Потоки ввода-вывода.

1. Строковые классы Си ++

Строковый шаблонный класс STL и его наиболее употребимые специализации (инстанцирования) определены в заголовочном файле `<string>`. Предназначен этот класс, главным образом, для удобной работы с текстовыми строками, однако, так как этот класс шаблонный, то он может использоваться и для реализации строк, состоящих не только из символов типа `char`.

Конечно, что-то похожее можно было бы реализовать и на основе `vector<char>`, однако, реализация отдельного шаблона позволила решить следующие задачи:

- расширенный набор функций, аналоги которых были определены в языке Си: поиск, сравнение, копирование строк и т.п.;
- более экономная реализация, позволяющая избежать лишних копирований и аллокаций памяти по сравнению с классом `vector` (тактика copy-on-write).

Основной шаблонный класс, в котором сосредоточена вся функциональность:

```
template <
   class CharType,
   class Traits=char_traits<CharType>,
   class Allocator=allocator<CharType>
>
class basic_string;
```

Важнейший параметр шаблона – первый, он определяет, какой тип будет сохраняться в данном контейнере. Два других параметра имеют значения по умолчанию и, как правило, задаются явно только в очень специальных случаях.

Определения производных типов (специализация)

`string`	специализация `basic_string<char>`, представляющая собой строку из обычных символов.
`wstring`	специализация `basic_string<wchar_t>`, представляющая собой строку из обычных символов в двухбайтовой кодировке Unicode.

Операторы

Часть операторов реализована как внеклассовые шаблонные функции, а другая часть – как члены класса `basic_string`.

`operator[]`	Возвращает ссылку (которой можно присваивать значение) на символ строки, имеющий указанный номер по порядку от начала.
`operator=`	Оператор присваивания строке, имеет несколько разных реализаций по

	типу аргумента: присваивание другой строки, одиночного символа и ли строки языка Си типа `char*`.
`operator+` `operator+=`	Добавление второй строки в конец первой. Аналогично оператору присваивания – допускает аргументы разных типов (Си++ строка, Си строка, одиночный символ).

`operator!=`
`operator==`
`operator<`
`operator<=`
`operator>`
`operator>=`
	Операторы сравнения двух строк.

`operator<<` `operator>>`	Операторы ввода и вывода строки в поток (например, `cin` или `cout`).

Отдельная функция

`getline`	Позволяет читать входной поток (такой, как `cin`) построчно, присваивает очередную считанную строку указанной переменной:
	`string s1;`
	`cout << "Enter a sentence:" << endl << flush;`
	`getline(cin,s1);`

Определения типов, встроенные в класс

`iterator` `const_iterator`	Аналогично прочим STL-контейнерам, со строкой можно оперировать при помощи итераторов (обобщения указателей).
`npos`	Специальный тип (эквивалентный значению целочисленной «минус единице»), предназначенный для сигнализации о неуспехе поиска в строке.

Конструкторы

`basic_string`	Конструкторы, в зависимости от типа своего аргумента позволяют создать строку из строки Си, другой Си++ строки, одиночного символа или заданного их количества, а также – создать пустую строку.

Member Functions

`size, length`	Возвращают текущее количество символов в строке.
`empty`	Проверка на пустую строку.
`resize`	Изменение размера строки: в зависимости от соотношения старого и нового размеров либо уничтожаются лишние символы в конце строки, либо специально указываемый символ добавляется в нужном количестве в конец строки..
`reserve`	Резервирует размер внутреннего буфера строки не менее, чем до

`capacity`	указанного количества символов. Возвращает текущий размер зарезервированного внутреннего буфера строки, который может быть использован под сохранение ее символов без вызова методов аллокации памяти.
`assign` `append`	Аналогично `operator=` и `operator+=` присваивают или дописывают в конец данной строки – другую строку, однако, в отличие от соответствующих операторов, содержат по нескольку вариантов с дополнительными аргументами, позволяющими выполнить копирование только части строки.
`at`	Как `operator[]` возвращает ссылку на символ строки с заданным номером, однако, в отличие от него выполняет строгую проверку индекса на попадание в диапазон допустимых значений и выставляет исключительное состояние в случае неуспеха.
`begin` `end`	Возвращают итераторы для позиций первого символа и символа, находящегося за последним значащим символом строки.
`c_str`	Метод возвращает указатель на обычную Си-строку, терминированную символом `'\0'`.
`data`	Аналогичный метод, возвращает указатель на массив символов, соответствующей текущей строке, но без обязательного терминирования их символом `'\0'`.
`erase`	Удаляет из строки отдельный символ или диапазон символов, начиная с указанной позиции и указанной длины.
`clear`	Уничтожает все символы строки, делает ее пустой.
`compare`	Лексикографическое сравнение двух строк.
`copy`	Копирует указанный диапазон символов строки во внешний массив символов.
`insert`	Вставляет в указанной позиции одиночный элемент или диапазон символов.
`push_back`	Добавляет указанный символ в конец строки.
`replace`	Заменяет указанный символ или диапазон символов в строке.
`substr`	Извлекает из строки подстроку указанной длины, начиная с указанной позиции.
`find`	Ищет в строке указываемую подстроку.
`find_first_not_of`	Ищет в строке символ, не являющийся ни одним из указанных.
`find_first_of`	Ищет в строке символ, совпадающий с одним из указанных.

`find_last_not_of`	Ищет в строке последний символ, который не совпадает ни с одним из указанных.
`find_last_of`	Ищет в строке последний символ, совпадающий с одним из указанных.
`rfind`	Проводит в строке поиск от конца к началу указанной подстроки.

2. Потоки ввода-вывода

В качестве альтернативы к стандартной библиотеке ввода-вывода `<stdio.h>`, язык Си++ предлагает объектно-ориентированный потоковый ввод-вывод. Для использования этой возможности необходимо подключить к исходному тексту программы файл `<iostream>`.

Аналогом стандартных файлов ввода-вывода в Си (`stdin`,`stdout`,`stderr`) служат потоки `cin, cout, cerr`. Эти потоки являются объектами классов, производных от `istream` для потоков ввода и `ostream` для потоков вывода. Существует так же определение класса `iostream` для потока ввода-вывода.

Для осуществления операций ввода-вывода нужно воспользоваться переопределенными операторами >> для операции ввода и << для вывода.

```
#include <iostream>

void main() {
      int n;
      cout << "Hello, world!\n" << "Enter number: ";
      cin >> n;
      cout << "\nYou entered: " << n;
}
```

Операции ввода-вывода определены для всех базовых типов. При этом используется формат «по умолчанию», когда для вывода, например, числа плавающей арифметики печатается полностью его текстовое представление с некоторым фиксированным числом значащих цифр.

Операторы вывода определены в классе `ostream` следующим образом:

```
class ostream : {
//.................................................
      ostream& operator<< (char);
      ostream& operator<< (int);
      ostream& operator<< (long);
      ostream& operator<< (float);
      ostream& operator<< (double);
      ostream& operator<< (const char *);
//.................................................
}
```

Каждый такой оператор возвращает ссылку на текущий поток вывода. Это дает возможность сцеплять несколько операций вывода в один оператор языка, сохраняя естественный порядок: вывод параметров будет осуществляться слева направо.

Не следует забывать, что данные операторы имеют стандартный приоритет языка Си. Следовательно иногда для корректной записи необходимы скобки:

```
int a,b, mask = 0x0F;
cout << a+b << (a & mask) << " Correct!";
```

Поток ввода реализован аналогичным образом.

Для того, чтобы обеспечить вывод типов данных, введенных пользователем, надо определить для них оператор вывода, например, следующим образом:

```
class CDate {
private:
      int day, month, year;
public:
      CDate( int day_in, int month_in, int year_in );
      //.........................
      friend ostream& operator<<(ostream& s, CDate& date);
};
//.....................
ostream& operator<<(ostream& s, CDate& date) {
      return s << date.day << "." << date.month << "." << date.year-2000;
}
//.....................
CDate today(08,05,2010);
cout << "\nToday = " << today;

Результат:
Today = 08.05.10
```

Форматирование вывода

По сравнению с функцией `printf()` базовые потоки предоставляют гораздо меньше возможностей по форматированию вывода. Основное форматирование осуществляется с помощью функций-манипуляторов входными потоками.

Основные манипуляторы:

```
// Вставка в поток символа конца строки '\n' и сброс потока
ostream& endl(ostream&);

// Вставка в поток символа null для окончания строки
ostream& ends(ostream&);

// Сброс потока, выводятся все символы из внутреннего буфера.
ostream& flush(ostream&);

// В потоке ввода будут пропущены все пробельные символы:
// (' ','\n','\r','\t'...)
istream& ws(istream &);
```

Следующие манипуляторы имеют действие до тех пор, пока в данном потоке не будет применен другой манипулятор, отменяющий его действие:

```
// Установка режима десятичного вывода целых чисел
ios&  dec(ios &);
```

```
// Установка режима шестнадцатиричного вывода целых чисел
ios& hex(ios &);

// Установка режима восьмиричного вывода целых чисел
ios&  oct(ios &);
```

Примеры применения манипуляторов:

```
cout << "Следующая строка напечатается с новой строки: " << endl << 
"Шестнадцатиричное число: " << hex << 256 <<" Снова десятичное: " << dec << 
256;
```

Результат:

```
Следующая строка напечатается с новой строки:
Шестнадцатиричное число: FF Снова десятичное: 256
```

Заметим, что мы не вызывали функцию-манипулятор явно, а передали лишь указатель на нее.

Дополнительные манипуляторы определены в файле <iomanip>:

```
// Установка режима восьмиричного вывода целых чисел:
smanip_int  setfill(int _f);

// Установка ширины поля. По умолчанию - вывод всего значения.
// Влияет только на ближайшую операцию вывода:
smanip_int  setw(int _n);

// Установка точности вывода чисел плавающей арифметики
// в n чисел после запятой:
smanip_int  setprecision(int n);
```

Пример:

```
cout << "Int: " << setfill('0') << setw(8) << 123 << "!" << endl;
cout << setfill(' ') << "Float: " << setprecision(5) << M_PI << endl;
```

Результат:

```
Int: 00000123!
Float: 3.14159
```

В этом же файле вводятся все необходимые декларации, необходимые для написания программистом своих манипуляторов.

Файловый ввод-вывод

Помимо предопределенных потоков, программист может заводить свои потоки ввода-вывода, ассоциированные как с уже существующими потоками, так и с файлами операционной системы или со строками в оперативной памяти компьютера.

Для того, чтобы ассоциировать некоторый поток с файлом нужно выполнить следующие действия:

```
#include <fstream>

// Заводим объект класса filebuf.
filebuf outfile;
// Открываем файл, режим - запись
if( !outfile.open( "c:\\filename.txt", ios::out ) ) {
     cerr << "Файл не открыт!";
     abort();
}
// Заводим объект класса ostream, ассоциированный с outfile
ostream cdst(&outfile);
cdst << "Ready";  // Осуществляем вывод.
```

Открытый файл будет закрыт в момент вызова деструктора объекта `cdst`.

Для открытия входного потока необходимо указать режим `ios::in` при открытии `filebuf`. Для открытия файла на чтение-запись режимы открытия можно комбинировать:

```
outfile.open( "c:\\filename.txt", ios::out | ios::out )
```

Прочие режимы открытия файла:

`app`	перед каждой операцией вывода позиционироваться в конец потока;
`ate`	позиционироваться в конец потока в момент создания объекта `filebuf`;
`trunk`	удалить содержимое открываемого файла при его открытии, сделать файл пустым;
`binary`	установить бинарный режим чтения/записи, аналогичный бинарному режиму функций чтения/записи языка C: без трансляции `\n <-> \r\n`.

Существует конструктор класса `filebuf`, которому можно указать дескриптор открытого файла:

```
filebuf::filebuf(int hFile);
```

Кроме того, есть возможность использовать специальные классы для файлового ввода-вывода, описанные в файле `<fstream>`:

```
#include <fstream>

ifstream fmyin("c:\\myfile1.dat" );        // Поток ввода
ofstream fmyout("c:\\myfile2.dat" );       // Поток вывода
fstream  fmyinout("c:\\myfile3.dat" );     // Поток ввода-вывода
```

В последнем случае второй аргумент (режим открытия для чтения-записи) подставлен по умолчанию, но, если нужно, его можно указать явно:

```
fstream fmyout2("c:\\myfile3.dat", ios::out ); // Только вывод
```

Естественно, обязательно нужно проверить – открылся ли файл или произошла ошибка:

```
if ( fmyin.is_open() ) {
   int x, y;
   fmyin >> x >> y;
   ............
}
```

Потоковым вводом-выводом можно пользоваться и для чтения/записи форматированных строк. Для этого объект специального класса ассоциируется со строкой:

```
#include <strstream>

char A[1024];

istrstream smyin( A, 1024 );                          // Поток ввода
ostrstream smyout( A, 1024 );                         // Поток вывода
strstream  smyinout( A, 1024, ios::in | ios::out );  // Поток ввода-вывода

smyout << "PI= " << 3.14159 << endl << ends;
```

В заголовочном файле `<sstream>` содержатся объявления аналогичных потоков ввода-вывода для класса `string`.

1. Вариант

Создать функцию подсчета суммы чисел в файле. Функция принимает в качестве параметра имя файла. Числа отделены друг от друга, словами, буквами, символами, знаками табуляции.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

Пример: строка в файле может выглядеть следующим образом: «10sdada350re `20sd100%dasd^#^#1sdas15`», сумма в ней чисел соответственно равна 496.

2. Вариант

Создать функцию, удаляющую комментарии из файлов Си и Си++. Функция принимает в качестве параметра имя входного и выходного файлов и удаляет из файла все части строки, следующие за символами «`//`», и все куски теста заключенные в символы «`/*…*/`», результат записывается в выходной файл.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

3. Вариант

Создать функцию, которая зашифровывает файл на основании файла-ключа. Функция принимает в качестве параметров имена входного файла, файла-ключа и выходного файла. Функция может реализовать шифр, например, выдавая положение букв исходного файла в файле-ключе.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

4. Вариант

Создать функцию, которая расшифровывает зашифрованный файл на основании файла-ключа. Функция принимает в качестве параметров имена входного файла, файла-ключа и выходного файла. Функция может реализовать расшифровку, например, выдавая буквы, координаты которых указаны в файле-ключе.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

5. Вариант

Создать функцию, которая должна считать массив чисел из файла, отсортировать и записать в другой файл. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа vector или list.

6. Вариант

Создать функцию, которая транспонирует матрицы. Матрица считывается из текстового файла и записывается в другой файл. Размеры матрицы заранее не известны. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа `vector` или `list`.

7. Вариант

Создать функцию, которая считывает из файла числовые данные, записанные в 2 колонки, и переписывает их в другой файл в одну строку через запятую в обратном порядке (то есть – снизу вверх). Функция принимает имена входного и выходного файлов.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны считываться в контейнер `string` библиотеки STL.

8. Вариант

Создать функцию поиска подстроки в текстовом файле. Функция принимает имя входного файла и искомую подстроку.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

9. Вариант

Создать функцию, убирающую из текстового файла все цифры. Функция принимает имена входного и выходного файлов.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

10. Вариант

Создать функцию, подсчитывающую количество слов в текстовом файле. Функция принимает имя входного файла и возвращает количество слов.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

11. Вариант

Создать функцию, подсчитывающую количество букв в текстовом файле. Функция принимает имя входного файла и возвращает количество букв.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

12. Вариант

Создать функцию обрабатывающую текстовый файл. В файле имеется матрица произвольной размерности. Функция считывает эту матрицу и записывает в выходной файл ее строки, отсортированные по возрастанию значений в первой колонке. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа `vector` или `list`.

13. Вариант

Создать функцию обрабатывающую текстовый файл. В файле имеется массив x-координат, функция считывает эти значения и записывает в выходной файл `x` и `log8(x)` в 2 колонки. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа `vector` или `list`.

14. Вариант

Создать функцию для определения встречаемости символов в тексте. Функция принимает имена входного и выходного файлов, обрабатывает входной файл и записывает результат в выходной файл в виде: «`. – 100, ! – 4, a – 250, b – 75`» и т.д.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер `string` библиотеки STL.

15. Вариант

Создать функцию для определения встречаемости слов в тексте. Функция принимает имена входного и выходного файлов, обрабатывает входной файл и записывает результат в выходной файл в виде: «`из – 100, стол – 4, идти – 250`» и т.д.

Файл должен считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер `string` библиотеки STL.

16. Вариант

Создать функцию архивации файлов. Имеется несколько файлов, необходимо слить их в один следующим образом: сначала идут названия всех файлов и их размеры, затем данные из этих файлов, разделенные, например, строками вида `##########################`. Функция принимает список (STL контейнер `list<string>`) содержащий имена входных файлов и имя выходного файла.

Файлы должны считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер string библиотеки STL.

17. Вариант

Создать функцию разархивации файлов. Несколько файлов были слиты в один следующим образом: сначала идут названия всех файлов и их размеры, затем данные из этих файлов, разделенные, например, строками вида `##########################`. Надо разделить этот файл обратно на несколько файлов. Функция принимает название входного файла.

Файлы должны считываться по строкам с использованием функций библиотеки `<fstream>`, строки должны записываться в контейнер `string` библиотеки STL.

18. Вариант

Создать функцию обрабатывающую текстовый файл. В файле имеется таблица пар x- и y-координат, упорядоченных по возрастанию x. Функция получает значение произвольной точки x (не обязательно совпадающее с любым x из таблицы в файле), считывает значения из файла и с помощью линейной интерполяции/экстраполяции по соседним строкам таблицы вычисляет значение y, соответствующее указанному x. Функция принимает значение x и имя входного файла.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа `vector` или `list`.

19. Вариант

Создать функцию обрабатывающую текстовые файлы. В двух файлах записаны квадратные матрицы заранее неизвестной размерности. Функция считывает эти матрицы и записывает в выходной файл их произведение (при необходимости – размерности матриц дополняются рядами и колонками с нулевыми значениями). Функция принимает имена входных и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`. Значения считываются в STL контейнер типа `vector` или `list`.

20. Вариант

Создать функцию обрабатывающую текстовый файл. В файле записаны арифметические выражения вида:

```
5+7/2-5 =
3-3*2 =
```

Скобки не используются, все операции выполняются слева направо. Функция должна вычислить результат каждого выражения и создать новый файл, вида:

```
5+7/2-5 = 3.5
3-3*2=-3
```

Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

21. Вариант

Создать функцию обрабатывающую текстовый файл. В файле записана база данных студентов в виде:

```
[1]
фамилия: Петров
имя: Иван
год рождения: 1990
группа: 101
[2]
фамилия: Сидоров
имя: Сергей
год рождения: 1991
группа: 108
[3]
фамилия: Иванов
имя: Евгений
год рождения: 1991
группа: 101
```

и т.д.

Необходимо считать базу данных и переписать ее в другой файл в виде:

```
Группа 101:
[1] Петров Иван (1990)
[3] Иванов Евгений (1991)

Группа 108:
[2] Сидоров Сергей (1991)
```

Номера групп должны идти в возрастающем порядке. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

22. Вариант

Создать функцию обрабатывающую текстовый файл. В файле записана база данных студентов в виде:

```
[1]
фамилия: Петров
имя: Иван
средний балл: 4
группа: 101
[2]
фамилия: Сидоров
имя: Сергей
средний балл: 4.8
группа: 108
[3]
фамилия: Иванов
имя: Евгений
средний балл: 3.1
группа: 101
```

и т.д.

Необходимо считать базу данных и переписать ее в другой файл в виде:

```
[Группа 101]
Количество студентов: 2
Средний балл группы: 4.4
[Группа 108]
Количество студентов: 1
Средний бал группы: 3.1
```

Номера групп должны идти в возрастающем порядке. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

23. Вариант

Создать функцию обрабатывающую текстовый файл. В файле записана база данных студентов в виде:

```
Группа 101:
[1] Петров Иван (1990)
[3] Иванов Евгений (1991)

Группа 108:
[2] Сидоров Сергей (1991)
```

Необходимо считать базу данных и переписать ее в другой файл в виде:

```
[1]
фамилия: Петров
имя: Иван
год рождения: 1990
группа: 101
[2]
фамилия: Сидоров
имя: Сергей
год рождения: 1991
группа: 108
[3]
фамилия: Иванов
имя: Евгений
год рождения: 1991
группа: 101
```

Студенты должны перечисляться в алфавитном порядке фамилий. Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

24. Вариант

Создать функцию обрабатывающую текстовый файл. В файле записана база данных студентов в виде:

```
Группа 101:
[1] Петров Иван (1990)
[3] Иванов Евгений (1991)

Группа 108:
[2] Сидоров Сергей (1991)
```

Необходимо считать базу данных и переписать ее в другой файл в виде списка студентов, отсортированных по фамилиям в виде:

```
Фамилия     Имя         Группа     Год рождения
Иванов      Евгений     101        1991
Петров      Иван        101        1990
Сидоров     Сергей      108        1991
```

Функция принимает имена входного и выходного файлов.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.

25. Вариант

Создать функцию обрабатывающую конфигурационный файл вида:

```
Data
{
Temperature = 273
Pressure = 1e+5
N= 100
}
Constants
{
kB=1.38e-23
}
A_material
{
N=100
}
B_material
{
N=50
}
```

и т.д.

Функция принимает название файла, название блока (см. пример выше: у `Data` или `A_material` блоки ограничены фигурными скобками) и параметра (`Pressure` или `N`) и возвращает значение параметра.

Работа с файлом должна осуществляться с помощью функций библиотеки `<fstream>`.