STL
для программистов
на C++

STL for C++ Programmers

Leen Ammeraal
Hogeschool Utrecht, The Netherlands

JOHN WILEY & SONS
Chichester • New York • Weinhwim • Brisbane • Toronto • Sungapore

STL для программистов на C++

Леен Аммерааль
Высшая школа Утрехта, Нидерланды

Москва

Леен Аммерааль
STL для программистов на C++. Пер. с англ./Леен Аммерааль – М.: ДМК. – 240 с., ил.

ISBN 5-89818-027-3

Книга Леена Аммерааля посвящена стандартной библиотеке шаблонов (STL) – мощному инструменту повышения эффективности труда программистов, пишущих на C++.

Умелое использование STL позволяет повысить надежность, переносимость и универсальность программ, а также снизить расходы на их разработку. В книге описана стандартизованная версия STL. Дается введение в предмет, которое позволяет быстро освоить библиотеку шаблонов. Приведен исчерпывающий справочный материал, в том числе по новому классу STL, string. Изложение сопровождается многочисленными примерами небольших, но законченных программ, иллюстрирующих ключевые понятия STL. Особое внимание уделено разъяснению сложных понятий библиотеки шаблонов, например, функциональных объектов и адаптеров функций.

Книга предназначена как для профессиональных программистов и тех, кто углубленно изучает C++, так и для тех, кто только начинает осваивать этот язык программирования, без преувеличения самый популярный в мире.

All Rights Reserved. Authorized translation from the English language edition published by John Wiley & Sons, Inc.

Все права сохранены. Никакая часть настоящего издания не может быть воспроизведена, сохранена в воспроизводящей системе либо перенесена в какой бы то ни было форме, с использованием электронных, механических, фотокопировальных, записывающих, сканирующих или любых других приспособлений без предварительного письменного разрешения со стороны издателя.

В соответствии с Законом об авторском праве, проектах и патентах от 1988 г. Леен Аммерааль заявлен обладателем права считаться автором данной работы.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность наличия технических и просто человеческих ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 0-471-97181-2

ISBN 5-89818-027-3

© John Wiley & Sons Ltd,
Baffins Lane, Chichester,
West Sussex PO19 1UD, England

© ДМК

Содержание

Предисловие 7
1. STL для начинающих 8
1.1. Шаблоны, пространства имен и тип bool 8
1.2. Знакомство с STL 14
1.3. Векторы, списки и двусторонние очереди 20
1.4. Сортировка 24
1.5. Алгоритм find 29
1.6. Алгоритм copy и итератор вставки 30
1.7. Алгоритм merge 32
1.8. Типы, определенные пользователем 34
1.9. Категории итераторов 35
1.10. Алгоритмы replace и reverse 41
1.11. Возвращаясь к алгоритму sort 42
1.12. Введение в функциональные объекты 43
1.13. Использование find_if, remove и remove_if 45
1.14. Класс auto_ptr 49
2. Другие алгоритмы и контейнеры 52
2.1. Алгоритм accumulate 52
2.2. Алгоритм for_each 54
2.3. Подсчет 55
2.4. Функциональные объекты, определенные в STL 57
2.5. Введение в ассоциативные контейнеры 59
2.6. Множества и множества с дубликатами 60
2.7. Словари и словари с дубликатами 62
2.8. Пары и сравнения 65
2.9. Снова словари 67
2.10. Функции insert 72
2.11. Удаление элементов словаря 74
2.12. Более удобные строки 74
3. Последовательные контейнеры 81
3.1. Векторы и связанные с ними типы 81
3.2. Функции capacity и reserve 85
3.3. Обзор функций-членов класса vector 88
3.4. Двусторонние очереди 92
3.5. Списки 94
3.6. Векторы векторов 101
3.7. Как избавиться от явного выделения памяти 103
4. **Ассоциативные контейнеры** 107
4.1. Введение 107
4.2. Функции-члены множеств 111
4.3. Объединение и пересечение множеств 117
4.4. Отличия множеств с дубликатами от просто множеств 119
4.5. Словари 120
4.6. Словари с дубликатами 124
4.7. Сводный указатель 127
5. Адаптеры контейнеров 131
5.1. Стеки 131
5.2. Очереди 134
5.3. Очереди с приоритетами 135

6. Функциональные объекты и адаптеры ... 138
6.2. Функциональные объекты ... 138
6.2. Унарные предикаты и привязки ... 142
6.3. Отрицатели ... 143
6.4. Два полезных базовых класса STL ... 145
6.5. Функциональные объекты и алгоритм transform ... 147
6.6. Адаптеры итераторов ... 150
7. Обобщенные алгоритмы ... 155
7.1. Немодифицирующие последовательные алгоритмы ... 156
7.1.1. Алгоритмы find, count, for_each, find_first_of и find_end ... 156
7.1.2. Алгоритм adjacent_find ... 158
7.1.3. Отличие ... 160
7.1.4. Сравнение на равенство ... 161
7.1.5. Поиск подпоследовательности ... 162
7.2. Модифицирующие последовательные алгоритмы ... 163
7.2.1. Преобразовать ... 163
7.2.2. Копировать ... 164
7.2.3. Переместить по кругу ... 166
7.2.4. Обменять ... 168
7.2.5. Заменить ... 170
7.2.6. Удалить ... 172
7.2.7. Заполнить ... 172
7.2.8. Породить ... 173
7.2.9. Убрать повторы ... 175
7.2.10. Расположить в обратном порядке ... 178
7.2.11. Перетасовать ... 178
7.2.12. Разделить ... 180
7.3. Алгоритмы, связанные с сортировкой ... 181
7.3.1. «Меньше» и другие операции сравнения ... 182
7.3.2. Сортировка ... 182
7.3.3. Стабильная сортировка ... 182
7.3.4. Частичная сортировка ... 184
7.3.5. N-й элемент ... 186
7.3.6. Двоичный поиск ... 187
7.3.7. Объединение ... 189
7.3.8. Операции над множествами для сортированных контейнеров ... 191
7.3.9. Операции над пирамидами ... 194
7.3.10. Минимум и максимум ... 197
7.3.11. Лексикографическое сравнение ... 199
7.3.12. Генераторы перестановок ... 200
7.4. Обобщенные численные алгоритмы ... 202
7.4.1. Накопление ... 202
7.4.2. Скалярное произведение ... 202
7.4.3. Частичная сумма ... 204
7.4.4. Разность между смежными элементами ... 205
7.5. Прикладная программа: метод наименьших квадратов ... 206
8. Прикладная программа: очень большие числа ... 211
8.1. Введение ... 211
8.2. Реализация класса large ... 215
8.3. Вычисление числа π ... 229
Библиография ... 235
Указатель идентификаторов и английских названий ... 236
Предметный указатель ... 238

Предисловие

Когда несколько лет назад в языке C++ появились шаблоны, лишь немногие из программистов на C++ могли предположить, какое влияние это окажет на стандарт библиотеки языка. Стандартная библиотека шаблонов (Standard Template Library) была первоначально разработана сотрудниками Hewlett-Packard А.А. Степановым и М. Ли совместно с Д.Р. Муссером из Ренсселэровского политехнического института. После внесения незначительных поправок Комитет по стандартизации языка C++ принял STL, сделав ее существенной составной частью стандартной библиотеки.

Использование STL дает возможность создавать более надежные, более переносимые и более универсальные программы, а также сократить расходы на их разработку. Это значит, что ни один профессиональный программист не может себе позволить пройти мимо этой библиотеки. Я написал книгу для таких программистов, а также для людей, в достаточной мере знакомых с C++.

В книге описана стандартная версия STL, а не изначальный вариант, разработанный в Hewlett-Packard. Вы можете загрузить примеры, приводимые в этой книге, из сети Internet не только для того, чтобы сэкономить на набивании кода, но и для упрощения проблем с переносимостью: некоторые из электронных версий примеров сделаны более переносимыми путем добавления условной компиляции, чтобы обойти нестандартное поведение версий STL, поставляемых с компиляторами Borland и Microsoft. Все примеры доступны в виде одного файла, *stlcpp.zip*, на моем Web-сайте по адресу:

http://www.econ.hvu.nl/~ameraal/

или напрямую с одного из следующих ftp-сайтов:

ftp://ftp.expa.fnt.hvu.nl/pub/ameraal

ftp://pitel-lnx.ibk.fnt.hvu.nl/pub/ameraal

Я благодарен Гэйнору Редверс-Маттону из издательства Wiley и Фрэнсису Глассброу из Ассоциации пользователей C и C++, которые убедили меня написать эту книгу и дали полезные рекомендации по ее содержанию.

1

STL для начинающих

1.1. Шаблоны, пространства имен и тип *bool*

Как легко догадаться из названия, стандартная библиотека шаблонов (STL) основывается на относительно новом понятии *шаблона*. Поэтому мы начнем с краткого обсуждения этого предмета.

Шаблонные функции

Предположим, что для некоторого положительного числа *x* нам приходится часто вычислять значение выражения

```
2 * x + (x * x + 1) / (2 * x)
```

где *x* может быть типа *double* или *int*. В последнем случае оператор деления / обозначает целочисленное деление, дающее целый результат. Например, если *x* имеет тип *double* и равен 5.0, тогда значение приведенного выражения составляет 12.6, но если *x* имеет тип *int* и равен 5, то значение выражения будет 12. Вместо того чтобы писать две функции, такие как

```
double f(double x)
{  double x2 = 2 * x;
   return x2 + (x * x + 1)/x2;
}
```

```
int f(int x)
{  int x2 = 2 * x;
   return x2 + (x * x + 1)/x2;
}
```

нам достаточно создать один *шаблон*, как показано в следующем примере, который представляет собой законченную программу:

```
// ftempl.cpp: Шаблонная функция.
#include <iostream.h>

template <class T>
T f(T x)
{  T x2 = 2 * x;
   return x2 + (x * x + 1)/x2;
}

int main()
{  cout << f(5.0) << endl << f(5) << endl;
   return 0;
}
```

Программа выведет

```
12.6
12
```

В этом шаблоне *T* – тип, задаваемый аргументом при вызове *f*. При вызове *f*(5.0) T будет обозначать *double* (это тип константы 5.0), так что, к примеру, в выражении (*x* * *x* + 1)*/x2* выполнится деление с плавающей точкой. Напротив, при исполнении вызова *f* (5) T будет обозначать тип *int*, что приведет к целочисленному делению.

При разборе программы *ftempl.cpp* компилятор создает две различные функции, весьма похожие на функции *f*(*double*) и *f*(*int*), с которых мы начинали наш пример. Следовательно, компилятор должен одновременно «видеть» как определения, так и вызовы шаблонов. Это делает шаблоны плохими кандидатами на раздельную компиляцию; вместо этого мы, как правило, помещаем шаблоны в файлы заголовка. Когда мы используем файлы заголовка, написанные кем-то другим, мы не видим определения шаблонов и вызываем их как обычные функции, что показано на примере вызовов *f*(5.0) и *f*(5) в нашей программе. Поэтому, применяя шаблоны функций STL, мы можем и не знать, что вызываем функции, созданные из шаблонов.

Что в имени?

Шаблон, который, подобно рассмотренному выше, начинается со слова *template*, а заканчивается закрывающей фигурной скобкой, следующей

за оператором *return*, создателем языка Бьерном Страуструпом был изначально назван *шаблоном функции*. Данный термин отражает, что мы имеем дело с определенным видом шаблона, отличающимся от шаблонов классов, о которых речь пойдет ниже. Сегодня многие авторы используют вместо этого выражение *шаблонная функция (template funcion)*, потому что шаблоны указанного типа очень похожи на обычные функции. В этой книге мы также будем использовать термин *шаблонная функция*, а иногда даже просто *функция* для обозначения этих шаблонов. То же относится и к обсуждаемому ниже понятию, которое изначально получило наименование *шаблон класса*, а в книге зовется *шаблонным классом (template class)* или просто *классом*.

Шаблонные классы

Мы можем использовать тип как параметр (*T* в предыдущем примере) для классов почти так же, как и для функций. Предположим, нам нужен класс *Pair*, чтобы хранить пары значений. Иногда оба значения принадлежат к типу *double*, иногда к типу *int*. Тогда вместо двух новых классов, к примеру,

```
class PairDouble {
public:
   PairDouble(double x1, double y1): x(x1), y(y1) {}
   void showQ();
private:
   double x, y;
};

void PairDouble::showQ()
{  cout << x/y << endl;
}
```

после чего следует аналогичный фрагмент с классом *PairInt*, нам достаточно написать один *шаблонный класс*:

```
// cltempl.cpp: Шаблонный класс.
#include <iostream.h>

template <class T>
class Pair {
public:
   Pair(T x1, T y1): x(x1), y(y1){}
   void showQ();
private:
   T x, y;
};

template <class T>
```