Введение в библиотеку STL

Дмитрий Рамодин

16.09.1997

Журнал "Мир ПК", #09, 1997 год

Итераторы

Алгоритмы

Контейнеры

Функциональные объекты

Листинг

Библиотека стандартных шаблонов STL (Standard Template Library) - один из тех немногих программных продуктов, чье появление было встречено единодушно всеми профессиональными программистами.

Практически все современные компиляторы Cи++ содержат библиотеку STL. Изначально она настроена на работу практически с любыми данными, что обеспечивается набором шаблонов функций и классов.

Четыре основных компонента составляют структуру STL:

итераторы - специальные указатели, позволяющие алгоритмам перемещаться по данным контейнера;

алгоритмы - вычислительные процедуры;

контейнеры - блоки хранения данных, управления ими и размещения;

функциональные объекты - инкапсулируют функции в объекте для использования другими компонентами.

Итераторы

Итератор - это некий обобщенный указатель. Обычные указатели языка Cи++ являются частным случаем итераторов, позволяющих работать с различными структурами данных и типами универсальным способом. Любой алгоритм (универсальная вычислительная процедура), принимая в качестве параметров итераторы, при их обработке не задумывается о типе данных, на которые передаваемые итераторы ссылаются.

Итераторы бывают пяти видов:

входные (input);

выходные (output);

однонаправленные (forward);

двунаправленные (bidirectional);

произвольного доступа (random access).

Входные итераторы служат для чтения адресуемых данных. Выходные, напротив, адресуют объекты, в которые данные должны быть записаны. Однонаправленные итераторы обладают всеми свойствами входных и выходных, а также могут перемещаться от начала последовательности адресуемых данных в конец. Что касается двунаправленных итераторов, то они не только обладают свойствами однонаправленных, но и способны перемещаться в любом направлении по цепочке данных: как вперед, так и назад. Итераторы произвольного доступа самые универсальные. Они обладают функциональностью всех четырех других итераторов. Кстати говоря, указатели языка Си++ также являются итераторами произвольного доступа.

Библиотека STL построена так, что итератор более старшего типа может быть подставлен вместо младшего. Так, итератор произвольного доступа может заменить двунаправленный, двунаправленный может быть подставлен вместо однонаправленного и т. д.

Применяя итераторы, важно учитывать такой элемент, как индикатор конца диапазона (end-of-range), т. е. элемент, идущий непосредственно за концом цепочки адресуемых итератором данных. Весьма похоже на схему адресации строк в языке Си++, когда признаком конца строки является символ '\0'. Но при работе с итераторами индикатором конца диапазона может быть любое число.

Алгоритмы

В библиотеке STL существует группа функций, выполняющих некоторые стандартные действия, например поиск, преобразование, сортировку, копирование и т. д. Они называются алгоритмами. Параметрами для алгоритмов, как правило, служат итераторы. Алгоритму нет никакого дела до типа переданного ему итератора. Главное, чтобы последний подпадал под определенную категорию. К примеру, если параметром алгоритма должен быть однонаправленный итератор, то подставляемый итератор должен быть либо однонаправленным, либо двунаправленным, или же итератором произвольного доступа.

Примером алгоритма может служить equal. Он сравнивает две цепочки данных, адресуемых входными итераторами, и описан следующим образом:

template <class InputIterator1, class InputIterator2>

bool equal(InputIterator1 first1, InputIterator1 last1, InputIterator2 first2);

Первый параметр - входной итератор, указывающий на первую цепочку сравниваемых данных. Второй адресует индикатор конца диапазона данных. Третий параметр - вторая цепочка сравниваемых данных. А вот фрагмент сравнения двух векторов (массивов) v1 и v2:

bool isEqual = equal(v1.begin(), v1.end(), v2.begin());

Здесь использованы стандартные методы векторов: begin() возвращает итератор, настроенный на начало цепочки данных, а end() возвращает индикатор выхода за диапазон. Если все элементы векторов попарно равны друг другу, то equal вернет значение "истина" (true).

Отметим, что все алгоритмы можно разделить на две основных категории: те, которые изменяют данные, и те, которые их не изменяют.

Контейнеры

Библиотека STL имеет в своем арсенале элементы, называемые контейнерами. Контейнеры - это объекты, хранящие в себе другие объекты. В STL таких контейнеров десять:

vector - массив с произвольным доступом, чаще всего применяемый в тех случаях, когда надо последовательно добавлять данные в конец цепочки;

list - похож на вектор, но эффективен при добавлении и удалении данных в любое место цепочки;

deque - контейнер, удобный для вставки данных в начало или конец;

set - набор уникальных элементов, отсортированных в определенном порядке;

multiset - то же, что и set, но может содержать повторяющиеся копии;

map - обеспечивает доступ к значениям по ключам;

multimap - то же, что и map, но допускающий повторяющиеся ключи;

stack - данные добавляются в одном порядке, а вынимаются в обратном;

queue - данные добавляются и вынимаются в том же порядке;

priority queue - то же, что и queue, но может сортировать данные по приоритету.

Надо отметить, что все алгоритмы работают с методами контейнеров, не вдаваясь в детали их реализации. Так, если алгоритму нужно определить, равен ли один элемент из контейнера другому, он просто вызывает перегруженный оператор сравнения "operator = = ()", реализованный в контейнере.

Функциональные объекты

Функциональные объекты - это объекты, у которых задан перегруженный оператор вызова функции "operator ()()". Они очень важны для эффективного использования. В тех местах, где предполагается передача указателей на функцию, создается интерфейс, принимающий объект с реализованным перегруженным оператором вызова функции. Все операторы обычно пишутся как inline, что дает дополнительный выигрыш в скорости.

Функциональными объектами являются все арифметические операторы: сложения (plus), вычитания (addition), умножения (times), деления (divides), взятия остатка (modulus) и обращения знака (negate). Имеются функциональные объекты для вычисления равенства (equal\_to), неравенства (not\_equal\_to), операции "больше" (greater), операции "меньше" (less), операции "больше или равно" (greater\_equal), операции "меньше или равно" (less\_equal). Для логических операторов имеются свои функциональные объекты: логическое "и" (logical\_and), логическое "или" (logical\_or) и логическое "не" (logical\_not).

В качестве примера приведена программа (см. листинг) с применением библиотеки STL, входящей в состав пакета Borland C++ Builder. Это маленький телефонный справочник.

Следует обратить внимание на то, что все включаемые файлы упоминаются без расширений .h и .hpp. Это новый стиль включения файлов, который, возможно, вскоре будет применяться во всех компиляторах языка Си++.

Листинг

// Используем контейнер map

# include <map>

// Основной включаемый файл STL

# include <algorithm>

// Используем контейнер vector

# include <vector>

# include <iostream.h>

# include <string>

// Эта строка обязательно должна быть,

// иначе может произойти конфликт имен

using namespace std;

// Объявляются два отображения

typedef map<string, long, less<string> > friendMap;

typedef map<long, string, less<long> > sortedMap;

// Создание имен для типов, хранимых в отображениях

// данных

typedef friendMap::value\_type entry\_type;

typedef sortedMap::value\_type sorted\_entry\_type;

// Вывести в поток запись

void printEntry(const entry\_type & entry)

{ cout << entry.first << ":" << entry.second << endl; }

void printSortedEntry(const sorted\_entry\_type & entry)

{ cout << entry.first << ":" << entry.second << endl; }

// Взять первые цифры телефона

int prefix(const entry\_type& entry)

{ return entry.second / 10000; }

// Сравнить первые цифры двух телефонов

bool prefixCompare(const entry\_type & a, const entry\_type & b)

{ return prefix(a) < prefix(b); }

// Функциональный объект, сравнивающий первые цифры телефонов

class checkPrefix {

public:

checkPrefix (int p) : testPrefix(p) { }

int testPrefix;

bool operator () (const entry\_type& entry)

{ return prefix(entry) == testPrefix; }

};

// Собственно справочник

class telephoneDirectory {

public:

// Добавить запись

void addEntry (string name, long number)

{ database[name] = number; }

// Удалить запись

void remove (string name)

{ database.erase(name); }

// Обновить запись

void update (string name, long number)

{ remove(name); addEntry(name, number); }

// Показать на экране все записи

void displayDatabase()

// for\_each перебирает все записи по очереди и вызывает

// метод printEntry, который показывает очередную запись

{ for\_each(database.begin(), database.end(), printEntry); }

void displayPrefix(int);

void displayByPrefix();

private:

// Здесь хранятся данные в виде контейнера set

friendMap database;

};

// Вывести все записи для заданных первых

// цифр телефона

void telephoneDirectory::displayPrefix(int prefix)

{

cout << "Listing for prefix " << prefix << endl;

// Создаем итератор, ссылающийся на

// контейнер с записями

map<string, long, less<string> >::iterator where;

// Искать с начала до конца все вхождения для заданных

// первых цифр телефона

where = find\_if(database.begin(), database.end(), checkPrefix(prefix));

while (where != database.end()) {

printEntry(\*where);

where = find\_if(++where, database.end(), checkPrefix(prefix));

}

cout << "end of prefix listing" << endl;

}

// Печатать в порядке телефонных номеров

void telephoneDirectory::displayByPrefix()

{

cout << "Display by prefix" << endl;

// Еще один контейнер set для хранения

// отсортированных записей

sortedMap sortedData;

// Заполнить контейнер, сортируя данные

for (friendMap::iterator i = database.begin(); i !=

database.end(); i++)

sortedData.insert(sortedMap::value\_type((\*i).second,

(\*i).first));

// Поочередно вывести отсортированные записи

for\_each(sortedData.begin(), sortedData.end(), printSortedEntry);

cout << "end display by prefix" << endl;

}

// Точка входа в программу

int main() {

cout << "Telephone Directory sample program" << endl;

telephoneDirectory friends;

// Добавление записей в справочник

friends.addEntry("Samantha", 6342343);

friends.addEntry("Brenda", 5436546);

friends.addEntry("Fred", 7435423);

friends.addEntry("Allen", 6348723);

// Вывести данные по порядку

friends.displayDatabase();

// Вывести телефоны, начинающиеся

// с цифры 634

friends.displayPrefix(634);

// Вывести данные, отсортировав их по номерам

friends.displayByPrefix();

cout << "End of telephone directory sample program" << endl;

return 0;

}