Библиотека стандартных шаблонов

STL

- Библиотека стандартных шаблонов (STL) (англ. Standard Template Library) — набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

- Библиотека стандартных шаблонов до включения в стандарт C++ была сторонней разработкой, вначале — фирмы HP(Hewlett-Packard), а затем SGI(Silicon Graphics, Inc.). Стандарт языка не называет её «STL», так как эта библиотека стала неотъемлемой частью языка, однако многие люди до сих пор используют это название, чтобы отличать её от остальной части стандартной библиотеки (потоки ввода-вывода (iostream), подраздел Си и др.).
- Архитектура STL была разработана Александром Степановым и Менг Ли.

Александр Степанов

СТРУКТУРА БИБЛИОТЕКИ

В библиотеке выделяют пять основных компонентов:

- Контейнер (англ. *container*) — хранение набора объектов в памяти.
- Итератор (англ. *iterator*) — обеспечение средств доступа к содержимому контейнера.
- Алгоритм (англ. *algorithm*) — определение вычислительной процедуры.
- Адаптер (англ. *adaptor*) — адаптация компонентов для обеспечения различного интерфейса.
- Функциональный объект (англ. *functor*) — сокрытие функции в объекте для использования другими компонентами.

КОНТЕЙНЕРЫ

Контейнеры библиотеки STL можно разделить на четыре категории:

- последовательные
- ассоциативные
- контейнеры-адаптеры
- псевдоконтейнеры

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ

vector

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

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ

list

- похож на вектор, но эффективен при добавлении и удалении данных в любое место цепочки.
  Поиск перебором медленнее, чем у вектора из-за большего времени доступа к элементу

ПОСЛЕДОВАТЕЛЬНЫЕ КОНТЕЙНЕРЫ

deque

- контейнер, удобный для вставки данных в начало или конец. Реализован в виде двусвязанного списка линейных массивов. С другой стороны, в отличие от vector, дек не гарантирует расположение всех своих элементов в непрерывном участке памяти, что делает невозможным безопасное использование арифметики указателей для доступа к элементам контейнера.

ПРИМЕР ПОСЛЕДОВАТЕЛЬНОГО КОНТЕЙНЕРА

```
#include <iostream>
#include <vector>
 #include <string>
 int main() {
 // Поддержка кириллицы в консоли Windows
     setlocale(LC_ALL, "");
// Создание вектора из строк
    std::vector<std::string>
     students;
// Буфер для ввода фамилии студента
    std::string buffer = "";
    std::cout << "Вводите фамилии студентов. " << "По окончание ввода введите пустую
                                                                строку" << std::endl;
   do {
        std::getline(std::cin, buffer);
        if (buffer.size() > 0) {
// Добавление элемента в конец вектора
 students.push_back(buffer);
  }}
   while (buffer != "");
// Сохраняем количество элементов вектора
   unsigned int vector_size = students.size();
// Вывод заполненного вектора на экран
std::cout << "Ваш вектор." << std::endl;
for (int i = 0; i < vector_size; i++) {
                        std::cout << students[i] << std::endl; }
return 0;
 }
```

АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ

set

- Упорядоченное множество уникальных элементов. При вставке/удалении элементов множества итераторы, указывающие на элементы этого множества, не становятся недействительными. Обеспечивает стандартные операции над множествами типа объединения, пересечения, вычитания. Тип элементов множества должен реализовывать оператор сравнения «<« или требуется предоставить функцию-компаратор. Реализован на основе самобалансирующего дерева двоичного поиска.

АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ

multiset

- То же что и set, но позволяет хранить повторяющиеся элементы.

АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ

map

- Упорядоченный ассоциативный массив пар элементов, состоящих из ключей и соответствующих им значений. Ключи должны быть уникальны. Порядок следования элементов определяется ключами. При этом тип ключа должен реализовывать оператор сравнения operator<, либо требуется предоставить функцию-компаратор.

АССОЦИАТИВНЫЕ КОНТЕЙНЕРЫ

multimap

- То же что и map, но позволяет хранить несколько одинаковых ключей.

ПРИМЕР АССОЦИАТИВНОГО КОНТЕЙНЕРА

```
#include "stdafx.h"
#include <iostream>
#include <map>
#include <string>
using namespace std;
 int main() {
map<string,int> m; //создаем контейнер
 //записываем данные в наш ассоциативный массив
m["s"]=5;
m["sr"]=52;
m["t"]=533;
map<string,int>:: iterator ii; // определяем итератор
for(ii=m.begin();ii!=m.end();ii++)cout<<ii->first<<":"<<ii->second<<endl;
 // к ключу можно обращаться еще вот так
//(*iter).first и (*iter).second соответственно
return 0;
 }
```

КОНТЕЙНЕРЫ-АДАПТЕРЫ

stack

- Стек — контейнер, в котором добавление и удаление элементов осуществляется с одного конца.

КОНТЕЙНЕРЫ-АДАПТЕРЫ

queue

- Очередь — контейнер, с одного конца которого можно добавлять элементы, а с другого — вынимать.

КОНТЕЙНЕРЫ-АДАПТЕРЫ

priority_queue

- Очередь с приоритетом, организованная так, что самый большой элемент всегда стоит на первом месте.

ПРИМЕР С КОНТЕЙНЕРАМИ-АДАПТЕРАМИ

```
#include <cstdlib>
#include <iostream>
 #include <string>
 #include <queue>
using namespace std;
int main() {
   queue<string> myqueue;
    string st,k,p,f;
    int n,r;
     cout<<"Enter size of queue: ";
     cin>>n;
     for(int count=1, i=0;i<n;i++,count++)
      { cout<<count<<". ";
        cin>>st;   //вписываем слова и кидаем их в очередь
        myqueue myqueue.push(st); }
        cout<<"Enter word which we must delete: ";
        cin>>f;  //пишем слово которое мы хотим удалить из очереди
       queue<string> newqueue;
       bool flag = false;
         cout<<"------------\n";
       while(!myqueue.empty())
        { k = myqueue.front();
          myqueue.pop();
        if(k==f && !flag) { flag = true; continue; }
          newqueue.push(k); }
          myqueue = newqueue;
          cout<<"-------------\n";
           system("PAUSE");
         return EXIT_SUCCESS;
}
```

ПСЕВДОКОНТЕЙНЕРЫ

bitset

- Служит для хранения битовых масок. Похож на vector<bool>фиксированного размера. Размер фиксируется тогда, когда объявляется объект bitset. Итераторов в bitset нет. Оптимизирован по размеру памяти.

ПСЕВДОКОНТЕЙНЕРЫ

basic_string

- Контейнер, предназначенный для хранения и обработки строк. Хранит в памяти элементы подряд единым блоком, что позволяет организовать быстрый доступ ко всей последовательности. Элементы должны быть POD'ами(Простыми структурами данных). Определена конкатенация с помощью +.

ПСЕВДОКОНТЕЙНЕРЫ

valarray

- Шаблон служит для хранения числовых массивов и оптимизирован для достижения повышенной вычислительной производительности. В некоторой степени похож на vector, но в нём отсутствует большинство стандартных для контейнеров операций. Определены операции над двумя valarray и над valarray и скаляром (поэлементные). Эти операции возможно эффективно реализовать как на векторных процессорах, так и на скалярных процессорах с блоками SIMD.

- **SIMD** (англ. *single instruction, multiple data* —**одиночный поток команд, множественный поток данных**, **ОКМД**) — принцип компьютерных вычислений, позволяющий обеспечить параллелизм на уровне данных.

ПРИМЕР С ПСЕВДОКОНТЕЙНЕРАМИ

```
#include <iostream>
#include <bitset>   // заголовочный файл битовых полей
#include <iomanip>  // для манипулятора setw()
using namespace std;

int main()
{
    bitset<8> number;

    cout << "Двоичное представление некоторых чисел:\n";
    for( int i = 0; i < 21; i++) {
        number = i;
        cout << setw(2) << number.to_ulong() << " = " << number << endl;
    }
    return 0;
}
```

КОНТЕЙНЕРЫ

- В контейнерах для хранения элементов используется семантика передачи объектов по значению. Другими словами, при добавлении контейнер получает копию элемента. Если создание копии нежелательно, то используют контейнер указателей на элементы. Присвоение элементов реализуется с помощью оператора присваивания, а их разрушение происходит с использованием деструктора. Сейчас мы увидим основные требования к элементам в контейнерах:

МЕТОДЫ

Конструктор копии	
Создаёт новый элемент, идентичный старому	Используется при каждой вставке элемента в контейнер

Оператор присваивания	
Заменяет содержимое элемента копией исходного элемента	Используется при каждой модификации элемента

Деструктор	
Разрушает элемент	Используется при каждом удалении элемента

МЕТОДЫ

Конструктор по умолчанию	
Создаёт элемент без аргументов	Применяется только для определённых операций

operator==	
Сравнивает два элемента	Используется при выполнении operator== для двух контейнеров

operator<	
Определяет, меньше ли один элемент другого	Используется при выполнении operator< для двух контейнеров

ИТЕРАТОРЫ

- В библиотеке STL для доступа к элементам в качестве посредника используется обобщённая абстракция, именуемая итератором. Каждый контейнер поддерживает «свой» вид итератора, который представляет собой «модернизированный» интеллектуальный указатель, «знающий» как получить доступ к элементам конкретного контейнера. Стандарт C++ определяет пять категорий итераторов:

КАТЕГОРИИ

Входные

operator++, operator*, operator->, конструктор копии, operator=, operator==, operator!=	Обеспечивают доступ для чтения в одном направлении. Позволяют выполнить присваивание или копирование с помощью оператора присваивания и конструктора копии.

Выходные

operator++, operator*, конструктор копии	Обеспечивают доступ для записи в одном направлении. Их нельзя сравнивать на равенство.

Однонаправленные

operator++, operator*, operator->, конструктор копии, конструктор по умолчанию, operator=, operator==, operator!=	Обеспечивают доступ для чтения и записи в одном направлении. Позволяют выполнить присваивание или копирование с помощью оператора присваиваивания и конструктора копии. Их можно сравнивать на равенство.

КАТЕГОРИИ

Двунаправленные

operator++, operator--, operator*, operator->, конструктор копии, конструктор по умолчанию, operator=, operator==, operator!=

Поддерживают все функции, описанные для однонаправленных итераторов (см. выше). Кроме того, они позволяют переходить к предыдущему элементу.

Произвольного доступа

operator++, operator--, operator*, operator->, конструктор копии, конструктор по умолчанию, operator=, operator==, operator!=, operator+, operator-, operator+=, operator-=, operator<, operator>, operator<=, operator>=, operator[]

Эквивалентны обычным указателям: поддерживают арифметику указателей, синтаксис индексации массивов и все формы сравнения.

СПАСИБО ЗА ВНИМАНИЕ

Выполнили:
Студенты 103 группы ФМиИТ
Полькин А.В. и Новиков Д.В.