Семинар 12
Введение в STL

Введение в STL

Standard Template Library – подмножество стандартной библиотеки Си++. Основное назначение – обеспечить программиста типовыми структурами данных и наиболее эффективными алгоритмами для обработки информации, представленной в этих структурах.

Структуры данных и алгоритмы в STL представлены в виде шаблонов.

Введение в STL

2 основные части STL:

- множество контейнеров
- набор обобщённых алгоритмов, позволяющих выполнять типовые операции над контейнерами

Введение в STL

Алгоритмы в STL не зависят от вида контейнера и типа элементов контейнера. Такая независимость достигается путём применения *итераторов*.

Введение в STL

Терминология.

Обобщённый алгоритм – алгоритм, пригодный для обработки множества контейнеров (независимо от вида контейнера и типа элементов).

Контейнер – представление коллекции (динамической структуры данных), в которую можно помещать объекты и из которой их можно получать обрато.

Итератор – звено связи алгоритма с контейнером.

Введение в STL

Самый простой пример контейнера – обычный массив. Контейнер в Си++ представляется объектом, способным включать другие объекты.

Назначение итератора – предоставить единый метод последовательного доступа к элементам контейнера. Также итератор обеспечивает возможность множественного доступа к контейнеру. Можно представить, что итератор – это указатель.

Введение в STL

При последовательном переборе элементов контейнера итератор не обязательно перемещается только по смежным (соседним в памяти) элементам. Но для алгоритма-клиента, который пользуется итератором, обращаясь к элементам контейнера, перебираемые элементы представляются в виде последовательности. У последовательности есть начало, с которого начинается обработка, и конец, не обрабатываемый алгоритмом.

Введение в STL

Для итератора должны быть определены операции:

*	получение значения элемента, на который в данный момент смотрит итератор
++	переход итератора к следующему элементу
==	сравнение позиций (не значений!) итераторов
!=	сравнение позиций итераторов на неравенство
=	присваивание итератору позиции в контейнере

Контейнеры STL

В STL определены контейнеры двух видов:
1) **последовательные**, в которых каждый элемент занимает конкретную позицию, не зависящую от значения элемента:
deque (дек – double-ended queue, двусторонняя очередь) – динамический массив, в который можно включать элементы и из которого можно брать элементы с обоих концов. Для применения в программу необходимо включить заголовок <deque>.

Контейнеры STL

list (двусвязный список) – каждый элемент списка содержит некоторое значение и два указателя (на предыдущий и последующий элементы). Заголовок: <list>

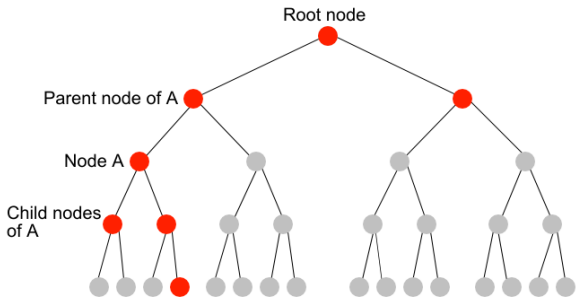
vector (вектор) – автоматически расширяющийся массив, обеспечивающий произвольный доступ к находящимся в нем элементам.

Контейнеры STL

Второй вид контейнеров STL:

2) **ассоциативные**, которые автоматически упорядочивают включаемые в них элементы по некоторому критерию, реализованному в виде функции, сравнивающей элементы.

Ассоциативные контейнеры реализуются в виде бинарных деревьев, где у каждого узла-элемента есть два узла-потомка.



Контейнеры STL

set (множество) – коллекция, в которой элементы уникальны и упорядочены по значениям. Заголовок: <set>

multiset (мультимножество) – множество, в котором допустимы элементы с одинаковыми значениями.

map (отображение) – коллекция, элементами которой являются пары «ключ»-«значение». Элементы упорядочены по значениям ключей. Все ключи должны быть уникальны. Заголовок: <map>

Контейнеры STL

multimap (мультиотображение) – отображение, в котором допустимы пары с одинаковыми ключами.

map и multimap могут быть использованы как ассоциативный массив.

Контейнеры STL

3) **специализированные** контейнеры (по стандарту относятся к последовательным, реализуются на базе рассмотренных конт.):
queue (очередь) – динамический массив, элементы которого можно включать в конец, а извлекать из начала (FIFO). Заголовок: <queue>.
stack (стек) – динамический массив, включение и извлечение элементов которого реализуется только с одного конца (LIFO). Заголовок: <stack>
priority_queue (очередь с приоритетом)

Контейнеры STL

bitset (битовое поле) – контейнер, в котором каждое логическое значение представлено одним битом. Предназначен для хранения битовых значений. При создании получает размер, который не изменяется при обработке. Заголовок: <bitset>.

Контейнеры STL

Небольшой пример применения контейнера stack.

```
#include "stdafx.h"
#include <iostream>
#include <stack>

int main(int argc, _TCHAR* argv[])
{
	std::stack<char> stackEx1;
	for(int i=int('0'); i<=int('Z'); i++){
		stackEx1.push(char(i)); // Занести элемент в стек
	}
	while(!stackEx1.empty()){ // Пока стек не пустой
		std::cout << stackEx1.top(); // Вывести верхний элемент из стека
		stackEx1.pop(); // Извлечь верхний элемент
	}
	return 0;
}
```

Основные методы контейнеров

1) Создание контейнеров

Создание пустого контейнера:

Тип_контейнера C;

Создание контейнера, содержащего N эл-в:

Тип_контейнера C(int N);

Создание копии (C1 и C2 – одного типа):

Тип_контейнера C1(C2);

Создание контейнера и инициализация элементами из интервала [beg, end):

Тип_контейнера C(beg, end);

Основные методы контейнеров

2) Уничтожение контейнеров

Деструктор:

C.~Тип_контейнера()

Основные методы контейнеров

3) Оценка числа элементов и размера

Функция для получения количества элементов контейнера:

C.size()

Проверка контейнера на пустоту (метод возвращает true, если контейнер пуст):

C.empty()

Функция для получения максимально возможного количества элементов контейнера:

C.max_size()

Основные методы контейнеров

Пример.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
#include <list>
using namespace std;

int main()
{
	int iArr[] = {1,2,3,4,5,6,7,8,9};
	vector <double> vect(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
	// iArr - адрес первого элемента массива
	// iArr + sizeof(iArr)/sizeof(iArr[0]) -
	// адрес первого элемента + размер массива
	cout << "vect.size()=" << vect.size() << endl; //9
	cout << "vect.max_size()=" << vect.max_size() << endl;
	cout << "vect.empty()=" << vect.empty() << endl; //Выведет 0, а это false
	list <float> lis(vect.begin(), vect.end()-2);
	cout << "lis.size()=" << lis.size() << endl; //7
	return 0;
}
```

Основные методы контейнеров

4) Сравнение контейнеров

Сравнение однотипных контейнеров выполняется с помощью стандартных операций отношений ==, !=, <, >, <=, >=

Два однотипных контейнера равны, если они поэлементно совпадают.

Основные методы контейнеров

4) Сравнение контейнеров (продолжение)

Отношения <, >, <=, >= проверяются лексикографически (как строки):

- если очередные два элемента не равны, результат сравнения этих элементов определяет результат сравнения контейнеров
- если количество элементов в контейнерах не равно, а те элементы, что есть, совпадают, контейнер с меньшим количеством элементов считается меньшим.

Основные методы контейнеров

Пример

```
int iArr[] = {1,2,3,4,5,6,7,8,9};
vector <double> vect1(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
vector <double> vect2(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]) - 3);
cout << "vect1 < vect2 is " << (vect1 < vect2 ? "true" : "false");
// false, т.к. в vect1 больше элементов, а остальные равны

cout << endl;
vector <double> vect3(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]) - 3);
vect3[1] = 9;
cout << "vect1 < vect3 is " << (vect1 < vect3 ? "true" : "false");
// true, т.к. первый элемент vect3 больше первого элемента
// vect1
```

Основные методы контейнеров

5) Присваивание контейнеров

Присваивание: все элементы из контейнера-источника копируются в контейнер-приёмник. Перед копированием все элементы контейнера-приёмника удаляются, память освобождается и выделяется для новых элементов. Присваивание – дорогостоящая операция.

Основные методы контейнеров

Пример.

```
int iArr[] = {1,2,3,4,5,6,7,8,9};
vector <double> vect1(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
vector <double> vect3(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]) - 3);
// Обратите внимание, что при инициализации в vect3 меньше
// элементов, чем в vect1
vect3 = vect1;
cout << "vect3.size()=" << vect3.size();
// Выведет: vect3.size()=9, т.к. в vect1 и vect3 теперь по 9 элем.
```

Основные методы контейнеров

6) Обмен содержимым контейнеров

Присваивание может быть заменено методом или функцией swap:

```
C1.swap(C2);
swap(C2, C1);
```

Основные методы контейнеров

7) Методы получения итераторов

C.begin() – итератор для 1-го элемента

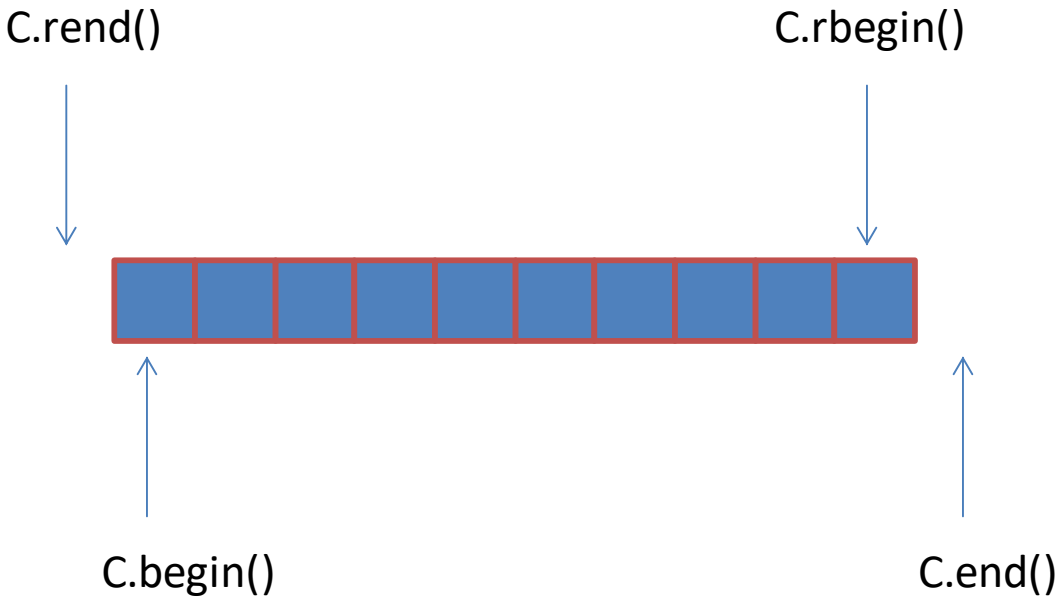
C.end() – итератор для позиции за последним элементом

C.rbegin() – обратный итератор для первого справа (последнего слева) элемента

C.rend() – обратный итератор для позиции за последним элементом при переборе в обратном порядке, т.е. для позиции, предшествующей первому элементу

Основные методы контейнеров

7) Методы получения итераторов



Основные методы контейнеров

Пример.

```
int iArr[] = {1,2,3,4,5,6,7,8,9};
vector <double> vect1(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
cout << "first element: " << *vect1.begin() << endl;
cout << "last element: " << *vect1.rbegin() << endl;
cout << "last element again: " << *(vect1.end() - 1) << endl;

// Обратите внимание, что итератор напоминает адрес,
который надо разыменовать для получения значения
```

Основные методы контейнеров

7) Методы вставки у удаления элементов

C.insert(pos, elem) – вставка элемента elem перед элементом с позицией pos (pos - итератор). Возвращает позицию нового элемента.

C.insert(pos, n, elem) – вставка n элементов elem.

C.insert(pos, beg, end) – вставка перед позицией pos копий элементов из диапазона [beg, end).

Основные методы контейнеров

Пример.

```
int iArr[] = {1,2,3,4,5,6,7,8,9};
vector <double> vect1(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
vect1.insert(vect1.begin(), 99);
cout << "first element: " << *vect1.begin() << endl;
```

Основные методы контейнеров

7) Методы вставки у удаления элементов

C.erase(pos) – удаление элемента из заданной позиции.

C.erase(beg, end) – удаление из контейнера элементов в диапазоне [beg, end).

C.clear() – удаление из контейнера всех элементов

Основные методы контейнеров

Пример

```
int iArr[] = {1,2,3,4,5,6,7,8,9};
vector <double> vect1(iArr, iArr + sizeof(iArr)/sizeof(iArr[0]));
vect1.erase(vect1.begin(), vect1.end());
cout << "vect1.size() = " << vect1.size();
```

Основные методы контейнеров

8) Методы вставки у удаления push и pop

C.push_back(elem) – вставка элемента в конец контейнера. Применим к деку, вектору, списку.

C.push_front(elem) – вставка элемента в начало контейнера. Применим к деку, списку.

C.pop_back(elem) – удаление последнего элемента контейнера. Применим к деку, вектору, списку.

C.pop_front(elem) – удаление первого элемента. Применим к деку, списку.

Основные методы контейнеров

9) Методы индексипрования

C[index] – обычное индексирование, применимо к вектору и деку.

C.at(index) – индексирование с проверкой допустимости обращения к элементу с данным индексом; применимо к вектору и деку; при выходе за границы диапазона формируется исключение out_of_range.

Итераторы STL

Каждый контейнер включает тип с названием iterator, т.о. итераторы можно создавать, как и обычные переменные

Для создания прямого итератора:

Тип_контейнера <тип> :: iterator имя_итер.;

Для создания обратного итератора:

Тип_контейнера <тип> :: reverse_iterator имя_итер.;

Итераторы STL

Пример.

```
list <double> list1;
list1.push_front(5.5);
list1.push_front(6.4);
list1.push_front(7.3);
list1.push_front(8.2);
//Output all elements:
list <double> :: iterator iter;
iter = list1.begin();
for(int i=0; i < list1.size(); i++)
    cout << i << "-th element = "<< *(iter++) << "\n";
```

Итераторы STL

Пример №2.

Итераторы разных типов могут выполнять разные действия.

```
deque <double> d1;
d1.push_front(5.5);
d1.push_front(6.4);
d1.push_front(7.3);
d1.push_front(8.2);
//Output all elements:
deque <double> :: iterator iter;
iter = d1.begin();
for(int i=0; i < d1.size(); i++)
      cout << i << "-th element = "<< *(iter+i) << "\n";
```

Итераторы STL

Для итераторов различных типов доступны различные операции. Пример:
Для итераторов контейнеров **vector** и **deque** доступны действия **++, --, +, -, +=, -=** и все операторы сравнения.
Для итераторов контейнеров **list, map, multimap, set, multiset** – действия **++, --** и операторы сравнения **==, !=**

Итераторы STL

Если iter – это итератор, то:

*** iter = x; - запись значения x в позицию, определяемую итератором**

x = *iter; - чтение в переменную x значения из позиции, определяемой итератором

Функторы

Функторы или объекты-функции или функциональные объекты – это объекты, к которым можно обратиться, используя синтаксис вызова функций:

Имя_объекта(список_аргументов)

Т.к. функтор является объектом, то он может сохранять некоторое состояние.

Функторы

В функторах нужно переопределять операцию () (круглые скобки).

Функторы часто используются в сочетании с алгоритмами STL.

Одна из функций STL для генерации последовательностей –
generate(first_el, last_el, functor)
Функтор вызывается заново для каждого эл-та.

Функторы

Пример.

```
class functorFib{
    int one, two;
public:
    functorFib(int x=1, int y=1) :one(x), two(y){}
    int operator()(void){
        int z;
        z = one+two;
        one=two;
        two=z;
        return z;
    }
};
```

Функторы

Пример.

```
void main()
{
	functorFib fib(2, 3);
	vector <int> vect1(5);
	vector <int> vect2(6);
	generate(vect1.begin(), vect1.end(), functorFib(-1, -2));
	generate(vect2.begin(), vect2.end(), fib);
	// не забудьте #include <algorithm>
	for(int i=0; i<5; i++){
		cout << vect1.at(i) << '\t';
	}
	cout << endl;
	for(int i=0; i<6; i++){
		cout << vect2.at(i) << '\t';
	}
}
```

Функторы

Ещё один пример.

```
#include "stdafx.h"
#include <iostream>
#include <vector>
using namespace std;
#include <algorithm>

class f{
        int number;
public:
        f(int x=1) :number(x) {}
        int operator()(void){
                int z = number;
                number++;
                return z;
        }
};

int main()
{
        vector <int> vect1(6);
        generate(vect1.begin(),  vect1.end(), f(10));
        for(int i=0; i<6; i++) {cout << vect1.at(i) << '\t';}
        return 0;
}
```

Алгоритмы STL

Заголовки: <algorithm>, <numeric>, <cstdlib>

Функции сравнения и обмена:

min(),
max(),
swap(),
max_element(),
min_element()

Алгоритмы STL

Пример применения:

```
vector<int> v(6);
generate(v.begin(), v.end(), functorFib(-1,-2));
cout << *min_element(v.begin(), v.end()) << endl;
// Вывод: -34
```

Алгоритмы STL

Пример применения:

```
bool myMax(int a, int b){
return (a<b);
}

void main()
{
cout << max(88, 77, myMax) << endl;
// возвращается значение b, если a<b
}
```

Алгоритмы STL

Присваивание значений n элементам

generate_n(first, n, functor)

Пример.

```
void main()
{
    int a[] = {11, 22, 33, 44, 55, 66};
    vector <int> v(5);
    generate_n(v.begin(), 5, functorFib(5,3));
    for(int i=0; i<v.size(); i++)
        cout << v[i] << '\t';
}
```

Алгоритмы STL

Сканирование последовательности с возможным изменением значения

for_each(first, last, func)

Функция for_each передаёт в функцию func очередное значение итератора. Функция func обрабатывает это значение (по ссылке).

Алгоритмы STL

```
void plus2(int &arg){
    arg+=2;
}

void main()
{
    int a[] = {11, 22, 33, 44, 55, 66};
    vector <int> v(a, a+sizeof(a)/sizeof(a[0]));
    for_each(v.begin(), v.end(), plus2);
    for(int i=0; i<v.size(); i++)
    cout << v[i] << '\t';
}
```

Алгоритмы STL

Замена/удаление значений последовательности по условию

replace_if(first, last, predicate, value)
remove_if(first, last, predicate)

Алгоритмы STL

Пример. Заменить единицами все элементы, равные 11

```
template <int I, typename C>
bool isequalto(C val){
	if (val==I) return true;
	return false;
}

void main()
{
	int a[] = {11, 22, 11, 44, 11, 66};
	vector <int> v(a, a+sizeof(a)/sizeof(a[0]));
	replace_if(v.begin(), v.end(), isequalto<11, int>, 1);
	for(int i=0; i<v.size(); i++)
	cout << v[i] << '\t';
}
```

Алгоритмы STL

Изменение порядка элементов

random_shuffle(first, last) – изменяет порядок элементов с использованием равномерного закона случайного распределения
random_shuffle(first, last, func) – позволяет задать свою функцию распределения

Алгоритмы STL

Пример.

```
void main()
{
    int a[] = {33, -10, 2, 4, 1, -100};
    vector <int> v(a, a+sizeof(a)/sizeof(a[0]));
    random_shuffle(v.begin(), v.end());
    for(int i=0; i<v.size(); i++)
        cout << v[i] << '\t';
}
```

Алгоритмы STL

Формирование нового порядка размещения элементов

sort()
partial_sort()
nth_element(first, nth, last); - элементы перемешиваются таким образом, что все элементы слева меньше *nth, а элементы справа – больше *nth.

Алгоритмы STL

Пример.

```
bool myfunction (int i, int j) { return (i>j); }
int main(){
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);
  sort (myvector.begin(), myvector.begin()+4,
myfunction);
  for(int i=0; i<8; i++){
    cout << myvector.at(i) << '\t';
  }
}
```

Алгоритмы STL

Объединение двух последовательностей (функция merge())

```
int main()
{
	vector <int> vect1(6);
	generate(vect1.begin(), vect1.end(), f(10));
	int i[] = {1, 2, 3, 4, 5};
	int len = 5;
	list <int> list1 (11);
	merge(&i[0], &i[len], vect1.begin(), vect1.end(), list1.begin());
	while(!list1.empty()){
		cout << list1.front() << '\t';
		list1.pop_front();
	}
	return 0;
}
```

Алгоритмы STL

Примеры по алгоритмам STL:
http://cppe.ru/index.php/C++
http://www.cplusplus.com/reference/algorithm/