Контейры в C++

Контейнеры

Шаблоны классов и алгоритмов, позволяющих программистам легко реализовывать общие структуры данных, такие как

- *очереди*
- *списки*
- *стеки*

Существует три вида контейнеров: *последовательные контейнеры, ассоциативные контейнеры, и неупорядоченные ассоциативные контейнеры*, каждый из которых предназначен для поддержки различных наборов операций.

Последовательные		Ассоциативные	
array (начиная с C++11)	статический непрерывный массив	**set**	Коллекция уникальных ключей, отсортированная по ключам
vector	динамический непрерывный массив	**map**	Коллекция пар ключ-значение, отсортированная по ключам, ключи являются уникальными
deque	Двусторонняя очередь		
forward_list (начиная с C++11)	односвязный список	**multiset**	Коллекция ключей, отсортированная по ключам
list	двусвязный список	**multimap**	Коллекция пар ключ-значение, отсортированная по ключам

Вектор

Контейнер ***VECTOR*** ведет себя как массив, но может автоматически увеличиваться по мере необходимости.

Он поддерживает прямой доступ и связанное хранение и имеет очень гибкую длину. По этим и многим другим причинам контейнер *vector* является наиболее предпочтительным последовательным контейнером для большинства областей применения.

Для добавления нового элемента в конец вектора используется метод push_back(). Количество элементов определяется методом size(). Для доступа к элементам вектора можно использовать квадратные скобки [], также, как и для обычных массивов.

- pop_back() — удалить последний элемент
- clear() — удалить все элементы вектора
- empty() — проверить вектор на пустоту

Программная реализация контейнеров

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector1(10); // вывод элементов вектора
    cout << "Входной массив: ";
    for(int i = 0; i < myVector1.size(); i++) {
        myVector1[i] = i;
        cout << myVector1[i] << ' ';
    }
    cout << "\nСкопированный массив: ";
    vector<int>  myVector2(myVector1);  // при  объявлении
второго вектора, копируется - первый
    for(int i = 0; i < myVector2.size(); i++) {
        myVector2[i] = i;
        cout << myVector2[i] << ' ';
    }
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
        vector<int> myVector(10,1);
         // объявляем вектор размером в 10
              // и заполняем его единицей, потом изменяем на
номер элемента
        for(int i = 0; i < myVector.size(); i++)
                myVector[i]=i;
         // вывод элементов вектора на экран
        for(int i = 0; i < myVector.size(); i++)
                cout << myVector[i] << ' ';
        return 0;
}
```

Итератор

Это объект, который позволяет перемещаться по элементам некоторой последовательности.

В отличие от разнообразных последовательностей элементов (массивы, списки, файлы), итераторы имеют одинаковый интерфейс: получение текущего элемента, перемещение к следующему. Это позволяет писать более общие алгоритмы, которые работают с любыми итераторами, поддерживающими этот минимальный набор функций.

Работа с итератором

Так же итератор можно использовать для быстрой сортировки массивов вида:

sort (array1.begin(), array1.end());

Или перестановки элементов:

swap (it1,it2);

И других функций, находящихся в <algorithm>

Программная реализация итератора

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
	int data[10] = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 };
	for (int i = 0;i < 10;i++)
		cout << data[i] << ' ';
	cout << endl;
	reverse(data + 2, data + 6);
	// интервал { 5, 7, 9, 11 } переходит в { 11, 9, 7, 5 }
	for (int i = 0;i < 10;i++)
		cout << data[i] << ' ';
	return 0;
}
```

```
template<class Type>class masi
{
        Type mas[MAXSIZE];
        int size;
public:
        class iterator
        {
                Type *current;
                public:
                iterator() { current = 0; }
                void operator+=(int temp) { current += temp; }
                void operator-=(int temp) { current -= temp; }
                void operator=(Type& temp) { current = &temp; }
                Type operator *() { return *current; }
                Type* operator ->() { return current; }
        };
           masi() {       size = 0;}
         void add(int temp){size++; mas[size - 1] = temp; }
         void del(){ size--; mas[size + 1] = 0; }
        void show()
        {
         cout << "Массив:\n";
         for (int i = 0;i < size;i++)
         cout << mas[i] << ' ';
         cout << endl;
        }
         Type& begin() { return mas[0]; }
         Type& end() { return mas[size]; }
};
```

```
#include <iostream>
using namespace std;
#define MAXSIZE 100

int main()
{
masi<int> a;
for (int i = 0;i < 5;i++)
a.add(i);
masi<int>::iterator it;
it = a.begin();
for (int i = 0;i < 5;i++)
{
cout << *it << ' ';
it+=1;
}
return 0;
}
```

Работа с контейнерами

Все контейнеры имеют похожий интерфейс функций, за исключением отсутствия некоторых функций, связанные в связи с различными методами связи между элементами контейнера.

Так же это ускоряет процесс изучения всех остальных контейнеров.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
	vector<int> array1; // создаем пустой вектор
	// добавляем в конец вектора array1 элементы 4, 3, 1
	array1.insert(array1.end(), 4);
	array1.insert(array1.end(), 3);
	array1.insert(array1.end(), 1);
	// вывод на экран элементов вектора через итератор
	vector<int>::iterator it; //объявление итератора
	it = array1.begin();
	for (int i = 0;i < array1.size();i++)
	{
		cout << *it;
		it++;
	}
	cout << endl;
	sort(array1.begin(), array1.end()); //сортировка вектора
	it = array1.begin();
	for (int i = 0;i < array1.size();i++)
	{
		cout << *it << " ";
		it++;
	}
	return 0;
}
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
vector<int> array1(3);
// инициализируем элементы вектора array1
array1[0] = 4;
array1[1] = 2;
array1[2] = 1;
vector<int> array2(3);
// инициализируем элементы вектора array2
array2[0] = 4;
array2[1] = 2;
array2[2] = 1;
// сравниваем массивы
if (array1 == array2) {
cout << "array1 == array2" << endl;
}
return 0;
}
```