Глава 11. **Библиотека стандартных контейнерных классов**

МГТУ им. Н.Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети
Лектор: д.т.н., проф.
Иванова Галина Сергеевна

Стандартные контейнерные классы STD

- **последовательные** - обеспечивают хранение конечного количества однотипных величин в виде непрерывной последовательности;

- **ассоциативные** контейнеры обеспечивают быстрый доступ к данным по ключу, построены на основе сбалансированных деревьев.

11.1 Последовательные контейнеры

- **`vector`** – динамический массив — структура, эффективно реализующая произвольный доступ к элементам, добавление в конец и удаление из конца;
- **`deque`** – двусторонняя очередь (дек) – эффективно реализует произвольный доступ к элементам, добавление в оба конца и удаление из обоих концов;
- **`list`** – линейный список – эффективно реализует вставку и удаление элементов в произвольное место, но не имеет произвольного доступа к своим элементам;
- stack – стек;
- queue – очередь;
- priority_queue – очередь с приоритетами.

Примеры вызова конструкторов вектора

```
а) vector<one> m1 (10);  /* вектор из 10 объектов класса one
                            (работает конструктор one без параметров) */

б) vector<int> v1 (10, 1);// вектор из 10 единичных элементов

в) vector<int> v2 (v1);  // вектор, равный вектору v1
```

Основные типы, используемые в шаблонах

`value_type` - тип элемента контейнера;

`size_type` - тип индексов элементов и т. д.;

`iterator` - тип "итератор" – указатель на элемент;

`const_iterator` - тип "константный итератор" – используется для неизменных данных;

`reverse_iterator` - обратный итератор – для просмотра от конца к началу;

`const_reverse_iterator` - константный обратный итератор;

`key_type` - тип ключа (для ассоциативных контейнеров);

`key_compare` - тип результата сравнения ключей.

`reference` - ссылка на элемент;

`const_reference` - константная ссылка на элемент;

Итераторы

Работа с элементами контейнеров ведется через итераторы и ссылки.

Итератор – объект, обладающий свойствами указателя.

Объявление итератора:

```
vector<double> ::iterator i;
```

Доступ к элементу через итератор:

```
*i
```

Перемещение итератора:

```
++i, i++
```

Операции над итераторами

Категория итератора	Допустимые операции	Контей-неры
Входной (input) – перемещается вперед и допускает только чтение элементов	`x=*i,` `++i,i++`	все
Выходной (output) – перемещается вперед и допускает запись элементов	`*i = x,` `++i, i++`	все

Прямой (forward) – допускает запись и чтение элементов при движении вперед.
Реверсивный (reverse) – допускает запись и чтение элементов при движении назад.
	`x =*i, *i = x,`
	`++i, i++`
	`x =*i, *i = x,`
	`--i, i--`
		все

Двунаправленный (bidirectional) – работает вперед и назад	`x=*i,` `++i,i++,--i,i--`	все

Произвольного доступа (random access)	`x = *i, *i = x,`
	`--i, i--`
	`i + n, i – n,`
	`i += n, i -= n`
	`i < j, i > j,`
	`i <= j, i >= j`
		все, кроме list

Методы получения адресов для инициализации итераторов

Указатель на первый:

```
iterator begin()
const_iterator begin() const
```

Указатель на следующий за последним:

```
iterator end()
const_iterator end() const
```

Указатель на первый при обратном просмотре:

```
reverse_iterator rbegin()
const_reverse_iterator rbegin() const
```

Указатель на элемент, следующий за последним, при обратном просмотре:

```
reverse_iterator rend()
const_reverse_iterator rend() const
```

Операции последовательных контейнеров

Операция	Метод	vector	deque	list
Вставка в начало	`push_front()`	-	+	+
Удаление из начала	`pop_front()`	-	+	+
Вставка в конец	`push_back()`	+	+	+
Удаление из конца	`pop_back()`	+	+	+
Вставка в произвольное место	`insert()`	+	+	+
Удаление из произвольного места	`erase()`	+	+	+
Произвольный доступ	`[ ], at`	+	+	-

Пример использования шаблона Динамический массив (Ex11.1)

```
#include <fstream>
#include <iostream>
#include <vector>
using namespace std;
int main()
{  ifstream in ("inp.txt");
   vector<int> v;  // инстанцирование шаблона Динамический
                                                  // массив
   int x;
   while ( in >> x )
       v.push_back(x); // добавление элемента в конец
   for (vector<int>::iterator i = v.begin();
                                          i != v.end(); ++i)
       cout << *i << " ";
    return 0;
 }
```

11.2 Ассоциативные контейнеры

- map – словарь уникальных ключей,
- multimap – словарь ключей с дубликатами,
- set - множество,
- multiset – мультиножество,
- bitset – битовое множество (набор битов).

Словари часто называют также *ассоциативными массивами* или *отображениями*.

Словарь построен на основе пар значений, первое из которых представляет собой ключ для идентификации элемента, а второе — собственно элемент.

Компонентная функция **`find()`** ищет элемент контейнера по заданному ключу.

Пример работы с ассоциативным контейнером

```
#include <fstream>
#include <iostream>
#include <string>
#include <map>
using namespace std;
typedef map <string, long> map_sl;
int main()
{  map_sl m;     string str;   long num;
   ifstream in("text.txt");
   while (in >> num)
   {
      in.get(); getline(in, str);
      m[str] = num;
      cout << str << "   " << num << endl;
   }
```

```
Petr  122
Nicolay  156
Vadim  354
```

Работа со словарем

```
// дополнение словаря
m["Martin"] = 213;

// вывод словаря
map_sl :: iterator i;
cout << "m:" << endl;
for (i = m.begin(); i != m.end(); i++)
   cout << (*i).first << "  " << (*i).second << endl;

// вывод второй пары
i = m.begin(); i++;
cout << "Second: ";
cout << (*i).first << "  " << (*i).second << endl;
```

```
m:
Martin  213
Nicolay  156
Petr  122
Vadim  354
Second: Nicolay  156
```

Поиск элемента по ключу

```
   // Вывод элемента по ключу

   cout << "Name: " << endl;
   getline(cin, str); // ввод ключа
   if (m.find(str) != m.end())   // поиск значения по ключу
      cout << m[str]<<endl;
   else
      cout << "Error" ;
   return 0;
}
```

```
Name:
Vadim
354
Для продолжения нажмите любую клавишу . . .
```