Лекция 16.

Библиотека STL. Отсортированные ассоциативные контейнеры.

Компоненты STL – множества.

Обеспечивают возможность быстрой выборки объектов из коллекции на основе значения ключа. Размер коллекции изменяется динамически.

- **`set<Key>`**

Множество - упорядоченная коллекция уникальных элементов (ключей). Обеспечивает быструю выборку искомого ключа.

- **`multiset<Key>`**

Мультимножество – аналогично множеству, но позволяет хранить повторяющиеся (дублированные) элементы.

Множество set

Контейнер set используется для хранения и извлечения данных из коллекции, в которой значения элементов уникальны. Элементы служат в качестве ключей, в соответствии с которым данные автоматически упорядочиваются.

Значение элемента не может быть изменено напрямую. Вместо этого используют удаление старого элемента и вставка нового.

Синтаксис определения set

```
template <class Key,
  class Traits=less<Key>,
  class Allocator=allocator<Key>>
class set
```

Параметры

Key тип элемента данных в множестве

Traits тип функционального объекта, который используется для сравнения элементов (по умолчанию less<Key>)

Allocator тип аллокатора объекта для размещения множества в памяти (по умолчанию allocator<*Key*>).

Пример 1

```
#include <set>
#include <iostream>
using namespace std;

int main()
{
 set<int> s1;
 set<int>::size_type i;
 s1.insert(1);
 s1.insert(1); // Ключи должны быть уникальными,
         // поэтому дубликаты игнорируются
 i = s1.count(1);
 cout << “Число элементов 1: " << i << endl;
 i = s1.count(2);
 cout << «Число элементов 2: " << i << endl;
}
```

Основные методы класса set

Емкость множества

`empty`	проверка на пустоту
`size`	количество элементов
`max_size`	максимальное число элементов

Модификаторы

`clear`	очистка содержимого
`insert`	вставка элемента
`erase`	удаление элемента

Просмотр содержимого

`count`	количество элементов, соотв. ключу
`find`	поиск элемента по заданному ключу

Пример 2 (начало)

```
#include <set>
using namespace std;

template <typename T> void print_element(const T& t)
{
    cout << "(" << t << ") ";
}

template <typename T> void print_collection(const T& t)
{
    cout << t.size() << " elements: ";

    for (const auto& p : t)
    {
        print_element(p);
    }
    cout << endl;
}
```

Пример 2 (продолжение)

```
template <typename C, class T>
void findit(const C& c, T val)
{
    cout << "Поиск элемента " << val << endl;
    auto result = c.find(val);
    if (result != c.end())
    {
        cout << "Элемент найден: ";
        print_element(*result);
        cout << endl;
    }
    else
    {
        cout << "Элемент не найден." << endl;
    }
}
```

Пример 2 (окончание)

```
int main()
{
    set<int> s({ 40, 45 });
    cout << "Исходное множество: " << endl;
    print_collection(s);

    s.insert(43);
    s.insert(41);
    s.insert(42);
    s.insert(44);
    s.insert(44);   // попытка дублирования

    cout << "Модифицированное множество: " << endl;
    print_collection(s);
    findit(s1, 45);
    findit(s1, 6);
}
```

Мультимножество (multiset)

Во многом аналогично множеству, но допускает хранение одинаковых элементов (дубликатов).

Синтаксис определения шаблона multiset совпадает с синтаксисом определения set.

Пример 3 (multiset)

```
#include <set>
#include <iostream>
using namespace std;

int main()
{
    multiset<int> ms;

    ms.insert(1);
    ms.insert(1);
    ms.insert(2);

    cout << "Элементов 1: " << ms.count(1) << endl;
    cout << "Элементов 2: " << ms.count(2) << endl;
    cout << "Элементов 3: " << ms.count(3) << endl;
}
```

Использование множеств и мультимножеств

Множества часто используются в ситуациях, когда необходимо быстро определить **наличие** заданного элемента в наборе (есть ключ или нет?).

Мультимножества используются в ситуациях, когда нужно найти **количество заданных элементов** в большом наборе (сколько ключей?).

Задачи:

1) определить количество различающихся слов в тексте (set)
2) определить частоту появления различных слов в тексте (multiset)

Компоненты STL – ассоциативные массивы

- **`map<Key, T>`**

Упорядоченный ассоциативный массив пар элементов, состоящих из ключа (типа Key) и соответствующего ему значения (типа T). Ключи должны быть уникальны. Обеспечивает быструю выборку элемента типа T на основе заданного ключа.

- **`multimap<Key, T>`**

Множественный упорядоченный ассоциативный массив. Аналогичен map, но позволяет хранить дублированные ключи.

Синтаксис определения map

```
template <class Key,
  class Type,
  class Traits=less<Key>,
  class Allocator=allocator< pair<const Key, type>>>
class map
```

Параметры

Key тип ключа в множестве

Type тип элемента данных в множестве

Traits тип функционального объекта, который используется для сравнения ключей
(по умолчанию less<Key>)

Allocator тип аллокатора для пары ключ/значение.

Пример использования map

Задача:
создать программу – телефонный справочник.

Данные в справочнике должны хранится в виде пар «имя абонента» - «номер телефона». Программа должна обеспечивать быструю выборку телефонного номера по имени абонента.

Имя абонента и его номер может храниться в переменных типа `string` и `long`. Так как поиск будет вестись по имени, то имя в этой паре является ключем, а номер телефона – значением элемента. Используем контейнер `map<string, long>` для хранения телефонного справочника.

Пример 4 (map) – начало

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
  map<string, long> directory;
  directory["Marina Semenova"] = 1234567;
  directory["Vasiliy Zaitsev"] = 9876543;
  directory["Andrey Petukhov"] = 3459876;
  ...
```

Пример 4 (map) - окончание

```
   ... // поиск номеров телефонов
   string name;
   while (cin >> name)
   {
      if(directory.find(name) !=
         directory.end())
         cout << «Телефон " << name << " - "
              << directory[name] << "\n";
      else
         cout << "Извините, в списке нет "
              << name << "\n";
   return 0;
}
```

Пример использования multimap

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

void main()
{
   multimap<string,int> mm = {{ "Mother", 37 },
                              { "Father", 40 },
                              { "Brother", 15},
                              { "Brother", 17},
                              { "Sister", 20 }};
   //вывод на экран
   for(auto i = mm.begin(); i != mm.end(); ++i)
   {
      cout << i->first << ":" << i->second << endl;
   }
}
```