Множества и мультимножества

В STL есть замечательный контейнер — set, он реализует такие сущности как множество и мультимножество. По сути это контейнеры, которые содержат некоторое количество отсортированных элементов. Да, именно так, при добавлении нового элемента в множество он сразу становится на свое место так, чтобы не нарушать порядка сортировки. Потому как в множестве и мультимножестве все элементы сортируются автоматически. Но вот вопрос, в чем же разница между множествами и мультимножествами? Множества содержат только уникальные элементы, а мультимножества могут содержать дубликаты, вот такая вот небольшая разница.

Для того, чтобы использовать множество или мультимножество необходимо подключить следующий заголовочный файл:

|  |  |
| --- | --- |
| 1 | #include <set> |

Давайте рассмотрим простой пример использования множества в программе:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  #include <set>  // заголовочный файл множеств и мультимножеств  #include <iterator>    using namespace std;  int main()  {      set<char> mySet; // объявили пустое множество        // добавляем элементы в множество      mySet.insert('I');      mySet.insert('n');      mySet.insert('f');      mySet.insert('i');      mySet.insert('n');      mySet.insert('i');      mySet.insert('t');      mySet.insert('y');        copy( mySet.begin(), mySet.end(), ostream\_iterator<char>(cout, " "));      return 0;  } |

Чтобы объявить множество, необходимо подключить заголовочный файл set, это сделано в **строке 2**. Для объявления множества необходимо воспользоваться классом set, **строка 8**. То есть в восьмой строке, мы создали объект — множество с именем mySet, элементы которого имеют тип данных char. **Строки 11-18** добавляют в множество новые элементы, до этого, множество было пустое. Чтобы добавить элемент в множество, достаточно воспользоваться методом insert(), которому в параметре передать новый элемент. Ну и в **строке 20** как обычно выполняется вывод множества на экран, с помощью функции copy(). Смотрим результат работы программы: 

А теперь давайте по порядку будем разбираться, почему же у нас получился такой вывод программы. Во-первых, обратите внимание на порядок добавления элементов в коде, **строки 11-18**, и на реальный порядок расположения этих элементов в множестве. Порядок ввода и реальный порядок элементов в множестве — разный, это связано с тем, что элементы множества автоматически сортируются. Еще одной очень важной деталью является то, что в множество не сохранились дубликаты, хотя дубликаты были при добавлении элементов в множество. Как видно в выводе программы, каждый элемент множества уникален.

Сра зу же хочу обратить ваше внимание на то, что это программа была простым примером использования МНОЖЕСТВА. Давайте немного переделаем эту программу:

|  |  |
| --- | --- |
| 1 | multiset<char> mySet; // объявили пустое множество |

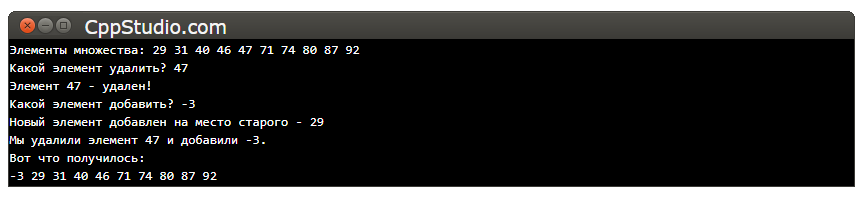
Я изменил всего одну строку, чтобы показать пример с мультимножеством. Как вы уже могли догадаться, достаточно было вместо класса set воспользоваться классом multiset — класс мультимножества. Весь остальной код не меняем и смотрим на результат работы программы:

Результат другой, так как мультимножество может хранить дубликаты элементов, все введенный буквы — сохранились. Ну и конечно все элементы отсортировались, как и в множестве. Из всего этого стоит запомнить то, что порядок ввода элементов в множество никак не влияет на порядок хранения в множестве. А также, мультимножество умеет хранить дубликаты, в отличие от множества. Если вам нужно объявить множество ,используйте класс set, если же вы хотите объявить мультимножество — воспользуйтесь классом multiset.

Автоматическая сортировка элементов в множествах накладывает определенные ограничения. Например, в множествах нельзя изменить значение какого-то элемента напрямую, так как это могло бы сломать сортировку. Поэтому, если вам сильно нужно это сделать, вы сначала должны удалить старый элемент, а потом добавить новый. Рассмотрим пример программы с удалением и добавлением нового элемента:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | #include <iostream>    #include <set>  // заголовочный файл множест и мультимножеств  #include <iterator>  #include <cstdlib>    using namespace std;  int main()  {      srand(time(NULL));      set<int> mySet; // объявили пустое множество        // добавляем элементы в множество      for( int i = 0; i < 10; i++) {          mySet.insert( rand() % 100 );      }        cout << "Элементы множества: ";      copy( mySet.begin(), mySet.end(), ostream\_iterator<int>(cout, " "));        int del = 0;      cout << "\nКакой элемент удалить? ";      cin >> del;        cout <<"Элемент " << \*mySet.find(del) << " - удален!" << endl;      mySet.erase(del);        int add = 0;      cout << "Какой элемент добавить? ";      cin >> add;        cout << "Новый элемент добавлен на место старого - " << \*mySet.lower\_bound(add) << endl;      mySet.insert(add);        cout << "Мы удалили элемент " << del << " и добавили " << add << ".\nВот что получилось: " << endl;      copy( mySet.begin(), mySet.end(), ostream\_iterator<int>(cout, " "));        return 0;  } |

В этом примере мы сначала удаляем элемент множества, это делается в **строке 26**, с помощью метода erase(). Потом мы добавляем новый элемент, **строка 33**. Кроме этого, обратите внимание на метод — find(), **строка 25**, он возвращает указатель на первый элемент множества, который равен его аргументу. Еще один интересный метод, которым мы воспользовались в программе называется — lower\_bound(), **строка 32**. Метод lower\_bound() возвращает указатель на первый элемент множества, значение которого больше либо равно аргументу. Вот вывод программы:



Все эти операции, в программах выше, характерны и для мультимножеств, поэтому отдельные примеры для мультимножеств не вижу смысла показывать. На этом пока все, это была вводная статья по множествам, надеюсь все было понятно.