Шаблонный класс set

Обсудим шаблонный класс set, имеющийся в стандартной библиотеке (соответствующий заголовочный файл так и называется set).

Множество – это структура данных, позволяющая хранить множества в математическом смысле, т. е. набор элементов без повторений. Отличие от математического понятия множества состоит в том, что все элементы множества должны принадлежать одному и тому же типу.

Контейнер set реализован при помощи сбалансированного двоичного дерева поиска, поэтому элементы в нем хранятся так, чтобы их последовательный перебор всегда происходил в порядке возрастания.

Шаблонный класс set определяет класс множества из элементов, тип которых задается шаблонным параметром. Единственное, что требуется от класса элементов – чтобы на нем были перегружены операции сравнения, причем то отношение порядка, которое они определяют, должно быть линейным, т. е. из любых двух разных элементов всегда один должен быть меньше другого. При этом смысл данного отношения порядка вообще не важен, лишь бы порядок был линейным.

Можно построить множество из пользовательского, структурного типа, лишь бы для него были перегружены операции сравнения.

Например, множество целых чисел можно определить, написав

set<int> s;

или множество строк:

set<string> s;

В обоих случаях при определении множества вызывается конструктор по умолчанию, который создает пустое множество.

Если мы хотим создать копию какого-либо множества, то воспользуемся конструктором копирования, который имеется в классе множества:

set<char> t (v); // множество t – копия множества v

Основные операции над множествами:

- добавление элемента в множество
- удаление элемента из множества
- проверка принадлежности элемента множеству

Метод класса set	Действие метода
insert	добавление элемента в множество (если добавляемый элемент уже присутствует в множестве, ничего не происходит)

erase	удаление элемента из множества (если такого элемента нет, ничего не происходит)
count	проверка принадлежности элемента множеству, единственный параметр метода – проверяемый элемент; значение 0 означает, что такого элемента в множестве нет, 1 – такой элемент есть.

Примеры.

```
set <string> s;
// конструктор по умолчанию создает пустое множество
s.insert("January");
s.insert("February");
s.insert("March");

// здесь множество состоит из трех элементов
```

Предположим, мы вставили элемент `March` в множество дважды:

```
set<string> s;
s.insert("January");
s.insert("March");
s.insert("February");
s.insert("March");
```

Множество s по-прежнему будет состоять из тех же трех элементов, поскольку при добавлении одинаковых значений множество в C++ будет хранить только один его экземпляр (все элементы множества уникальны, как в математике).

Удалим элемент из множества, используя метод erase:

```
s.erase("January");
```

Теперь множество s будет состоять из двух элементов: `February и March.`

Далее, проверим, принадлежит ли элемент множеству с помощью метода count:

```
cout << s.count("March") << endl;
// 1 - элемент принадлежит множеству
cout << s.count("January ") << endl;
// 0 - такого элемента в множестве нет
```

Для проверки принадлежности значения x множеству s можно использовать следующий код:

```
if (s.count(x) == 1) //  или  if (s.count(x) == 0)
```

```
{
     ...
}
```

Также, кроме вышеописанных методов, класс множества имеет стандартный набор средств, имеющихся у всех контейнеров стандартной библиотеки. Например, метод size() возвращает число элементов множества, а метод clear() удаляет все его элементы.

Кроме того, как и для последовательных контейнеров, например, vector или list, можно перебрать все элементы множества при помощи соответствующего итератора. Как уже говорилось ранее, такой способ перебора элементов всегда дает упорядоченную по возрастанию последовательность.

Вывести все элементы множества можно так:

```
set <string> s;
set <string>::iterator i;
for (i = s.begin(); i != s.end(); ++i)
     cout << *i << " ";
cout << endl;
```

Наличие итератора позволяет перебирать элементы множества с помощью короткого цикла for:

```
for (string x : s)
     cout << x << " ";
cout << endl;
```

Если это действие часто встречается в программе, то можно оформить его в виде функции:

```
void print(set <string> s)
{
 for (auto x : s)
     cout << x << endl;
}
```

Здесь тип переменной x подбирается компилятором автоматически, исходя из типа контейнера s.

Замечание 1.

Переменную x можно передать в цикл for по константной ссылке (чтобы избежать накладных расходов на копирование). Так же можно поступить и с параметром s функции print:

```
void print(const set <string> & s)
{
     for (const auto & x : s)
          cout << x << endl;
```

```
}
```

Благодаря тому, что множества хранятся в упорядоченном виде, все элементы будут выведены в порядке возрастания значений.

Замечание 2. Поскольку элементы множества хранятся в дереве двоичного поиска, позиция, занимаемая конкретным элементом, зависит от его величины, т. е. от результатов сравнения его с другими элементами множества. В частности, это означает, что нельзя менять элемент множества через итератор, т. е. писать так:

*i = новое значение элемента;

где i – итератор множества. Если нужно поменять значение какого-либо элемента в множестве, нужно сначала удалить старое значение методом erase, и затем вставить новое методом insert.

Задачи.

Все задания обязательные (общая сумма 7 баллов)

1. *(1 балл)* Написать функцию, вводящую с клавиатуры набор целых чисел (в любом порядке, конец ввода – Ctrl Z на пустой строчке), и выводящую числа из этого набора в порядке возрастания и без повторений.

2. *(2 балла)* Написать функцию, вводящую с клавиатуры набор целых чисел (в любом порядке, конец ввода – Ctrl Z на пустой строчке), и выводящую числа из этого набора, которые при вводе встречались не меньше двух раз.

3. *(2 балла)* Написать функцию, вводящую с клавиатуры набор целых чисел (в любом порядке, конец ввода – Ctrl Z на пустой строчке), и выводящую числа из этого набора, которые при вводе встречались ровно два раза.

4. *(2 балла)* Написать шаблонную функцию, принимающую два множества и возвращающую их объединение.

Комментарий к задаче № 4.

Если нужно объявить итератор для шаблонного класса, то нужно написать ключевое слово typename.

```
typename set<T>::iterator i;
```

Почему? Мы подсказываем компилятору, что set<T>::iterator – это тип, а не что-то другое.