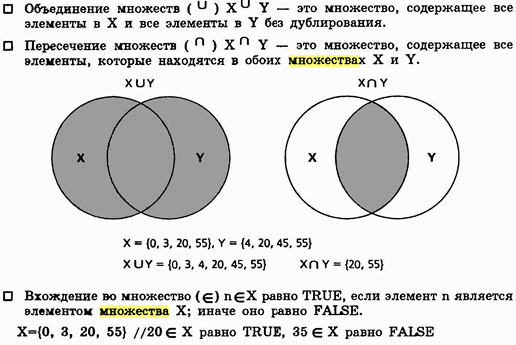
Целочисленные множества

Множество — это группа объектов, выбранная из коллекции, называемой универсальным множеством (universal set). Множество записывается как спи­ сок, разделяемый запятыми и заключенный в фигурные скобки.

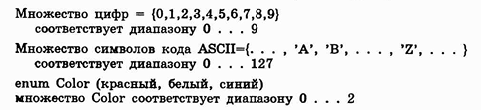


Множества целочисленных типов

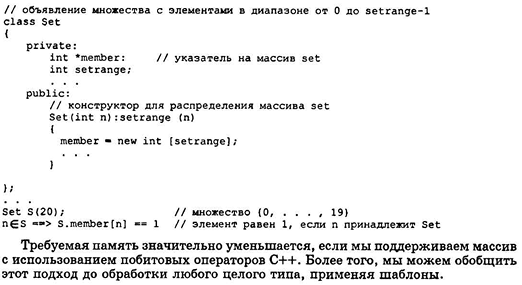
Целочисленный тип (integer type) — это любой тип, элементы которого представлены целыми значениями. Типы char, int всех размеров и пере­ числения являются целочисленными типами. Например, набор символов кода ASCII соответствует 8-битовым целым в дипазоне от 0 до 127. Тогда как приложения используют традиционное представление символов ’А ’, ’В’, для их внутреннего хранения применяются целые коды 65, 66 и так далее. Программист имеет возможность выбирать любое представление.



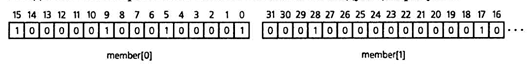
В этом разделе разрабатываются множества с элементами целочислен­ ного типа. Универсальное множество имеет соответствие "один-к-одному" с беззнаковыми целыми в диапазоне от 0 до setrange-1, где setrange — это количество элементов множества. Рассмотрим следующие множества:

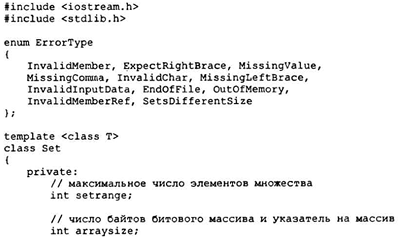


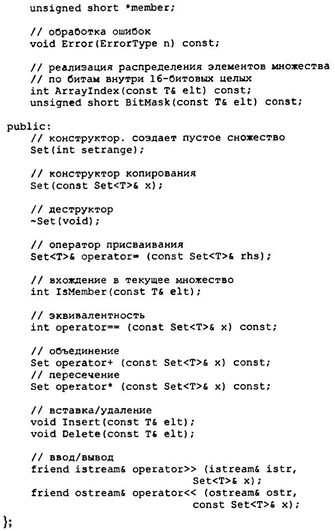
Мы можем реализовать тип Set, с помощью массива значений нулей и единиц (0 и 1). В этом массиве значение в позиции i равно 1 (TRUE), если элемент i находится в данном множестве, или — О (FALSE), если он отсутствует в нем. В письменном упражнении 6.13 описывается метод для реализации множества целых значенией с использованием статического массива. Этот под­ ход использует одно целое значение для каждого возможного элемента множе­ ства. Мы можем выделять память для этого массива целых динамически.



При определении битовых операторов C++ для эффективной реализации объ­ екта Set используются отдельные биты в слове. Диапазон значений множества (0..setrange-1) хранится в динамическом массиве из 16-битовых беззнаковых целых. Массив с именем member связывает целые числа в диапазоне 0..setsize-1 как цепочку битов. Каждый бит представляет один элемент множества, и элемент находится в этом множестве, если соответствующий бит равен 1. Нулевой элемент множества представлен крайним правым битом первого элемента массива, а 15 представляется крайним левым битом первого элемента массива. Далее продол­ жаем крайним правым битом второго элемента массива, представляющим 16, и так далее. Схема хранения в памяти показана на следующем рисунке.







Код, реализующий класс Set, находится в файле set.h.

Класс Set предоставляет клиенту возможность создания множества объектов определяемых пользователем перечислимых типов и стандартных целочисленных типов, таких как int и char. Если требуется ввод/вывод множества для перечислимых типов, потоковые операторы должны быть перегружены.

## АТД класса Set

Операция: **Add**

* Условия входные: Нет особых условий.
* Что на вход: Элемент типа **T**, который нужно добавить в множество.
* Процесс: Если элемент уже находится в множестве, он не будет добавлен. В противном случае элемент будет добавлен в конец списка элементов множества.
* Выход: Нет.
* Постусловия: Элемент добавлен в множество, если его еще не было там.

Операция: **Remove**

* Условия входные: Нет особых условий.
* Что на вход: Элемент типа **T**, который нужно удалить из множества.
* Процесс: Если элемент находится в множестве, он будет удален из списка элементов множества. В противном случае никаких изменений не будет.
* Выход: Нет.
* Постусловия: Элемент удален из множества, если он там находился.

Операция: **Contains**

* Условия входные: Нет особых условий.
* Что на вход: Элемент типа **T**, наличие которого нужно проверить в множестве.
* Процесс: Если элемент находится в множестве, метод возвращает **true**. В противном случае метод возвращает **false**.
* Выход: Булево значение **true**, если элемент находится в множестве, и **false** в противном случае.
* Постусловия: Нет.

Операция: **Count**

* Условия входные: Нет особых условий.
* Что на вход: Нет.
* Процесс: Метод возвращает количество элементов в множестве.
* Выход: Целочисленное значение, равное количеству элементов в множестве.
* Постусловия: Нет.

Операция: **Clear**

* Условия входные: Нет особых условий.
* Что на вход: Нет.
* Процесс: Метод очищает список элементов множества.
* Выход: Нет.
* Постусловия: Множество не содержит элементов после вызова метода.

Операция: **Union**

* Условия входные: Нет особых условий.
* Что на вход: Другой объект типа **Set<T>**, с которым нужно объединить текущее множество.
* Процесс: Метод создает новый объект **Set<T>** и добавляет в него все элементы текущего множества и другого множества. Дубликаты не добавляются.
* Выход: Новый объект типа **Set<T>**, который содержит все уникальные элементы обоих множеств.
* Постусловия: Ни одно из исходных множеств не изменено, новый объект содержит все элементы обоих множеств.

Операция: **Intersection**

* Условия входные: Нет особых условий.
* Что на вход: Другой объект типа **Set<T>**, с которым нужно найти пересечение.
* Процесс: Метод создает новый объект **Set<T>** и добавляет в него только те элементы, которые присутствуют в обоих множествах. Дубликаты не добавляются.
* Выход: Новый объект типа **Set<T>**, который содержит все уникальные элементы, которые присутствуют и в текущем, и в переданном множестве.
* Постусловия: Ни одно из исходных множеств не изменено, новый объект содержит только те элементы, которые присутствуют в обоих множествах.

Операция: **Difference**

* Условия входные: Нет особых условий.
* Что на вход: Другой объект типа **Set<T>**, с которым нужно найти разность.
* Процесс: Метод создает новый объект **Set<T>** и добавляет в него только те элементы, которые присутствуют в текущем множестве, но отсутствуют в переданном. Дубликаты не добавляются.
* Выход: Новый объект типа **Set<T>**, который содержит все уникальные элементы, которые присутствуют в текущем множестве, но отсутствуют в переданном.
* Постусловия: Ни одно из исходных множеств не изменено, новый объект содержит только те элементы, которые присутствуют в текущем множестве, но отсутствуют в переданном.

source: https://metanit.com/cpp/tutorial/7.13.php