ЛАБОРАТОРНАЯ РАБОТА №8
БИБЛИОТЕКА STL

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Целью данной лабораторной работы является знакомство с библиотекой STL – стандартной библиотекой шаблонов - в языке C++, а также показать ее использование на примерах.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Собственно, сам механизм шаблонов был встроен в компилятор C++ с целью дать возможность программистам C++ создавать эффективные и компактные библиотеки. Естественно, через некоторое время была создана одна из библиотек, которая впоследствии и стала стандартной частью C++. STL это самая эффективная библиотека для C++, существующая на сегодняшний день.

Сегодня существует великое множество реализаций стандартной библиотеки шаблонов, которые следуют стандарту, но при этом предлагают свои расширения, что является с одной стороны плюсом, но, с другой, не очень хорошо, поскольку не всегда можно использовать код повторно с другим компилятором. Поэтому я рекомендую вам все же оставаться в рамках стандарта, даже если вы в дальнейшем очень хорошо разберетесь с реализацией вашей библиотеки.

Начнем рассмотрение с краткого обзора основных коллекций. Каждая STL коллекция имеет собственный набор шаблонных параметров, который необходим ей для того, чтобы на базе шаблона реализовать тот или иной класс, максимально приспособленный для решения конкретных задач. Какой тип коллекции вы будете использовать, зависит от ваших задач, поэтому необходимо знать их внутреннее устройство для наиболее эффективного использования. Рассмотрим наиболее часто используемые типы коллекций. Реально в STL существует несколько большее количество коллекций, но, как показывает практика, нельзя объять необъятное сразу. Поэтому, для начала, рассмотрим наиболее популярные из них, которые с большой

вероятностью могут встретиться в чужом коде. Тем более, что этих коллекций более чем достаточно для того, чтобы решить 99% реально возникающих задач.

vector - коллекция элементов T, сохраненных в массиве, увеличиваемом по мере необходимости. Для того, чтобы начать использование данной коллекции, включите #include <vector>.

list - коллекция элементов T, сохраненных, как двунаправленный связанный список. Для того, чтобы начать использование данной коллекции, включите #include <list>.

map - это коллекция, сохраняющая пары значений pair<const Key, T>. Эта коллекция предназначена для быстрого поиска значения T по ключу const Key. В качестве ключа может быть использовано все, что угодно, например, строка или int но при этом необходимо помнить, что главной особенностью ключа является возможность применить к нему операцию сравнения. Быстрый поиск значения по ключу осуществляется благодаря тому, что пары хранятся в отсортированном виде. Эта коллекция имеет соответственно и недостаток - скорость вставки новой пары обратно пропорциональна количеству элементов, сохраненных в коллекции, поскольку просто добавить новое значение в конец коллекции не получится. Еще одна важная вещь, которую необходимо помнить при использовании данной коллекции - ключ должен быть уникальным. Для того, чтобы начать использование данной коллекции, включите #include <map>. Если вы хотите использовать данную коллекцию, чтобы избежать дубликатов, то вы избежите их только по ключу.

set - это коллекция уникальных значений const Key - каждое из которых является также и ключом - то есть, проще говоря, это отсортированная коллекция, предназначенная для быстрого поиска необходимого значения. К ключу предъявляются те же требования, что и в случае ключа для map. Естественно, использовать ее для этой цели нет смысла, если вы хотите сохранить в ней простые типы данных, по меньшей мере вам необходимо определить свой класс, хранящий пару ключ - значение и определяющий операцию сравнения по ключу. Очень удобно использовать данную коллекцию, если вы хотите избежать повторного сохранения одного и того же значения.

Для того, чтобы начать использование данной коллекции, включите #include <set>.

multimap - это модифицированный map, в котором отсутствует требования уникальности ключа - то есть, если вы произведете поиск по ключу, то вам вернется не одно значение, а набор значений, сохраненных с данным ключом. Для того, чтобы начать использование данной коллекции включите #include <map>.

multiset - то же самое относится и к этой коллекции, требования уникальности ключа в ней не существует, что приводит к возможности хранения дубликатов значений. Тем не менее, существует возможность быстрого нахождения значений по ключу в случае, если вы определили свой класс. Поскольку все значения в map и set хранятся в отсортированном виде, то получается, что в этих коллекциях мы можем очень быстро отыскать необходимое нам значение по ключу, но при этом операция вставки нового элемента T будет стоить нам несколько дороже, чем, например, в vector. Для того, чтобы начать использование данной коллекции, включите #include <set>.

STL Строки

Не существует серьезной библиотеки, которая бы не включала в себя свой класс для представления строк или даже несколько подобных классов. STL - строки поддерживают как формат ASCII, так и формат Unicode.

string - представляет из себя коллекцию, хранящую символы char в формате ASCII. Для того, чтобы использовать данную коллекцию, вам необходимо включить #include <string>.

wstring - это коллекция для хранения двухбайтных символов wchar_t, которые используются для представления всего набора символов в формате Unicode. Для того, чтобы использовать данную коллекцию, вам необходимо включить #include <xstring>.

Строковые потоки

Используются для организации сохранения простых типов данных в STL строки в стиле C++. Практическое знакомство с STL мы

начнем именно с этого класса. Ниже приведена простая программа, демонстрирующая возможности использования строковых потоков:

```
1) //stl.cpp: Defines the entry point for the console
   application
2) #include "stdafx.h"
3) #include <iostream>
4) #include <strstream>
5) #include <string>
6) using namespace std;
7) int _tmain (int argc, _TCHAR* argv [])
8) {
9) strstream xstr;
10)  for (int i = 0; i < 10; i++)
11)  {
12)  xstr << "Demo " << i << endl;
13)  }
14)  cout << xstr.str ();
15)  string str;
16)  str.assign (xstr.str (), xstr.pcount ());
17)  cout << str.c_str ();
18)  return 0;
19)  }
```

Строковый поток - это просто буфер, в конце которого установлен нуль терминатор, поэтому мы наблюдаем в конце строки мусор при первой распечатке, то есть реальный конец строки определен не посредством нуль терминатора, а с помощью счетчика, и его размер мы можем получить с помощью метода: pcount ().

Далее мы производим копирование содержимого буфера в строку и печатаем строку второй раз. На этот раз она печатается без мусора.

Основные методы, которые присутствуют почти во всех STL коллекциях, приведены ниже.

empty - определяет, является ли коллекция пустой.

size - определяет размер коллекции.

begin - возвращает прямой итератор, указывающий на начало коллекции.

end - возвращает прямой итератор, указывающий на конец коллекции. При этом надо учесть, что реально он не указывает на ее последний элемент, а указывает на воображаемый несуществующий элемент, следующий за последним.

rbegin - возвращает обратный итератор, указывающий на начало коллекции.

rend - возвращает обратный итератор, указывающий на конец коллекции. При этом надо учесть, что реально он не указывает на ее последний элемент, а указывает на воображаемый несуществующий элемент, следующий за последним.

clear - удаляет все элементы коллекции, при этом, если в вашей коллекции сохранены указатели, то вы должны не забыть удалить все элементы вручную посредством вызова delete для каждого указателя.

erase - удаляет элемент или несколько элементов из коллекции.

capacity - вместимость коллекции определяет реальный размер - то есть размер буфера коллекции, а не то, сколько в нем хранится элементов. Когда вы создаете коллекцию, то выделяется некоторое количество памяти. Как только размер буфера оказывается меньшим, чем размер, необходимый для хранения всех элементов коллекции, происходит выделение памяти для нового буфера, а все элементы старого копируются в новый буфер. При этом размер нового буфера будет в два раза большим, чем размер буфера, выделенного перед этим - такая стратегия позволяет уменьшить количество операций перераспределения памяти, но при этом очень расточительно расходуется память. Причем в некоторых реализациях STL первое выделение памяти происходит не в конструкторе, а как ни странно, при добавлении первого элемента коллекции. Фрагмент программы ниже демонстрирует, что размер и вместимость коллекции - две разные сущности:

```
1) vector<int> vec;
2) cout << "Real size of array in vector: " <<
   vec.capacity () << endl;
3) for (int j = 0; j < 10; j++)
4) {
```

```
5) vec.push_back (10);
6) }
7) cout << "Real size of array in vector: " <<
   vec.capacity () << endl;
8) return 0;
```

ЗАДАНИЕ

1) Используйте шаблон vector для массива данных о студентах.

2) Используйте шаблон list для двусвязного списка данных класса Complex.

3) Используйте шаблон queue для очереди авто на мойке.