# Лабораторная работа №8. Работа со стандартной библиотекой шаблонов STL

**1 Цель работы**

Изучить теоретические принципы и получить практические навыки разработки программ в среде Visual Studio 2005 с использованием стандартной библиотеки шаблонов STL*.*

**2 Порядок выполнения работы**

Получить задание на выполнение лабораторной работы (раздел 6) согласно своему варианту. Разработать и отладить программу. Составить и защитить отчет о лабораторной работе у преподавателя.

**3 Содержание отчета**

* наименование и цель работы;
* задание на лабораторную работу согласно варианту;
* схема алгоритма, текст программы на алгоритмическом языке;
* результаты работы программы.

**4 Краткая теория**

Значительную часть библиотеки классов С++ составляет *стандартная библиотека шаблонов (standard template library - STL)*. Эта библиотека предоставляет шаблонные классы и функции общего назначения, которые реализуют многие популярные и часто используемые алгоритмы и структуры данных. Например, она обеспечивает поддержку векторов, списков, очередей и стеков, а также определяет различные процедуры, обеспечивающие доступ к этим объектам. Поскольку библиотека STL состоит из шаблонных классов, алгоритмы и структуры данных могут быть применены к данным практически любого типа.

**4.1 Обзор библиотеки STL**

Ядро стандартной библиотеки шаблонов составляют три основных элемента: *контейнеры, алгоритмы* и *итераторы*. Они работают совместно, предоставляя готовые решения различных проблем программирования.

*Контейнеры* – это объекты, содержащие другие объекты. Существует несколько типов контейнеров:

* Последовательные. Представляют собой линейный список. Примеры: вектора, очереди, списки.
* Ассоциативные. Позволяют эффективно находить нужные значения на основе заданных ключей. Примеры: отображения, множества.

Каждый контейнерный класс определяет набор функций, которые можно применять к данному контейнеру. Например, контейнер списка включает функции, предназначенные для выполнения вставки, удаления и объединения элементов. А стек включает функции, которые позволяют помещать значения в стек и извлекать их из стека.

*Алгоритмы* действуют в контейнерах. Они позволяют выполнять инициализацию, сортировку, поиск и преобразовывать содержимое контейнеров. Многие алгоритмы работают с группой (или диапазоном) элементов внутри контейнера.

*Итераторы –* это объекты, которые действуют подобно указателям. Они позволяют циклически опрашивать содержимое контейнера так же, как это делается с помощью указателя при циклическом опросе элементов массива. существует 5 типов итераторов (таблица 8.1):

*Таблица 8.1 – Виды итераторов*

|  |  |
| --- | --- |
| **Итераторы** | **Разрешаемый доступ** |
| Произвольного доступа (random access) | Сохраняют и считывают значения; к элементам позволяют организовать произвольный доступ |
| Двунаправленные (bidirectional) | Сохраняют и считывают значения; обеспечивают инкрементно-декрементное перемещение |
| Однонаправленные (forward) | Сохраняют и считывают значения; обеспечивают только инкрементное перемещение |
| Входные (input) | Считывают, но не записывают значений; обеспечивают только инкрементное перемещение |
| Выходные (output) | Записывают, но не считывают значения; обеспечивают только инкрементное перемещение |

Итераторы обрабатываются аналогично указателям. Их можно инкрементировать и декрементировать. К ним можно применять операцию разыменования \*. Итераторы объявляются с помощью типа **iterator**, определяемого различными контейнерами.

Библиотека STL поддерживает *реверсивные итераторы*, которые являются либо двунаправленными, либо итераторами произвольного доступа, т.е. позволяют перемещаться по последовательности в обратном направлении. Следовательно, если реверсивный итератор указывает на конец последовательности, то после инкрементирования он будет указывать на элемент, расположенный перед концом последовательности.

**4.2 Контейнерные классы**

В таблице 8.2 приведены контейнеры, определенные библиотекой STL.

*Таблица 8.2 – Контейнеры библиотеки STL*

|  |  |  |
| --- | --- | --- |
| **Контейнер** | **Описание** | **Заголовок** |
| **bitset** | Битовое множество | **<bitset>** |
| **deque** | Дек (очередь с двусторонним доступом) | **<deque>** |
| **list** | Линейный список | **<list>** |
| **map** | Отображение. Хранит пары ключ/значение, в которых каждый ключ связан только с одним значением | **<map>** |
| **multimap** | Мультиотображение. Хранит пары ключ/ значение, в которых каждый ключ связан с двумя или более значениями | **<map>** |
| **multiset** | Множество, в котором элементы могут быть не уникальными (мультимножество) | **<set>** |
| **priority\_queue** | Очередь с приоритетами | **<queue>** |
| **queue** | Очередь | **<queue>** |
| **set** | Множество, в котором каждый элемент уникален | **<set>** |
| **stack** | Стек | **<stack>** |
| **vector** | Динамический массив | **<vector>** |

Рассмотрим подробнее основные из перечисленных контейнеров.

**4.2.1 Списки и динамические массивы**

Класс **vector** поддерживает динамический массив. Класс **list** поддерживает работу списка. Классы имеют следующие конструкторы (в упрощенном виде), представленные в таблице 8.3. Здесь **T** – тип сохраняемых данных.

*Таблица 8.3 – Конструкторы классов vector и list*

|  |  |  |
| --- | --- | --- |
| 1 | **Конструктор** | **vector();**  **list();** |
| Описание | Создает пустой вектор или список |
| **Пример** | **vector<int> v0;**  **list<char> c0;** |
| 2 | **Конструктор** | **vector(int num, T val);**  **list(int num, T val);** |
| Описание | Создает контейнер, который содержит **num** элементов со значением **val** |
| **Пример** | **vector<int> v1(5,2);**  **list<char> c1(3,’a’);** |
| 3 | **Конструктор** | **vector(vector<T> ob);**  **list(list<T> ob);** |
| Описание | Создает вектор или список, который содержит те же элементы, что и контейнер **ob** |
| **Пример** | **vector<int> v2(v1);**  **list<char> c2(c1);** |
| 4 | **Конструктор** | **vector(итератор start, итератор end)**  **list(итератор start, итератор end)** |
| Описание | Создает вектор или список, который содержит элементы в диапазоне, заданном параметрами **start** и **end** |
| **Пример** | **vector<int> v3(v2.begin()+1,v2.begin()+3);**  **list<char>::iterator it = c1.begin();**  **it++;**  **list<char> c3(c1.begin(),it);** |

Практически для всех контейнеров определены следующие операторы сравнения: ==, <, <=, !=, >, >=.

Класс **vector** и **list** содержат следующие компонентные функции (наиболее часто используемые), приведенные в таблице 8.4.

*Таблица 8.4 – Основные компонентные функции классов vector и list*

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **assign(итератор start, итератор end)** | Помещает в контейнер последовательность, определяемую параметрами **start** и **end** |
| **assign(num,val)** | Помещает в контейнер **num** элементов со значением **val** |
| **begin()** | Возвращает итератор для первого элемента в контейнере |
| **end()** | Возвращает итератор для конца контейнера |
| **front()** | Возвращает ссылку на первый элемент контейнера |
| **back()** | Возвращает ссылку на последний элемент контейнера |
| **empty()** | Возвращает **true**, если контейнер пуст, иначе **false** |
| **max\_size()** | Возвращает максимальное число элементов, которое может содержать контейнер |
| **size()** | Возвращает текущее количество элементов контейнера |
| **clear()** | Удаляет все элементы из контейнера |
| **push\_back(val)** | Добавляет в конец контейнера элемент со значением **val** |
| **pop\_back()** | Удаляет последний элемент контейнера |
| **resize(num,val)** | Устанавливает размер контейнера равным значению **num**. Если контейнер для этого нужно удлинить, то в его конец добавляются элементы со значением **val** |
| **insert(i,val)** | Вставляет значение **val** непосредственно перед элементом, заданным параметром **i**; возвращает итератор для этого элемента |
| **insert(i,num,val)** | Вставляет **num** копий значения **val** непосредственно перед элементом, заданным параметром **i** |
| **insert(i, итератор start, итератор end)** | Вставляет в контейнер последовательность, определяемую параметрами **start** и **end**, непосредственно перед элементом, заданным параметром **i** |
| **erase(i)** | Удаляет элемент, адресуемый итератором **i**; возвращает итератор для элемента, расположенного после удаленного |
| **erase(итератор start, итератор end)** | Удаляет элементы в диапазоне, задаваемом параметрами **start** и **end**; возвращает итератор для элемента, расположенного за последним удаленным элементом |
| **swap(ob)** | Выполняет обмен элементами вызывающего контейнера и контейнера **ob** |

Класс **vector** содержит некоторые дополнительные функции (таблица 8.5).

*Таблица 8.5 – Дополнительные функции класса vector*

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **capacity()** | Возвращает текущую емкость вектора, или количество элементов, которое может храниться в векторе до того, как возникнет потребность в выделении дополнительной памяти |
| **reserve(num)** | Устанавливает емкость вектора равной не менее заданного значения **num** |

Класс **list** также содержит некоторые дополнительные функции (таблица 8.6).

*Таблица 8.6 – Дополнительные функции класса list*

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **push\_front(val)** | Добавляет в начало списка элемент со значением **val** |
| **pop\_front()** | Удаляет первый элемент списка |
| **merge(ob)** | Объединяет упорядоченный список **ob** с упорядоченным списком вызывающего объекта. Результат также упорядочивается. После объединения список ob остается пустым |
| **merge(ob,cmpfn)** | Аналогично. Параметр **cmpfn** задает функцию сравнения, которая определяет, когда один элемент меньше другого |
| **remove(val)** | Удаляет из списка элементы со значением **val** |
| **reverse()** | Реверсирует список |
| **sort()** | Сортирует список |
| **sort(cmpfn)** | Аналогично. Параметр **cmpfn** задает функцию сравнения, которая определяет, когда один элемент меньше другого |
| **unique()** | Удаляет из списка элементы-дубликаты |

**4.2.2 Стеки, очереди, множества**

Класс **queue** поддерживает очередь с односторонним доступом, класс **stack** - стек. Конструкторы классов создают соответственно пустую очередь и пустой стек. По умолчанию в качестве контейнера используется дек, но к очереди можно получить доступ по принципу «первый пришел – первый ушел», а к стеку – «последний пришел – первый ушел». Можно также в качестве контейнера использовать список, а для стека – еще и вектор. Примеры:

**queue <char> q1;**

**queue <int, list<int> > q2;**

**stack <char> dsc1;**

**stack <int, vector<int> > vsi1;**

**stack <int, list<int> > lsi;**

Классы **queue** и **stack** содержат перечисленные ниже компонентные функции (таблица 8.7).

*Таблица 8.7 – Компонентные функции классов queue и stack*

|  |  |  |
| --- | --- | --- |
| **Функция** | **Очередь** | **Стек** |
| **front()** | Возвращает ссылку на первый элемент в очереди | – |
| **back()** | Возвращает ссылку на последний элемент в очереди | – |
| **empty()** | Возвращает **true**, если контейнер пуст, иначе **false** | |
| **size()** | Возвращает текущее количество элементов в контейнере | |
| **push(val)** | Добавляет элемент **val** в конец очереди | Помещает элемент **val** в вершину (конец) стека |
| **pop()** | Удаляет первый элемент из очереди | Удаляет верхний (последний) элемент из стека |
| **top()** | – | Возвращает ссылку на вершину стека |

Класс **set** поддерживает множество, в котором неуникальным ключам соответствуют определенные значения, и элементы упорядочены по значению ключа. Класс имеет следующие конструкторы:

**set(); //пустое множество**

**set(ob); //множество из элементов другого контейнера**

**set(итератор start, итератор end) //или части контейнера**

Например:

**set<int> s1;**

**set<int> s2(s1);**

**set<int> s3(s1.begin(),s2.end());**

Класс **set** содержит следующие функции (таблица 8.8).

*Таблица 8.8 – Основные функции класса set*

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| **begin()** | Возвращает итератор для первого элемента множества |
| **end()** | Возвращает итератор для конца множества |
| **empty()** | Возвращает **true**, если множество пустое, иначе **false** |
| **max\_size()** | Возвращает максимальное число элементов, которое может содержать множество |
| **size()** | Возвращает текущее количество элементов множества |
| **clear()** | Удаляет все элементы из множества |
| **insert(val)** | Вставляет значение **val** в множество, если его еще нет в множестве |
| **insert(i,val)** | Вставляет значение **val** после позиции **i**. Элементы-дубликаты не вставляет. Возвращает итератор для этого элемента |
| **insert(итератор start, итератор end)** | Вставляет элементы заданного диапазона. Дубликаты не вставляет |
| **erase(k)** | Удаляет элементы с ключом **k**; возвращает количество удаленных элементов |
| **erase(итератор start, итератор end)** | Удаляет элементы в диапазоне от **start** до **end** |
| **find(k)** | Возвращает итератор для ключа **k**. Если ключ не найден, возвращает итератор конца множества |
| **lower\_bound(k)** | Возвращает итератор для первого элемента множества, ключ которого >= **k** |
| **upper\_bound(k)** | Возвращает итератор для первого элемента множества, ключ которого >**k** |
| **count(k)** | Возвращает число вхождений ключа **k** в множество |
| **swap(ob)** | Выполняет обмен элементами вызывающего множества и множества **ob** |

Примечание. Если нужно создать множество, элементы которого представляют собой составной тип данных (например, класс), то для этого типа нужно перегрузить операцию <, т.е. необходимо указать, по какому именно полю нужно упорядочивать данные.

**4.3 Алгоритмы библиотеки STL**

Алгоритмы работают с данными, содержащимися в контейнерах, посредством итераторов. Все алгоритмы представляют собой шаблонные функции. Их использование требует включения заголовочного файла **<algorithm>**. Перечень основных алгоритмов приведен в таблице 8.9.

*Таблица 8.9 – Основные алгоритмы библиотеки STL*

|  |  |
| --- | --- |
| **Алгоритм** | **Описание** |
| **adjacent\_find(start,end)** | Поиск совпадающих смежных элементов внутри последовательности от **start** до **end**, возвращает итератор для первого элемента. Если ни одной смежной пары не обнаружено, возвращает **end**. |
| **binary\_search(start,end, val)** | Бинарный поиск значения **val** внутри упорядоченной последовательности **start** и **end**. Возвращает **true**, если значение найдено, иначе **false** |
| **copy(start,end,result)** | Копирует последовательность от **start** до **end** и помещает результат в последовательность **result**. |
| **count(start,end,val)** | Возвращает количество элементов со значением **val** в последовательности **start** и **end**. |
| **equal(start1,end1, start2)** | Определяет, одинаковы ли диапазон **start1** и **end1** и диапазон, адресуемый параметром **start2**. |
| **equal\_range(start,end, val)** | Возвращает диапазон, в который можно вставить элемент **val**, не нарушая порядок последовательности. Область, где ведется поиск диапазона, определяется параметрами **start** и **end** |
| **fill(start,end,val)**  **fill\_n(start,num,val)** | Заполняют диапазон значением **val**. В первом случае диапазон задается параметрами **start** и **end**. Во втором – задается начало диапазона **start** и количество элементов **num** |
| **find(start,end,val)** | Поиск элемента **val** в диапазоне **start** и **end**. Возвращает итератор для первого вхождения элемента или **end**, если элемент не обнаружен |
| **find\_first\_of(start1, end1, start2,end2)** | Поиск первого элемента внутри последовательности **start** и **end**, который совпадает с каким-либо элементом из диапазона **start2** и **end**. |
| **for\_each(start,end,fn)** | Применяет функцию **fn** к диапазону элементов **start** и **end** |
| **generate(start,end, fngen)** | Присваивает значения, возвращаемые функцией-генератором **fngen**, элементам из диапазона **start** и **end** |
| **includes(start1,end1, start2,end2)** | Возвращает **true**, если все элементы последовательности **start1** и **end1** обнаружены в последовательности **start2** и **end2** |
| **iter\_swap(i,j)** | Меняет местами значения, адресуемые итераторами **i** и **j** |
| **lexicographical\_compare (start1,end1,start2, end2)** | Сравнивает последовательность **start1** и **end1** с последовательностью **start2** и **end2**. Возвращает **true**, если первая последовательность лексически меньше второй |
| **lower\_bound(start,end, val)** | Поиск первого элемента в последовательности **start** и **end**, значение которого не меньше **val**. Возвращает итератор для этого элемента |
| **max(i,j)** | Максимум из двух значений |
| **max\_element(start,end)** | Возвращает итератор для наибольшего элемента внутри диапазона **start** и **end** |
| **merge(start1,end1, start2,end2,result)** | Объединяет две упорядоченные последовательности **start1** – **end1** и **start2** – **end2**, помещая результат в последовательность **result**. |
| **min(i,j)** | Минимум из двух значений |
| **min\_element(start,end)** | Возвращает итератор для наименьшего элемента внутри диапазона **start** и **end** |
| **mismatch(start1,end1, start2)** | Выполняет поиск первого несовпадения элементов в двух последовательностях и возвращает итераторы для этих двух элементов |
| **nth\_element(start,el, end)** | Упорядочивает последовательность **start** и **end** так, чтобы все элементы, значения которых меньше **el**, размещались перед этим элементом, а все элементы, значения которых больше, размещались после него |
| **random\_shuffle(start, end)** | Заполняет случайными числами последовательность **start** и **end** |
| **remove(start,end,val)** | Удаляет из диапазона **start** и **end** элементы со значениями **val** |
| **replace(start,end,old, new)** | Заменяет в диапазоне **start** и **end** элементы, которые имеют значение **old**, элементами, имеющими значение **new** |
| **reverse(start,end)** | Меняет порядок следования элементов диапазона **start** и **end** на противоположный |
| **rotate(start,mid,end)** | Циклический сдвиг влево элементов диапазона **start** и **end** таким образом, чтобы элемент **mid** стал первым. |
| **search(start1,end1, start2,end2)** | Поиск последовательности **start2** – **end2** внутри последовательности **start1** – **end1**. Если последовательность найдена, возвращается итератор ее начала, иначе – **end1**. |
| **set\_difference(start1, end1,start2, end2, result)** | Создает последовательность, которая содержит разность двух упорядоченных множеств с параметрами **start1** – **end1** и **start2** – **end2**. Результат упорядочивается и помещается в последовательность **result** |
| **set\_intersection(start1,end1,start2, end2,result)** | Аналогично: пересечение множеств |
| **set\_symmetric\_difference(start1,end1, start2,end2,result)** | Аналогично: симметрическая разность множеств (элементы, которые не являются общими для обоих множеств) |
| **set\_union(start1,end1,start2, end2,result)** | Аналогично: объединение множеств |
| **sort(start,end)** | Сортирует значения диапазона **start** и **end** |
| **swap(i,j)** | Меняет местами значения **i** и **j** |
| **swap\_ranges(start1,end1,start2)** | Обмен элементов диапазона **start1** и **end1** и элементов последовательности, начало которой задается параметром **start2** |
| **transform(start,end, result,func)** | Применяет функцию **func** к диапазону **start** – **end** и сохраняет результат в последовательности **result** |
| **unique(start,end)** | Удаляет повторяющиеся элементы из диапазона **start** и **end** |
| **upper\_bound(start,end, val)** | Поиск последнего элемента в последовательности **start** и **end**, значение которого не больше **val**. Возвращает итератор для этого элемента |

**5 Примеры программ**

Пример 1. Работа с классом vector.

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**void main()**

**{**

**//создаем вектор нулевой длины**

**vector<int> v;**

**//отображаем исходный размер вектора**

**cout<<"Size = "<<v.size()<<endl;**

**//помещаем значения в конец вектора,**

**//вектор будет расти по мере необходимости**

**for (int i=0;i<10;i++)**

**v.push\_back(i);**

**//отображаем текущий размер вектора**

**cout<<"Size = "<<v.size()<<endl;**

**//доступ к содержимому вектора через индексы**

**for (int i=0;i<10;i++)**

**cout<<v[i]<<' ';**

**cout<<endl;**

**//доступ к первому**

**cout<<"First = "<<v.front()<<endl;**

**//и последнему элементам вектора**

**cout<<"Last = "<<v.back()<<endl;**

**//доступ через итератор**

**vector<int>::iterator p = v.begin();**

**while (p!=v.end())**

**{**

**cout<<\*p<<' ';**

**p++;**

**}**

**cout<<endl;**

**}**

Пример 2. Класс «студент». Поля: ФИО, курс. Создать множество из объектов этого класса.

**#include <iostream>**

**#include <set>**

**#include <string.h>**

**using namespace std;**

**//описание класса**

**struct student**

**{**

**char\* fio;**

**int kurs;**

**//конструктор класса**

**student(char\* f,int k)**

**{**

**fio = f;**

**kurs = k;**

**}**

**};**

**//перегрузка операции < для создания множества объектов**

**bool operator <(const student s1,const student s2)**

**{**

**if (strcmp(s1.fio,s2.fio)<0)**

**return true;**

**return false;**

**}**

**void main()**

**{**

**//создаем пустое множество студентов**

**set<student> s;**

**//создаем 4 объекта класса**

**student st1("Petrov",3);**

**student st2("Ivanov",1);**

**student st3("Sidorov",2);**

**student st4("Petrov",4);**

**//добавляем созданные объекты в множество**

**s.insert(st1);**

**s.insert(st2);**

**s.insert(st3);**

**s.insert(st4);**

**//печатаем содержимое множества с помощью итератора**

**set<student>::iterator it;**

**for (it = s.begin(); it != s.end(); it++)**

**cout<<(\*it).fio<<' '<<(\*it).kurs<<endl;**

**}**

Пример 3. Демонстрация работы с алгоритмами библиотеки STL.

**#include <iostream>**

**#include <list>**

**#include <vector>**

**#include <algorithm>**

**using namespace std;**

**//шаблон функции печати списка**

**template <class T>**

**void print(list<T> lst)**

**{**

**//устанавливаем итератор на начало списка**

**list<T>::iterator p = lst.begin();**

**//пока не дошли до конца,**

**while (p!=lst.end())**

**{**

**//печатаем содержимое списка**

**cout<<\*p<<' ';**

**//и двигаем итератор к следующему элементу**

**p++;**

**}**

**cout<<endl;**

**}**

**//функция, которая будет применяться к элементам списка**

**int xform(int i)**

**{**

**return i\*i;**

**}**

**void main()**

**{**

**//создаем пустой список**

**list<int> x1;**

**//в цикле заполняем его элементами**

**for (int i=0;i<10;i++)**

**x1.push\_back(i);**

**//и печатаем**

**print(x1);**

**int key;**

**//вводим ключ для поиска**

**cout<<"Input key for search: ";**

**cin>>key;**

**//ищем элемент и проверяем результат поиска**

**if (binary\_search(x1.begin(),x1.end(),key))**

**cout<<"Element is found\n";**

**else**

**cout<<"Element is not found\n";**

**//устанавливаем 2 итератора: на 1-й и последний элементы**

**list<int>::iterator it1=x1.begin(),it2=x1.end();**

**//смещаем их соответственно на 2-й и предпоследний элементы**

**it1++;**

**it2--;**

**//реверсируем элементы списка со второго до предпоследнего**

**reverse(it1,it2);**

**//печатаем измененный список**

**cout<<"After reverse:\n";**

**print(x1);**

**//применяем к элементам списка функцию xform,**

**//т.е. каждый элемент возводим в квадрат**

**it1 = transform(x1.begin(),x1.end(),x1.begin(),xform);**

**//печатаем измененный список**

**cout<<"After transform:\n";**

**print(x1);**

**//создаем массив v1 - копию списка x1**

**vector<int> v1(x1.begin(),x1.end());**

**//сортируем элементы массива**

**sort(v1.begin(),v1.end());**

**cout<<"After sort:\n";**

**//печатаем массив (можно также с помощью цикла while)**

**for (vector<int>::iterator p=v1.begin(); p != v1.end(); p++)**

**cout<<\*p<<' ';**

**cout<<endl;**

**}**

**6 Контрольные вопросы**

1. Для чего предназначена библиотека STL? Из каких компонентов она состоит?

2. Что такое итератор? Перечислите известные Вам виды итераторов.

3. Дайте определение контейнеру. Какие виды контейнеров содержатся в библиотеке STL? В каких заголовочных файлах они описаны?

4. Что общего и чем отличаются списки и динамические массивы библиотеки STL? Какие операции можно над ними выполнять?

5. Укажите основные принципы работы со стеками и очередями библиотеки STL?

6. Что такое множество в библиотеке STL? Как создать множество, элементами которого являются объекты классов?

7. Перечислите основные возможности работы с алгоритмами библиотеки STL.

**7 Варианты заданий для самостоятельного решения**

Для решения использовать стандартные функции библиотеки STL

1. Класс "Доставка продуктов": наименование продукта, время (только часы), кол-во. Заполнить очередь из N (вводится с клавиатуры) заказов. Перенести из очереди в массив все заказы, которые имеют одинаковое время, отсортировать массив по возрастанию количества.
2. Класс "Успеваемость": ФИО, оценки по 4 предметам. Сформировать стек из N (вводится с клавиатуры) студентов. Распечатать список всех студентов, которые имеют средний балл >=4. Сформировать множество из тех студентов, которые не вошли в список.
3. Сформировать массив из N случайных чисел в диапазоне от m до n (N,m n вводится с клавиатуры). Удалить из массива все повторяющиеся значения. Найти min и max элементы массива, отсортировать массив по возрастанию.
4. Сформировать множество из N точек в пространстве (координаты целочисленные). Найти самую удаленную точку от начала координат. Разбить исходное множество на множества точек, лежащих на одной окружности. Найти максимальное количество точек, лежащих на одной окружности.
5. Сформировать символьное множество из текстовой строки. Посчитать число вхождений каждого символа. Найти дополнение этого множества до множества всех символов. Найти пересечение исходного множества с множеством знаков препинания.
6. Класс "Коробка": длина, ширина, высота. Сформировать массив из N коробок. Занести в стек коробки, чей объем > среднего объема. Вывести количество коробок в стеке.
7. Сформировать множество из N точек на плоскости. Сформировать массив из треугольников, чьи вершины образованы из этого множества. Отсортировать массив по возрастанию площадей. Найти max площадь.
8. Дана очередь данных о клиентах пункта проката автомобилей: ФИО, адрес (улица, дом, квартира) и марка машины. Во второй массив записать отсортированные по алфавиту данные только тех людей, кто ездит на "Audi".
9. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Дан стек из N рациональных чисел. Создать новый список из дробей, обратных исходным (числитель и знаменатель меняются местами), отсортировать его по убыванию дробей. Удалить из этого списка максимальное и минимальное значения.
10. Дан массив данных о работниках фирмы: ФИО и дата поступления на работу (месяц, год). В список записать только данные тех из них, кто на сегодняшний день проработал уже не менее 5 лет. Отсортировать данные по алфавиту.
11. Дан список иногородних студентов из ***n*** человек: ФИО, адрес (город, улица, дом-квартира), приблизительное расстояние до Краснодара. Для них в общежитии выделено ***k*** мест. Вывести очередь студентов, которых необходимо селить в общежитие в первую очередь. Критерий отбора: расстояние до города.
12. Дан список данных о студентах некоторой группы: фамилия, имя, отчество и дата рождения (день, месяц, год). Вывести на экран фамилию и имя тех студентов, у кого сегодня день рождения (сегодняшнюю дату вводить с клавиатуры). Удалить сведения об этих студентах из исходного списка.
13. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Дан список рациональных чисел. Записать в другой список все неправильные дроби, предварительно удалив их из исходного списка и преобразовав в правильные. Неправильной называется дробь, у которой числитель больше знаменателя. Определить количество элементов каждого списка. Отсортировать оба списка по убыванию числителя.
14. Рациональное число можно представить записью с двумя полями: числитель и знаменатель. Разработать метод приведения рационального числа к несократимому виду. Дан стек рациональных чисел. Сократить эти числа и записать их в список, отсортировав по убыванию.
15. Даны два списка, содержащие перечни товаров, производимых концернами SHARP и LG. Создать список товаров, выпускаемых как одной, так и другой фирмой. Отсортировать его по алфавиту. Организовать поиск в этом списке заданного товара.
16. Дан список четырехзначных номеров лотерейных билетов. Отсортировать его по возрастанию сумм цифр числа. Сформировать очередь из чисел, которые состоят из одинаковых цифр (1111, 2222, 3333…).
17. Дана очередь данных о служащих предприятия: фамилия, отдел, зарплата. Сформировать список данных по алфавиту о работниках бухгалтерии и определить их суммарную зарплату.
18. Ведомость абитуриентов, сдавших вступительные экзамены в институт, содержит: ФИО, адрес, оценки. Определить количество абитуриентов, проживающих в Краснодаре, и сдавших экзамены со средним баллом не ниже 4,5, записать их в отдельный список, отсортировать по фамилии и удалить из исходного списка.
19. Класс "АТС": дата разговора, код и название города, время разговора, тариф, номер телефона в этом городе и номер телефона абонента. Из массива записей сформировать стек, записав в него название города, общее время разговоров с ним и сумма. Организовать поиск по названию города.
20. Из массива случайных чисел сформировать массив из чисел, принадлежащих диапазону [m, n]. Инвертировать центральные 5 элементов полученного массива, найти максимум из них.
21. Класс "Дети": ФИО, пол, возраст. Из массива детей организовать список пар детей (в пару входят мальчик и девочка одного возраста). Найти самую взрослую пару.
22. Сформировать стек из записей: наименование продукта, калорийность единицы продукта, количество продукта. Найти самый калорийный продукт; найти суммарную калорийность всех продуктов. В массив перенести только те продукты, чья калорийность ниже средней.
23. Класс "Прямоугольник": длина, высота. Сформировать массив прямоугольников. Найти квадраты. Найти прямоугольник с самой большой диагональю. Отсортировать массив по площади.
24. Класс "Окружность": координаты центра окружности, радиус. Сформировать стек окружностей. Найти все пересекающиеся пары окружностей.
25. Из введенного текста сформировать массив слов. Найти все слова, которые не содержат повторяющиеся буквы. Удалить из массива слова, чья длина меньше 3 букв.
26. Класс "Вектор из начала координат": X, Y. Сформировать из массива векторов множества векторов по квадрантам. Найти самый короткий вектор.
27. Сформировать стек из записей: автор, название произведения, кол-во страниц. Организовать поиск произведений по автору, формируя множество записей. Поменять местами произведения с min и max количеством страниц.
28. Сформировать стек из основного состава игроков и очередь из запасного. Сделать N замен, меняя первых запасных игроков на последных основных.