 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Практичне заняття №6**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Робота з бібліотекою STL»

Варіант №11

**Виконав:**

студент гр. БС-52

Розюк П.В.

**Перевірив:**

ст. викладач каф. БМК

Радогуз А.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2017

**Завдання:**

1. Ознайомитися з теоретичними відомостями для роботи зі стандартною бібліотекою STL за списком рекомендованої літератури.
2. Переробити розроблений в попередній лабараторній роботі шаблонний клас із використанням контейнерів STL.
3. Скласти і захистити звіт по роботі.

**🞏 Практичне заняття без зауважень** **🞏 Практичне заняття має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 багатостатність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Літинг програми:

main.cpp

#include <iostream>

#include <time.h>

#include <string>

#include <sstream>

#include <vector>

#include "SkipList.h"

using namespace std;

void main()

{

SkipList<int> A;

int c;

int i, counter = 1;

do

{

system("cls");

cout << "1.Add Element" << endl;

cout << "2.Delete Element" << endl;

cout << "3.Print List " << endl;

cout << "4.Delete list " << endl;

cout << "5.Exit " << endl << endl;

cout << "Input : ";

cin >> i;

cout << endl;

switch (i)

{

case 1:

cout << "Input the ellement: ";

cin >> c;

A.input(counter, c);

counter++;

system("pause");

break;

case 2:

cout << "Input the ellement you want to delete: ";

int j;

cin >> j;

if (j > counter || j < 0)

{

cout << "Error,not found\n ";

system("pause");

break;

}

A.crossout(j);

cout << "Element has been deleted" << endl;

system("pause");

break;

case 3:

cout << "List: \n";

A.print();

system("pause");

break;

case 4:

A.delete\_list();

cout << " The list has been deleted " << endl;

system("pause");

break;

case 5:

exit(1);

break;

default:

cout << "Error" << endl;

system("pause");

}

} while (1);

system("pause");

}

SkipList.h

#pragma once

#include <string>

#include <vector>

using namespace std;

template <class T>

struct Node {

int a;

T value;

vector<Node\*> first;

Node(int k, const T& c, int stage);

};

template <class T>

class SkipList

{

public:

SkipList();

~SkipList();

void print();

Node<T>\* find(int valk);

void input(int valk, T newValue);

void crossout(int valk);

void delete\_list();

private:

Node<T>\* head;

Node<T>\* null;

int randomLevel();

int createNode(const vector<Node<T>\*>& v);

Node<T>\* makenewNode(int k, T val, int stage);

float chance;

int maxLevel;

};

template <class T>

Node<T>::Node(int k, const T& v, int level)

: a(k), value(v)

{

for (int i = 0; i < level; ++i) first.emplace\_back(nullptr);

}

// constructor

template <class T>

SkipList<T>::SkipList() : chance(0.5), maxLevel(16)

{

int headk = numeric\_limits<int>::min();

head = new Node<T>(headk, 0, maxLevel);

int nullk = numeric\_limits<int>::max();

null = new Node<T>(nullk, 0, maxLevel);

for (size\_t i = 0; i < head->first.size(); ++i)

head->first[i] = null;

}

// destructor

template <class T>

SkipList<T>::~SkipList()

{

delete head;

delete null;

}

template <class T>

int SkipList<T>::randomLevel()

{

int v = 1;

while ((((double)std::rand() / RAND\_MAX)) < chance && abs(v) < maxLevel)

v += 1;

return abs(v);

}

template <class T>

int SkipList<T>::createNode(const vector<Node<T>\*>& v)

{

int curStage = 1;

int nilKey = numeric\_limits<int>::max();

if (v[0]->a == nilKey)

return curStage;

for (size\_t i = 0; i < v.size(); ++i)

if (v[i] != nullptr && v[i]->a != nilKey)

++curStage;

else

break;

return curStage;

}

template <class T>

Node<T>\* SkipList<T>::find(int valk)

{

Node<T>\* x = head;

int curMax = createNode(head->first);

for (int i = curMax; i-- > 0;)

while (x->first[i] != nullptr && x->first[i]->a < valk)

x = x->first[i];

x = x->first[0];

if (x->a == valk)

return x;

else

return nullptr;

}

template <class T>

void SkipList<T>::print()

{

Node<T>\* list = head;

while (list->first[0] != null)

{

cout << list->first[0]->a << ") " << list->first[0]->value << endl;

list = list->first[0];

}

}

template <class T>

Node<T>\* SkipList<T>::makenewNode(int key, T val, int level)

{

return new Node<T>(key, val, level);

}

template <class T>

void SkipList<T>::delete\_list()

{

delete head;

delete null;

SkipList();

}

template <class T>

void SkipList<T>::input(int valk, T newValue)

{

Node<T>\* x = nullptr;

x = find(valk);

if (x)

{

x->value = newValue;

return;

}

vector<Node<T>\*> update(head->first);

unsigned int currentMaximum = createNode(head->first);

x = head;

for (unsigned int i = currentMaximum; i-- > 0;)

{

while (x->first[i] != nullptr && x->first[i]->a < valk)

x = x->first[i];

update[i] = x;

}

x = x->first[0];

int newNodeLevel = 1;

if (x->a != valk)

{

newNodeLevel = randomLevel();

int currentLevel = createNode(update);

if (newNodeLevel > currentLevel)

for (int i = currentLevel + 1; i < newNodeLevel; ++i)

update[i] = head;

x = makenewNode(valk, newValue, newNodeLevel);

}

for (int i = 0; i < newNodeLevel; ++i)

{

x->first[i] = update[i]->first[i];

update[i]->first[i] = x;

}

}

template <class T>

void SkipList<T>::crossout(int searchKey)

{

vector<Node<T>\*> update(head->first);

Node<T>\* x = head;

unsigned int currentMaximum = createNode(head->first);

for (unsigned int i = currentMaximum; i-- > 0;)

{

while (x->first[i] != nullptr && x->first[i]->a < searchKey)

x = x->first[i];

update[i] = x;

}

x = x->first[0];

if (x->a == searchKey)

{

for (size\_t i = 0; i < update.size(); ++i)

{

if (update[i]->first[i] != x)

break;

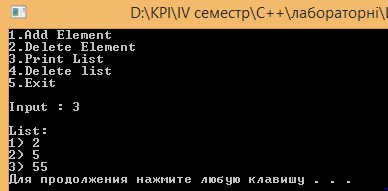
update[i]->first[i] = x->first[i];

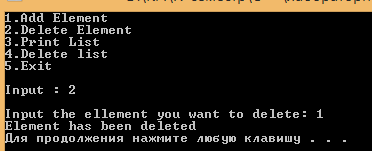
}

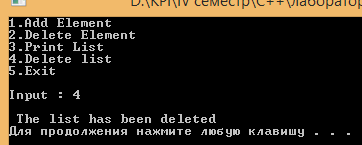
delete x;

}

}







UML-діаграма:



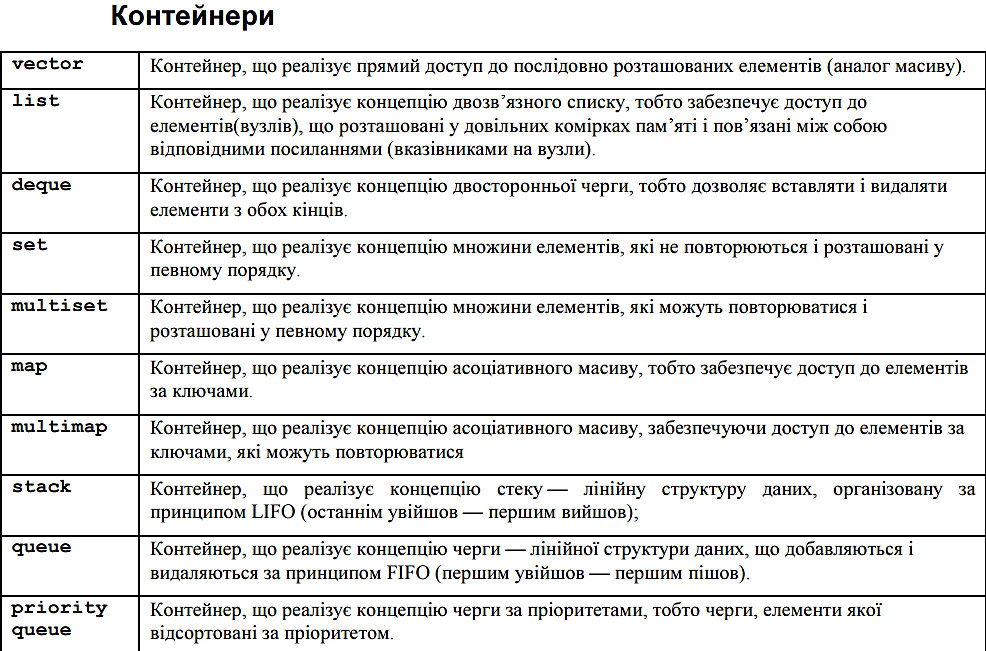
Контрольні питання:

1)Опишіть організацію контейнера STL.

Библиотека содержит пять основных видов компонентов:

* **алгоритм** (*algorithm*): определяет вычислительную процедуру.
* **контейнер** (*container*): управляет набором объектов в памяти.
* **итератор** (*iterator*): обеспечивает для алгоритма средство доступа к содержимому контейнера.
* **функциональный объект** (*function object*): инкапсулирует функцию в объекте для использования другими компонентами.
* **адаптер** (*adaptor*): адаптирует компонент для обеспечения различного интерфейса.

2)Які класи належать до категорії контейнерів?



3)Які вимоги до типів, які можна використовувати з контейнерами STL?

Все ссылочные типы, которые появляются в контейнерах STL должны иметь, как минимум, следующие элементы:

• Открытый копирующий конструктор

• Открытый оператор присваивания.

• Открытый деструктор.

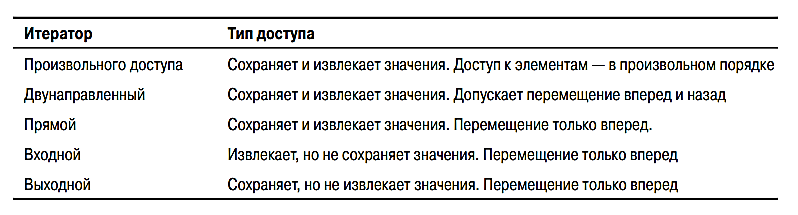
Кроме того, ассоциативные контейнеры, такие как set и map должны иметь открытый конкретный оператор сравнения, operator< по умолчанию. Некоторые операции в контейнерах могут также потребовать открытый конструктор по умолчанию и открытый оператор равенства.

4)Яка різниця між контейнерами та адаптерами контейнерів?

**Контейнеры** - это объекты, которые содержат другие объекты. Они управляют размещением в памяти и свобождением этих объектов через конструкторы, деструкторы, операции вставки и удаления.

**Адаптеры контейнеров** — это разновидность последовательного или ассоциативного контейнера, который обеспечивает ограниченные интерфейсы контейнеров. Библиотека предоставляет *stack*, *queue* и *priority\_queue* через адаптеры, которые могут работать с различными типами последовательностей. Контейнеры-адаптеры не поддерживают итераторы.

5)Що таке ітератори,яких типів вони бувають та в чому їх принципова різниця?

Итераторы — это объекты, которые ведут себя более или менее подобно указателям. Они предоставляют возможность выполнять циклическую обработку элементов контейнера — подобно тому, как используется указатель для организации цикла по массиву. Существует пять типов итераторов.

6)Які операції можна виконувати над усіма типами ітераторів?

Ко всем типам интераторов можна выполнять операции: присвоения, разименования и инкремента.

7)Що таке ітератор вільного доступу? Які операції він дозволяє виконувати?

Итераторы произвольного доступа (random access iterator) обладают всеми свойствами двунаправленнх итераторов, а также поддерживают операции сравнения и адресной арифметики, то есть непосредственный доступ по индексу.  
i += n, i + n, i -= n, i - n, i1 - i2, i[n], i1 < i2, i1 <= i2, i1 > i2, i1 >= i2  
  
8)Які вимоги до ітераторів з боку алгоритмів?

Разные виды литераторов обозначают разные требования к 5 видов литераторов: входной, выходной, однонаправленный, двунаправленный и свободного доступа. В каждом виде реализованы разные операции.   
  
9)Що таке стандартні алгоритми? Наведіть приклад.

В C++ существует множество спецальных алгоритмов для работы с массивами. Они позволяют сортировать данные в массиве, искать в нем какое-либо значение, менять элементы местами и т.д. Для их работы вам необходимо подкючить <algorithm> в начале программы.

Алгоритмы сортировки  
• reverse(start,end) - инвертирует элементы последовательности start-end (т.е. сортирует в обратном порядке. Если была последовательность 1 2 3, то в результате получим 3 2 1)  
• random\_shuffle(start,end) - сортирует элементы между start и end в случайном порядке  
• sort(start,end) - сортирует элементы от start до end в порядке возврастания.

Удаления элементов  
• remove(begin,end,what) - в промежутке begin-end удаляет все what  
• unique(begin,end) - удаляет все дубликаты

10) Які алгоритми вимагають впорядкованості?

Последовательности требуют алгоритмы бинарного поиска, lower\_bound ( находит первую позицию в диапазоне, перед которой можна вставить елемент), upper\_bound (находит первую позицию в диапазоне, перед которой можна вставить елемент ), equal\_range ( находит  поддиапазон в отсортированном диапазоне, в котором элемент можно вставить перед любым итератором в этом диапазоне без нарушения упорядоченности), set\_union (формирует множество, является объединением двух диапазонов), set\_intersection (формирует множество,является пересечением двух диапазонов), set\_difference (формирует множество, является разницей между двумя диапазонами), set\_symmetric\_difference (формирует последовательность, представляет собой симметричную разность двух диапазонов)