 МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Компьютерный практикум №6**

з дисципліни «Об'єктно-орієнтоване програмування»

# на тему: «STL»

Варіант №7

**Виконав:**

студент гр. БС-03

Затуловский Г. А.

**Перевірив:**

ас. каф. БМК Бабенко В.О.

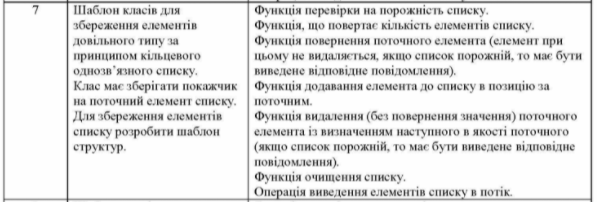
Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

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

                      (підпис викладача)

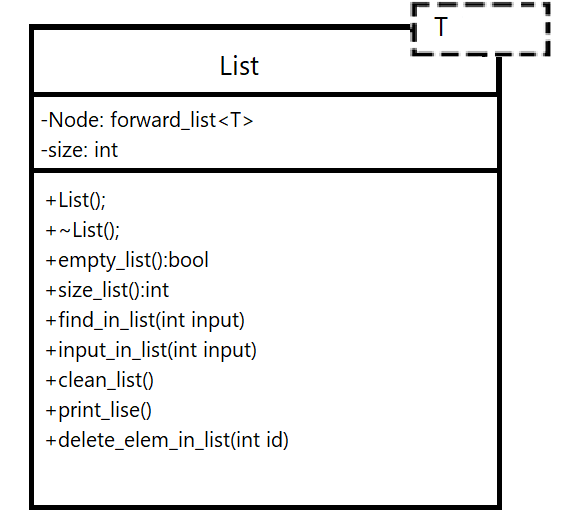
Київ-2022

**Завдання:**

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

:

**UML діаграма:**

****

**Код програми:**

**OOP.cpp**

#include "List.h"

#include <iostream>

#include <string>

using namespace std;

int main()

{

List<int> lst;

bool prog = true;

bool emp = false;

int choice = 1;

int input = 0;

cout << "1 - check for emptiness\n"

<< "2 - size list\n"

<< "3 - return the current element\n"

<< "4 - add new element\n"

<< "5 - delete element\n"

<< "6 - clean list\n"

<< "7 - print list\n"

<< "8 - exit\n" << endl;

while (prog)

{

cin >> choice;

switch (choice)

{

case 1:

emp = lst.empty\_list();

if (emp)

{

cout << "List is empty" << endl;

}

else

{

cout << "List is not empty" << endl;

}

break;

case 2:

cout << "List size: " << lst.size\_list() << endl;

break;

case 3:

cout << "Current index: ";

cin >> input;

lst.find\_in\_list(input);

break;

case 4:

cout << "Input data: ";

cin >> input;

lst.input\_in\_list(input);

break;

case 5:

cout << "Input index of element wich you want delete: ";

cin >> input;

lst.delete\_elem\_in\_list(input);

break;

case 6:

lst.clean\_list();

cout << "List cleared" << endl;

break;

case 7:

lst.print\_lise();

break;

case 8:

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

prog = false;

break;

default:

cout << "Anoun command" << endl;

break;

}

}

return 0;

}

**List.h**

#pragma once

#include <iostream>

#include <string>

#include <forward\_list>

#include <iterator>

using namespace std;

template<typename T>

class List

{

private:

forward\_list<T> Node;

int size;

public:

List();

~List();

bool empty\_list();

int size\_list();

void find\_in\_list(int input);

void input\_in\_list(int input);

void clean\_list();

void print\_lise();

void delete\_elem\_in\_list(int id);

};

template<typename T>

inline List<T>::~List()

{

}

template<typename T>

inline bool List<T>::empty\_list()

{

if (Node.empty() == false)

{

return false;

}

else

{

return true;

}

}

template<typename T>

inline int List<T>::size\_list()

{

return size;

}

template<typename T>

inline void List<T>::find\_in\_list(int input)

{

if (size > input)

{

forward\_list<int>::iterator it = Node.begin();

for (int i = 0; i < input; i++, it++)

{

if (i == input)

{

cout << "Current element: " << \*it << endl;

}

}

}

}

template<typename T>

inline void List<T>::input\_in\_list(int input)

{

typename forward\_list<T>::iterator it = Node.begin();

if (size == 0)

{

Node.push\_front(input);

}

else

{

for (int i = 0; i <= size; i++, it++)

{

if (size-1 == i)

{

Node.emplace\_after(it, input);

}

}

}

size++;

}

template<typename T>

inline void List<T>::clean\_list()

{

Node.clear();

size = 0;

}

template<typename T>

inline void List<T>::print\_lise()

{

for (auto elem : Node)

{

cout << elem << " -> ";

}

cout << "NULL" << endl;

}

template<typename T>

inline void List<T>::delete\_elem\_in\_list(int id)

{

if (size >= 1 && id == 0)

{

Node.pop\_front();

size--;

}

if (id < size && id != 0)

{

typename forward\_list<T>::iterator it = Node.begin();

for (int i = 0; i < id; i++, it++)

{

if (id - 1 == i)

{

Node.erase\_after(it);

size--;

}

}

}

}

template<typename T>

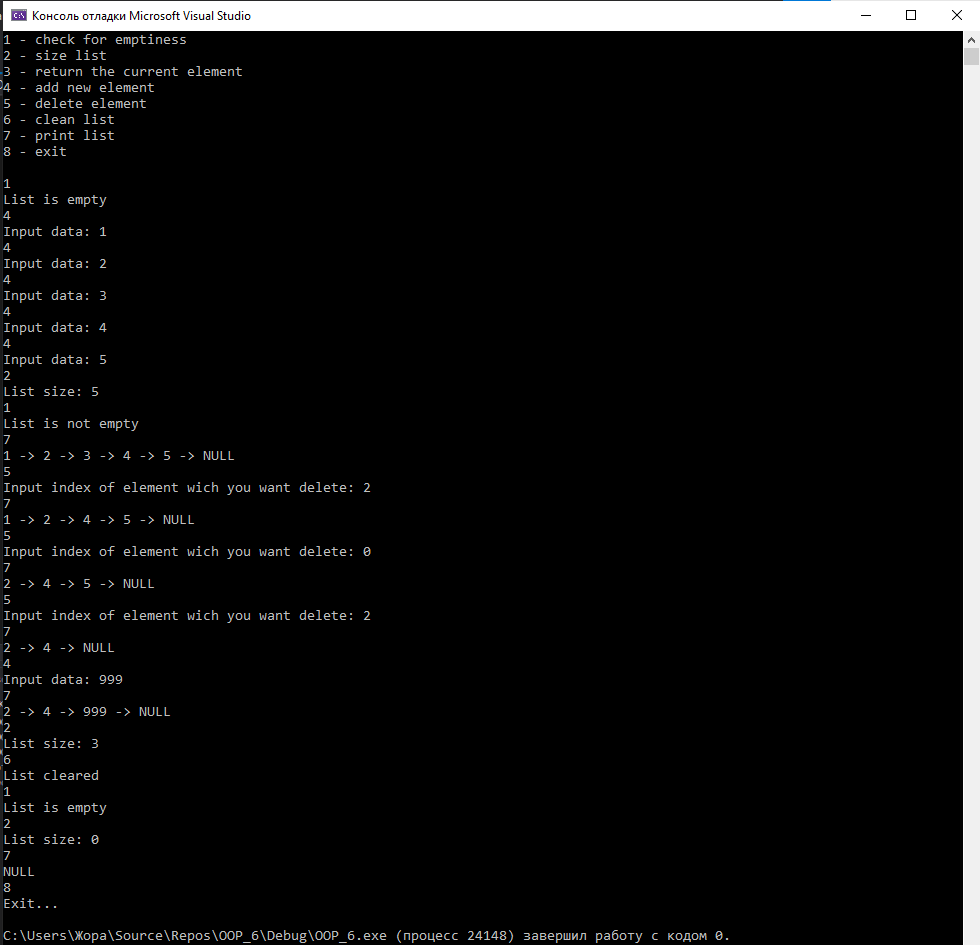
inline List<T>::List()

{

size = 0;

}

**Результати:**



**Контрольні запитання**

**1. Опишіть організацію контейнера STL?**

STL – це набір узгоджених загальних алгоритмів, контейнерів, засобів доступу до їх вмісту та різних допоміжних функцій в С++. В STL виділяють декілька основних компонентів: контейнер, ітератор, алгоритм, адаптер, функціональний об’єкт

**2. Які класи належать до категорії контейнерів?**

Концепцію контейнера реалізують класи vector, list, deque, set, multiset, map, multimap, stack, queue і priority queue, bitset, basic\_string, valarray

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

Всі типи даних можна використовувати з контейнерами STL.

**4. Яка різниця між контейнерами та адаптерами контейнерів?**

Адаптери контейнерів надають обмежений інтерфейс до контейнера. Зокрема, вони не надають ітераторів. Стандартні контейнери зберігаються як елементи даних у класах-адаптерах в розділі private

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

Ітератор – це об'єкт, що дозволяє програмісту перебирати всі елементи колекції без урахування особливостей її реалізації.. Ітератори бувають таких типів:

1. Вхідні – забезпечують доступ для читання в одному напрямку, дозволяють виконувати операції присвоювання або копіювання.

2. Вихідні – забезпечують доступ для запису в одному напрямку, їх не можна порівнювати на рівність.

3. Однонапрямлені – забезпечують доступ для читання та запису в одному напрямку, володіють властивостями вхідних ітераторів, і на відміну від вихідних ітераторів, їх можна порівнювати на рівність.

4. Двонапрямлені – володіють всіма функціями однонапрямлених ітераторів, на додачу вони дозволяють переходити до попереднього елементу.

5. Довільного доступу – еквівалентний звичайному вказівнику, підтримує арифметику вказівників, синтаксис індексації масивів і всі форми порівняння.

**6. Які операції можна виконувати над усіма типами літераторів?**

Оскільки ітератор є аналогом вказівника, до нього можна застосовувати ті ж самі операції, що і до звичайного вказівника: розіменування, інкремента, декремента і порівняння

**7. Що таке ітератор довільного доступу? Які операції він дозволяє виконувати?**

Ітератори довільного доступу – найбільш умілі з основних ітераторів. Вони не тільки реалізують всі функції, властиві ітераторам нижчого рівня, але і володіють великими можливостями, такими як: арифметика вказівників, синтаксис індексації масивів та всі форми порівняння.

**8. Які вимоги до ітераторів з боку алгоритмів?**

Ітератори не повинні виконувати одну й ту саму дію

**9. Що таке стандартні алгоритми? Наведіть приклад.**

Стандартні алгоритми – це алгоритми, що визначені в STL. До них відносяться алгоритми пошуку, сортування, видалення елементів, операції відношення, кучі, злиття.

Наприклад:

#include <iostream>

#include <vector>

#include <algorithm>

int main()

{

    std::vector<int> vect;

    vect.push\_back(4);

    vect.push\_back(8);

    vect.push\_back(-3);

    vect.push\_back(3);

    vect.push\_back(-8);

    vect.push\_back(12);

    vect.push\_back(5);

    std::sort(vect.begin(), vect.end());

    std::vector<int>::const\_iterator it;

    for (it = vect.begin(); it != vect.end(); ++it)

        std::cout << \*it << ' ';

    std::cout << '\n';

    std::reverse(vect.begin(), vect.end());

    for (it = vect.begin(); it != vect.end(); ++it)

        std::cout << \*it << ' ';

    std::cout << '\n';

}

Результат:

-8 -3 3 4 5 8 12  
12 8 5 4 3 -3 -8

**10. Які алгоритми вимагають впорядкованості?**

До впорядкованих алгоритмів можна віднести алогоритми бінарного пошуку lower\_bound, upper\_bound, equal\_range, set\_union, set\_intersection, set\_difference, set\_symmetric\_difference, merge.