МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №5**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Шаблони»

Варіант №5

**Виконав:**

студент гр. БС-71

Воробйов І.О.

**Перевірив:**

асистент каф. БМК

Рисін С.В.

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

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

(підпис викладача)

Київ-2019

**🞏 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**

**🞏 потужність**

**🞏 інші зауваження:**

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

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

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 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:**

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

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

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

**Завдання:**

1. Ознайомитися з теоретичними відомостями із шаблонами функцій та шаблонами класів.
2. Відповідно до свого варіанту розробити шаблон класів абстрактної структури даних із визначенням заданих функцій та операцій (в кожному шаблонному класі також має бути визначений конструктор копіювання та операції привласнення) і відповідну діаграму класів в нотації UML, написати програму тестування, в якій перевіряється використання шаблона для стандартних типів даних:

|  |  |  |
| --- | --- | --- |
| Номер варіанту | Шаблон класів | Функції та операції |
| 5 | Шаблон класів множини для збереження унікальних елементів довільного типу (для типу елементів має бути визначена перевірка на рівність). Розмір множини задається як аргумент конструктора. | Функція перевірки на порожність множини.  Функція, що повертає кількість елементів множини.  Функція пошуку елемента (якщо елемент відсутній, то має бути відповідне повідомлення)  Функція додавання елемента (якщо елемент, який додається, вже присутній в множині, то має бути виведене відповідне повідомлення)  Функція видалення (без повернення значення) елемента (якщо множина порожня або такий елемент відсутній, то мають бути виведені відповідні повідомлення).  Операція об’єднання (+) та перетину (\*) двох множин.  Операція виведення елементів множини в потік. |

1. Скласти і захистити звіт по роботі.

**UML діаграма**



**Код програми:**

**plural.h:**

#pragma once

#include <iostream>

using namespace std;

template<typename T>

class plural

{

public:

T elem;

plural\* r;

plural() {};

plural(int s)

{

plural<int>\* tmp = this;

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

{

cin >> tmp->elem;

if (i == s - 1)

tmp->r = nullptr;

else

tmp->r = new plural<T>;

tmp = tmp->r;

}

}

plural(plural \*data)

{

plural<int>\* tmp = this;

while (data != nullptr)

{

tmp = new plural<T>;

tmp->elem = data->elem;

data = data->r;

if (data!=nullptr)

tmp->r = nullptr;

else

tmp->r = new plural<T>;

tmp = tmp->r;

}

}

bool is\_empty()

{

if (this == nullptr)

return true;

else

return false;

}

int size()

{

int count = 0;

plural<T>\* tmp = this;

while (tmp != nullptr)

{

count++;

tmp = tmp->r;

}

return count;

}

bool find(T tmp\_elem)

{

plural<T>\* tmp = this;

while (tmp != nullptr)

{

if (tmp->elem == tmp\_elem)

return true;

tmp = tmp->r;

}

return false;

}

void add(T data)

{

plural<T>\* tmp = this;

while (true)

{

if (tmp->elem == data)

{

cout << "Elem inside." << endl;

return;

}

if (tmp->r == nullptr)

{

tmp->r = new plural<T>;

tmp = tmp->r;

break;

}

tmp = tmp->r;

}

tmp->elem = data;

tmp->r = nullptr;

}

void del(T data)

{

plural<T>\* tmp = this;

if (tmp->elem == data)

{

plural<T>\* tmp1 = tmp->r;

delete tmp;

tmp = tmp1;

return;

}

while (tmp->r != nullptr)

{

if (tmp->r->elem == data)

{

plural<T>\* tmp\_del = tmp->r;

tmp->r = tmp->r->r;

delete tmp\_del;

cout << "Deleted." << endl;

return;

}

tmp = tmp->r;

}

cout << "Elem not find." << endl;

}

template<typename T>

friend ostream& operator <<(ostream& cout, plural<T>& obj)

{

plural<T>\* tmp = obj.r;

cout << obj.elem << " ";

while (tmp != nullptr)

{

cout << tmp->elem << " ";

tmp = tmp->r;

}

cout << endl;

return cout;

}

template<typename T>

friend plural<T> operator + (plural<T>& obj, plural<T>& obj1)

{

plural<T> tmp = plural<T>(obj);

plural<T>\* tmp\_1 = new plural<T>(obj1);

while (tmp\_1 != nullptr)

{

tmp.add(tmp\_1->elem);

tmp\_1 = tmp\_1->r;

}

return tmp;

}

template<typename T>

friend plural<T> operator \* (plural<T>& obj, plural<T>& obj1)

{

plural<T> tmp\_plur = plural<T>(obj);

plural<T> tmp = tmp\_plur;

while (true)

{

plural<T>\* tmp\_1 = new plural<T>(obj1);

while (true)

{

if (tmp\_1->elem == tmp.elem)

break;

if (tmp\_1->r == nullptr)

{

tmp.del(tmp.elem);

break;

}

tmp\_1 = tmp\_1->r;

}

if (tmp.r == nullptr)

break;

tmp = tmp.r;

}

return tmp\_plur;

}

};

**Source.cpp:**

#include "plural.h"

int main()

{

int n;

cout << "Input n: ";

cin >> n;

plural<int>\* arr = new plural<int>(n);

int k = 1;

while (k != 0)

{

cout << "1-Check for empty." << endl;

cout << "2-Get size." << endl;

cout << "3-Find elem." << endl;

cout << "4-Add elem." << endl;

cout << "5-Del elem." << endl;

cout << "6-Union plural." << endl;

cout << "7-Intersection plural." << endl;

cout << "8-Print." << endl;

cout << "0-Exit" << endl;

cout << ">";

cin >> k;

switch (k)

{

case 1:

{

if (arr->is\_empty())

cout << "Plural empty!" << endl;

else

cout << "Plural not empty!" << endl;

break;

}

case 2:

{

cout << "Numb of elem: " << arr->size() << endl;

break;

}

case 3:

{

int value;

cout << "Input value:";

cin >> value;

if (arr->find(value))

cout << "Elem inside." << endl;

else

cout << "Elem not find." << endl;

break;

}

case 4:

{

int value;

cout << "Input value:";

cin >> value;

arr->add(value);

break;

}

case 5:

{

int value;

cout << "Input value:";

cin >> value;

arr->del(value);

break;

}

case 6:

{

int n1;

cout << "Input n: ";

cin >> n1;

plural<int>\* arr1 = new plural<int>(n1);

plural<int> arr2= \*arr + \*arr1;

cout << arr2;

break;

}

case 7:

{

int n1;

cout << "Input n: ";

cin >> n1;

plural<int>\* arr1 = new plural<int>(n1);

plural<int> arr2 = \*arr \* \*arr1;

cout << arr2;

break;

}

case 8:

{

cout << \*arr;

}

}

system("pause");

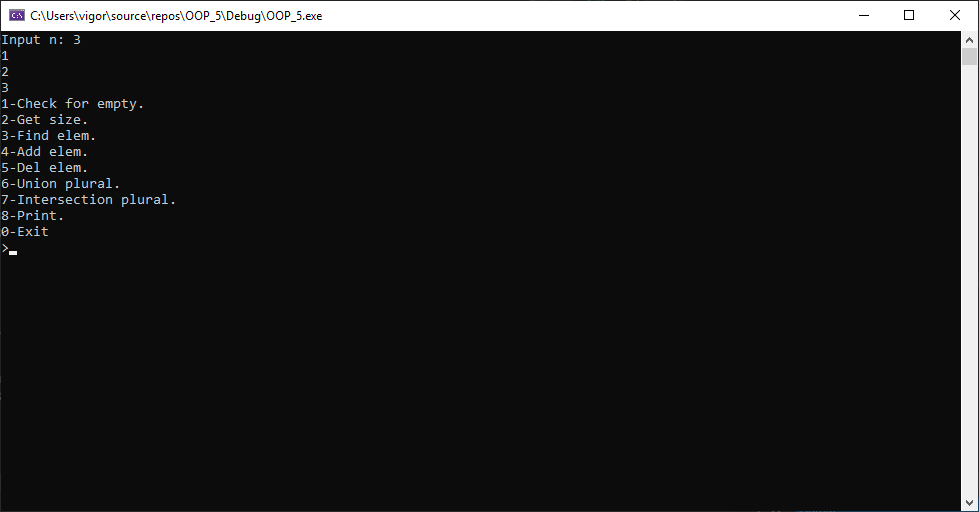
system("cls");

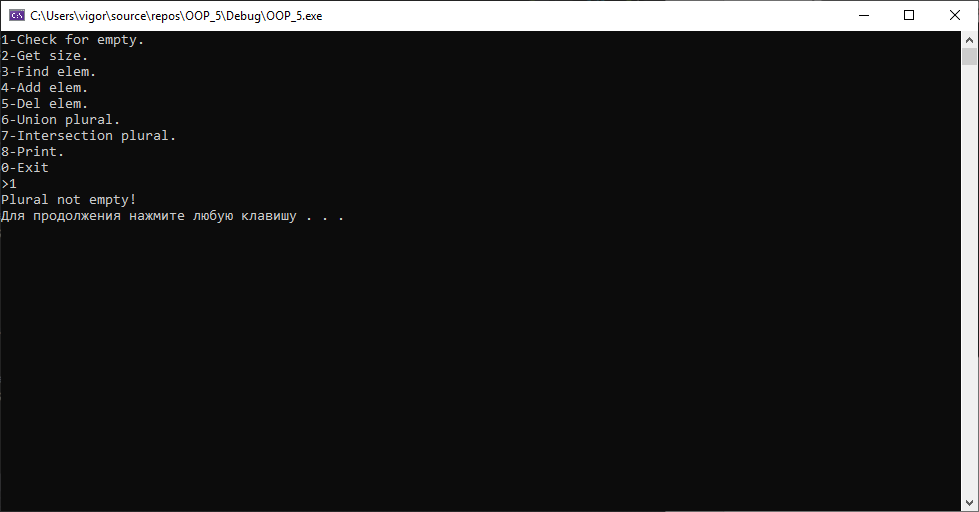
}

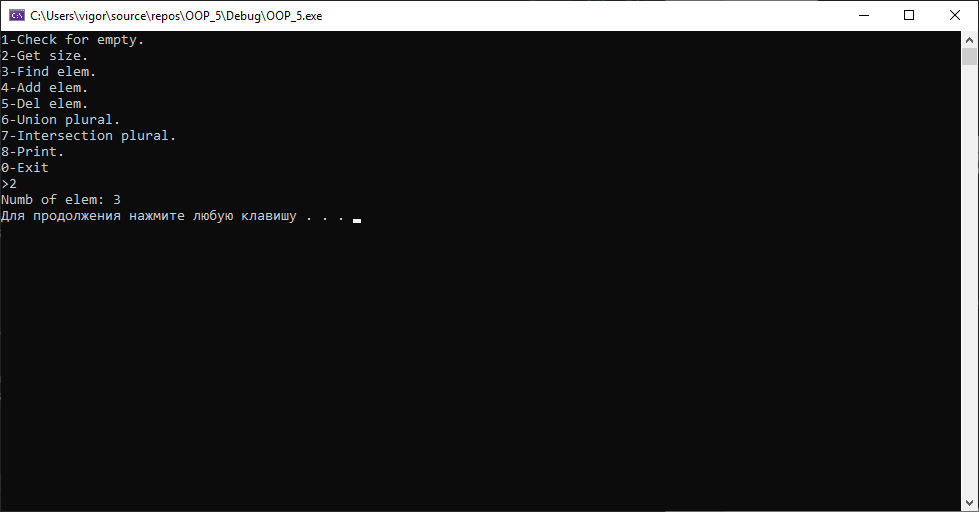
return 0;

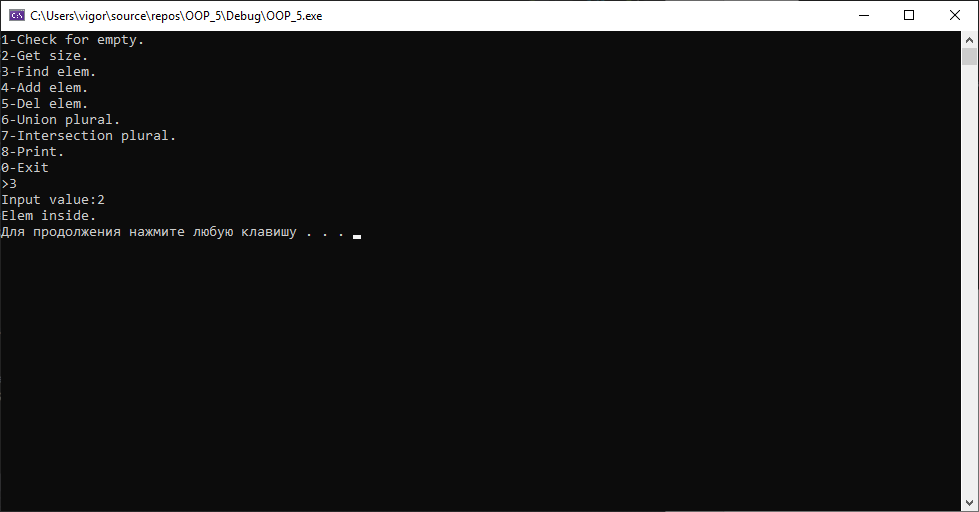
}

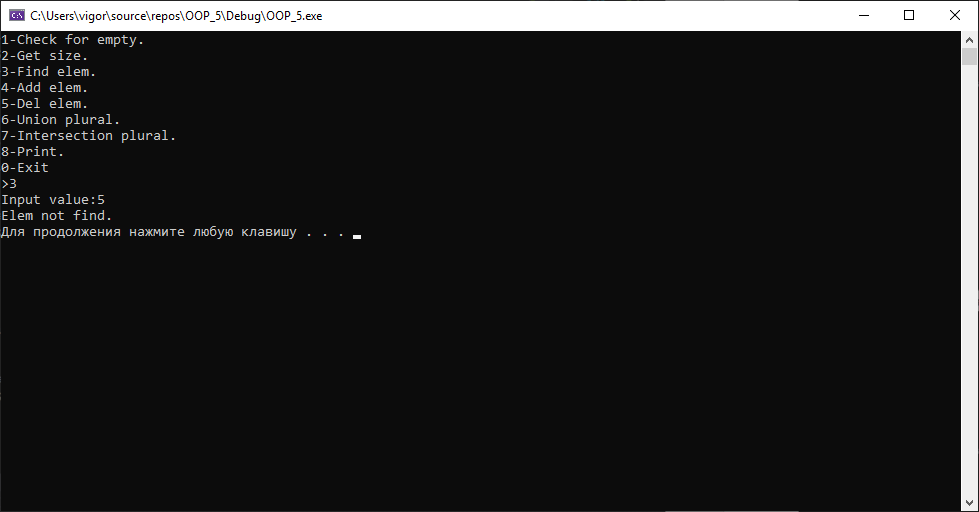
**Результат роботи програми:**

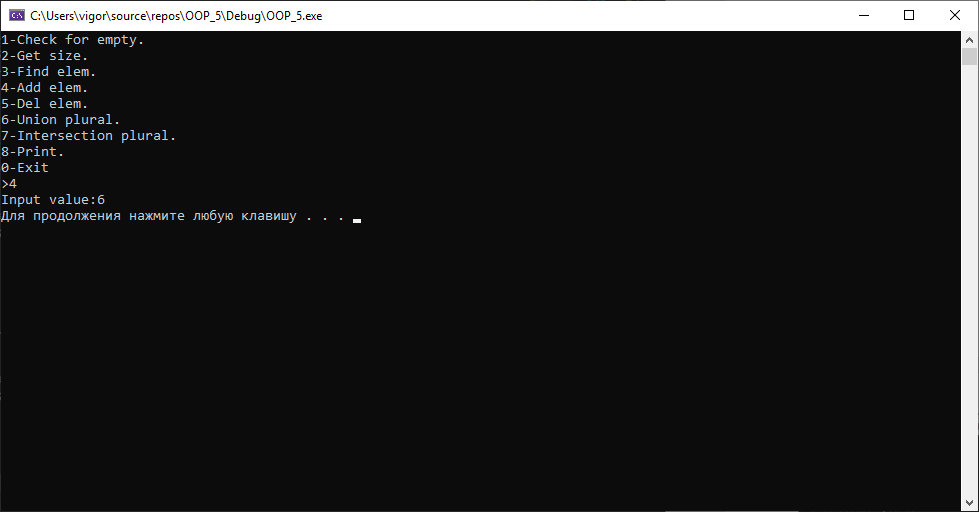


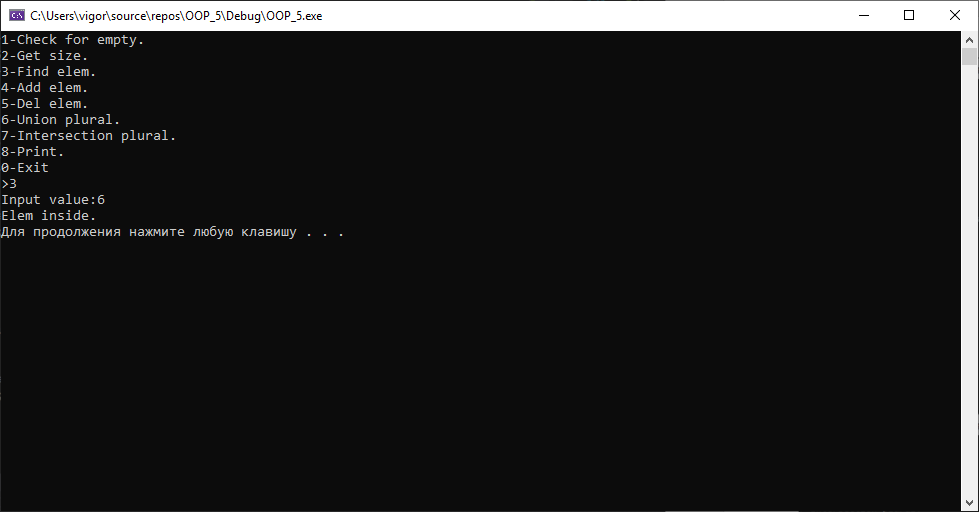


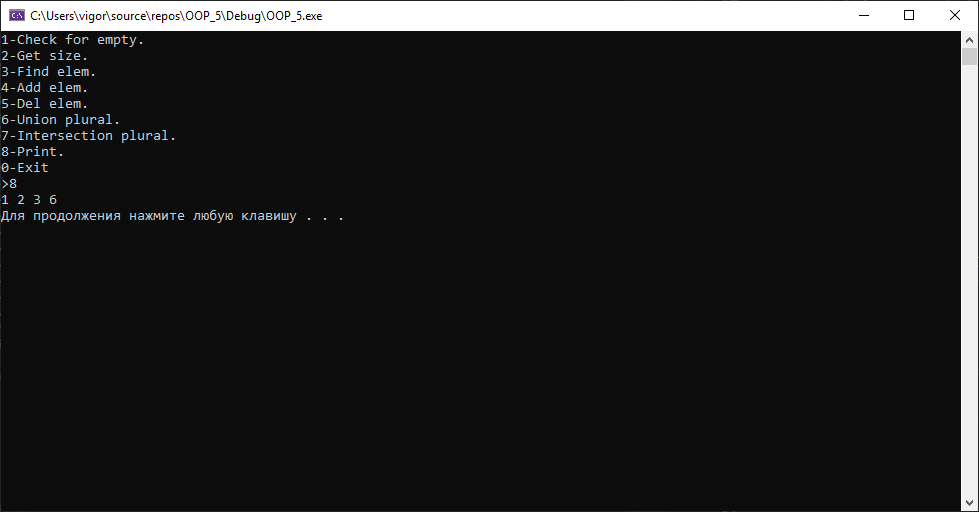


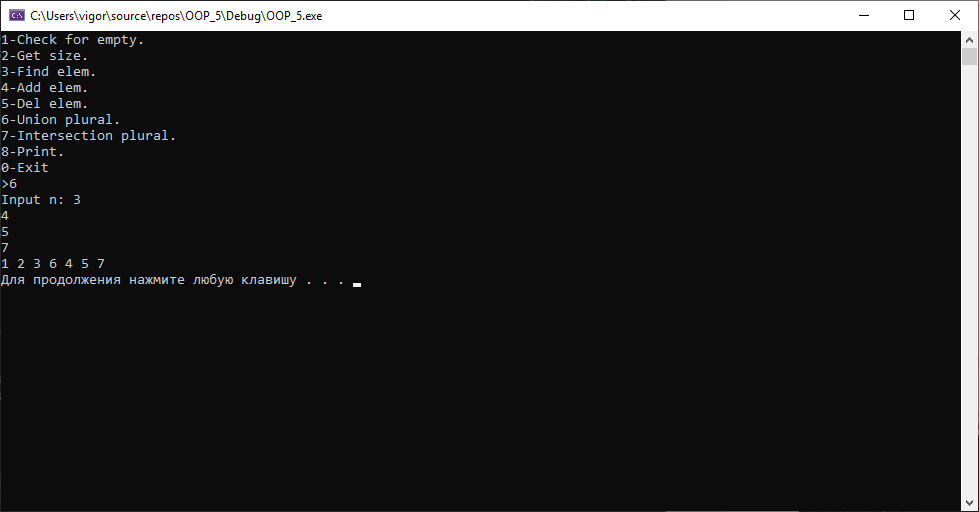












**Контрольні запитання**

1. Для чого використовують шаблони?

Шаблони використовують для кодування узагальнених алгоритмів без прив’язки до деяких параметрів(напр.: тип даних)

1. Що таке шаблон класів, як його визначити в С++?

Шаблони класів - це такі класи з універсальним типом даних, який використовують, щоб для кожного можливого значення переданих параметрів створити окремий клас.

template <class T, …class Tn>

class Name

{

//визначення класу

};

1. Що таке шаблон функцій, як його визначити в С++?

Шаблони функцій – це функції, які дозволяють використовувати одну функцію для різних типів даних.

tenplate <class T, … class Tn>

return\_type func\_name(список параметрів)

{

//тіло функції

}

1. Що таке таке конкретизація шаблона? В яких випадках необхідне виконнання явної конкретизації шаблона?

При конкретизації шаблону замість параметра-типу підставляється фактичний вбудований або визначений користувачем тип. Конкретизація використовується, коли компілятор не може розпізнати тип, для якого він має конкретизувати функцію.

1. Що таке спеціалізація шаблона, коли її необхідно виконувати?

Спеціалізація шаблону – це перевизначення шаблону для перного типу. Її використовують тоді, коли потрібно вказати особливу поведінку шаблону для певного типу.

1. Для чого використовують параметри за замовченням для шаблону класів?

Якщо при конкретизації об’єкта не буде вказано параметра, будуть використовуватися параметри по замовчуванню. Когда какое то значение спользуется достоточно часто по сравнению с другим его можно использівать как значение по умолчанию.

1. Чи можна викликати параметризовану функцію без параметрів?

Так, якщо у всіх параматрах є значенння по замовчувані.

1. Чи може бути порожнім список параметрів шаблона? Відповід пояснити.

Список параметрів шаблону функції не може бути пустим, так як при цьому втрачається можливість параметризації і шаблон функцій стає звичайним визначенням конкретної функції.

1. Чи можна за допомогою шаблона створити функцію з таким самим ім’ям, як і в явно визначеної функції? Відповідь пояснити.

Можна. Явно визначена функція з таким самим ім’ям, як і в шаблона функції, буде спеціалізаціює шаблона.

1. Чи можуть шаблони класів містити віртуальні функції?

Ні