МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННО БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Учебный Центр Информационных Технологий «Информатика»

Лабораторная работа № 4
по дисциплине «Объектно-ориентированное программирование»

Выполнил слушатель: Пешков Е.В.
Вариант: 9
Дата сдачи:
Преподаватель: Васюткина И.А.

Новосибирск, 2019г.

Задание:

Реализовать с помощью шаблонов классов динамическую списочную структуру, содержащую объекты классов, связанных наследованием из лаб.работы 3. Добавить в классы виртуальные методы для изучения свойства ООП полиморфизм(методы ввода объектов с клавиату - ры, вывода на экран, рассчетов и т.д.).

В шаблонах реализовать методы добавления, удаления, вставки по номеру, удаления по номеру, поиск и просмотр всей структуры.

Изменить демонстрационную программу так, чтобы она демонстрировала полиморфическое поведение классов.

Исследовать, как реализуется механизм полиморфизма.

Протестировать работу шаблонов на встроенных типах языка C++ (например, int, float).

Вариант 9

Структура данных : односвязный список.

Объект хранения : указатель на объект.

1. Постановка задачи.

В соответствии с заданием необходимо построить шаблон класса односвязного списка для работы с комплексными числами. Для этих целей в класс Complex добавлены виртуальные методы для ввода и вывода данных.

```
virtual void Show(ostream& os);
virtual void Input(istream& is);

void Complex::Show(ostream& os)
{
    os << "Complex " << *this << endl;
}
void Complex::Input(istream& is)
{
    is >> *this;
}
```

Поля класса шаблона:

- int Size;– размер списка
- Node<T>* head – указатель на первый элемент списка,

Конструкторы:

- List();– конструктор без параметров

Деструктор:

- ~List();– деструктор объектов класса без параметров

Методы класса:

- void pop_back(); - удаление последнего элемента
- void removeAt(int); - удаление по индексу
- void insertAt(T* data, int index); - вставка по индексу
- void push_front(T* data); - добавление в начало списка
- void clear(); - удаление всех элементов списка
- void pop_front(); - удаление первого элемента списка
- void push_back(T* data); - добавление в конец списка

- int GetSize(); - получить количество элементов списка
- void show(); - просмотр всей структуры
- void input(int n); - изменение элемента по индексу

2. Определение пользовательского класса.

Шаблон List

```
#include <string>
#include <iostream>
using namespace std;

template <typename T>
class List
{

public:

   List();
   ~List();

   // Методы

   void push_back(T* data); // добавление в конец списка
   void pop_back();// удаление последнего элемента
   void push_front(T* data); // добавление в начало списка
   void pop_front(); // удаление первого элемента списка
   void removeAt(int); // удаление по индексу
   void insertAt(T* data, int index); // вставка по индексу
   void clear(); // удаление всех элементов списка
   int GetSize(); // количество элементов
   void show(); // просмотр всей структуры
   void input(int n); // изменение элемента по индексу

private:

   template <typename T>
   class Node
   {
   public:
      Node* pNext;
      T* data;

      Node(T* data = nullptr, Node* pNext = nullptr)
      {
         this->data = data;
         this->pNext = pNext;
      }
   };
   int Size;
   Node<T>* head;
};
```

// **Конструктор без параметров**
List<T>::List()
Параметры:
Возвращаемый результат: ничего не возвращает.
Принцип работы: создает объект класса List.

// **Деструктор**
List<T>::~List()
Параметры:
Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет объект класса List.

// **Метод добавления в начало списка**
template<typename T>
void List<T>::push_front(T* data)
Параметры:

T* data – указатель на добавляемые данные.
Возвращаемый результат: ничего не возвращает.
Принцип работы: добавляет элемент в начало списка.

// **Метод удаления всех элементов списка**
template<typename T>
void List<T>::clear()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет все элементы списка.

// **Метод удаления последнего элемента**
template<typename T>
void List<T>::pop_back()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет последний элемент списка.

// **Метод удаления по индексу**
template<typename T>
void List<T>::removeAt(int index)
Параметры:

int index – индекс элемента.
Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет элемент по заданному индексу .

// Метод вставки элемента по индексу
template<typename T>
void List<T>::insertAt(T* value, int index)
Параметры:

int index – индекс элемента.
T* value – указатель на объект хранения
Возвращаемый результат: ничего не возвращает.
Принцип работы: вставляет элемент по заданному индексу .

// Метод удаления первого элемента
template<typename T>
void List<T>::pop_front()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: удаляет первый элемент списка .

// Метод добавления элемента в конец списка
template<typename T>
void List<T>::pop_front()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: добавляет элемент в конец списка .

// Метод доступа к полю Size
template<typename T>
int List<T>::GetSize()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: возвращает значение поля Size .

// Метод просмотра всего списка
template<typename T>
void List<T>::show()
Параметры:

Возвращаемый результат: ничего не возвращает.
Принцип работы: печатает в консоль все элементы списка .

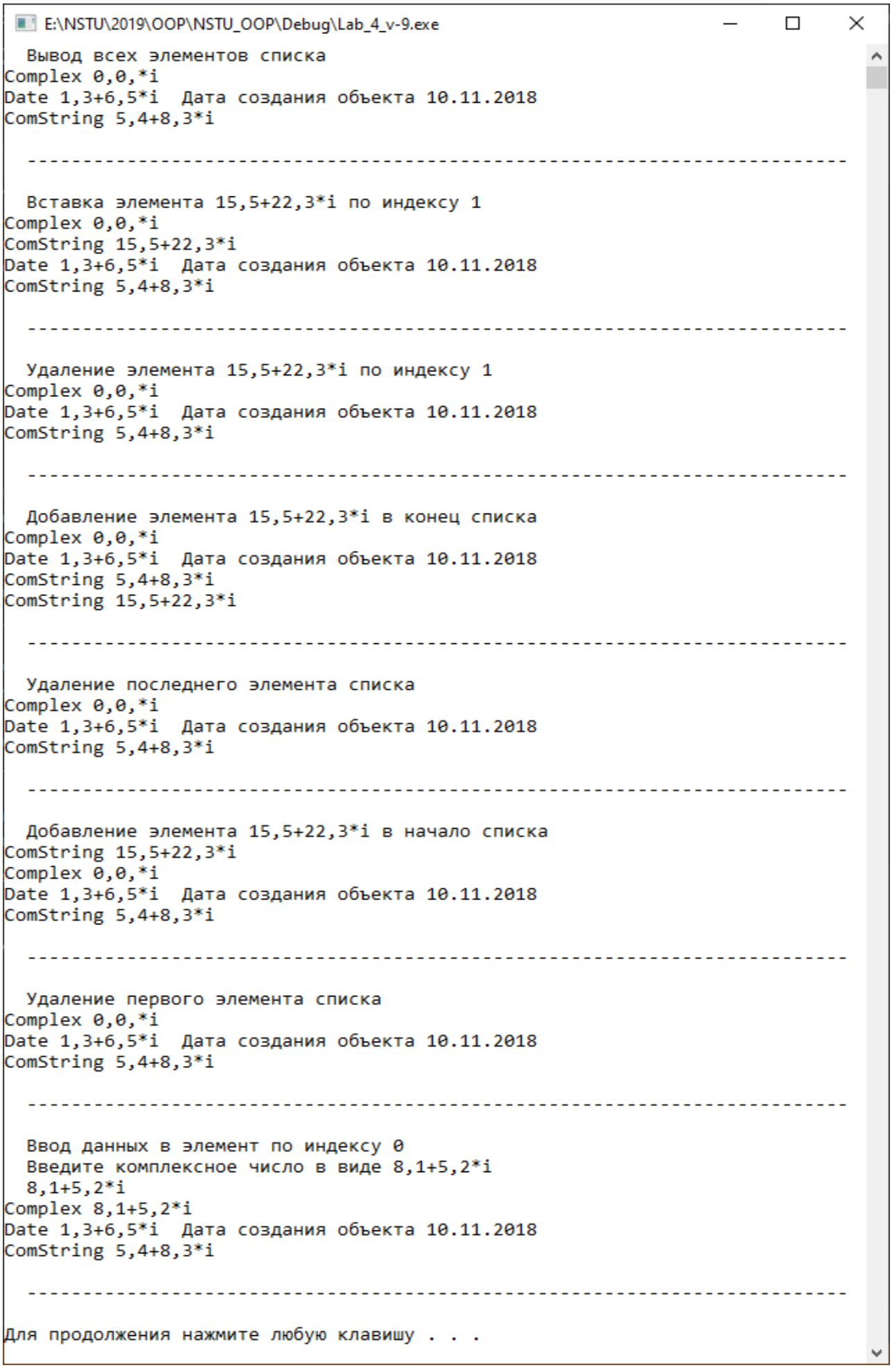
// Метод изменения элемента по индексу
template<typename T>
inline void List<T>::input(int n)
Параметры:
int n – индекс элемента
Возвращаемый результат: ничего не возвращает.
Принцип работы: изменяет элемент по заданному индексу .

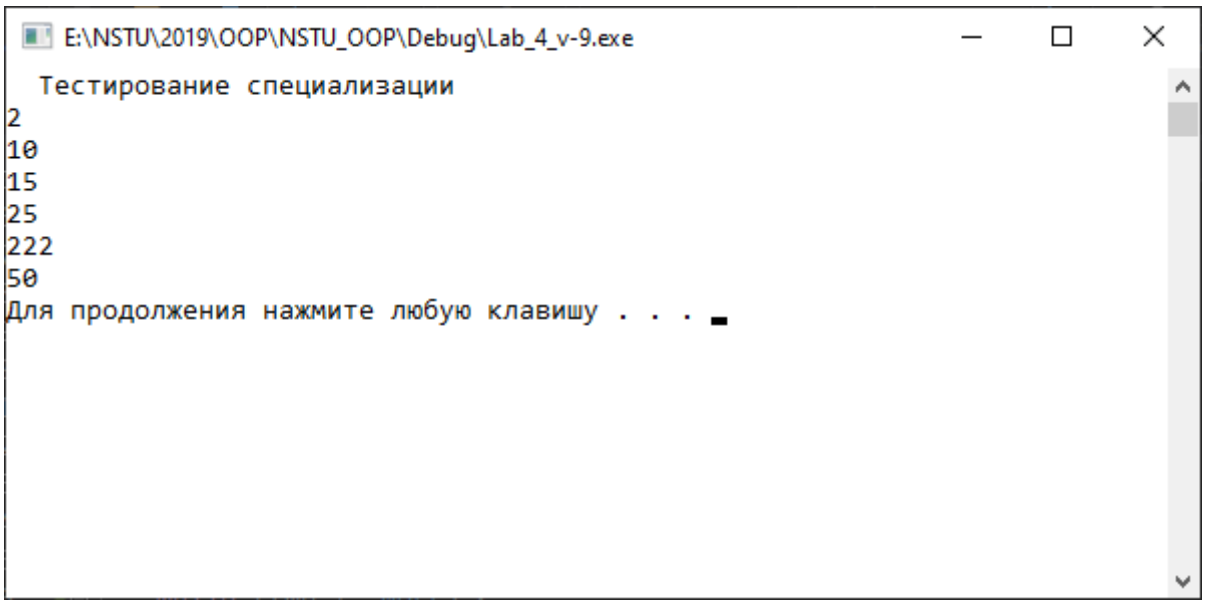
3. Листинг тестовой программы.

```
using namespace std;

#include <iostream>
#include "complex.h"
#include "com_date.h"
#include "com_string.h"
#include "my_list.h"

void print_line();

int main()
{
    setlocale(LC_ALL, "Russian");

      Complex* a = new Complex();
    Complex* b = new  Date(1.3, 6.5, 10, 11, 2018);
    Complex* c = new ComString("5,4+8,3*i");
    Complex* d = new ComString("15,5+22,3*i");

    List<Complex> list_1;

    list_1.push_back(a);
    list_1.push_back(b);
    list_1.push_back(c);

    cout << "  Вывод всех элементов списка" << endl;
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Вставка элемента 15,5+22,3*i по индексу 1 " << endl;
    list_1.insertAt(d, 1);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление элемента 15,5+22,3*i по индексу 1 " << endl;
    list_1.removeAt(1);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Добавление элемента 15,5+22,3*i в конец списка " << endl;
    list_1.push_back(d);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление последнего элемента списка " << endl;
    list_1.pop_back();
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Добавление элемента 15,5+22,3*i в начало списка " << endl;
    list_1.push_front(d);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление первого элемента списка " << endl;
    list_1.pop_front();
    list_1.show();
    print_line();
```

```
    cout << endl;

    cout << "  Ввод данных в элемент по индексу 0 " << endl;
    list_1.input(0);
    list_1.show();
    print_line();
    cout << endl;

    system("pause");
    return 0;
}

void print_line()
{
    cout << "\n  ";
    for (int i = 0; i < 74; i++)
    {
        cout << "-";
    }
    cout << "\n";
}
```

4. Пример работы программы.





Приложение. Текст программы

Файл main.cpp

```
using namespace std;

#include <iostream>
#include "complex.h"
#include "com_date.h"
#include "com_string.h"
#include "my_list.h"

void print_line();

int main()
{
    setlocale(LC_ALL, "Russian");

       Complex* a = new Complex();
    Complex* b = new  Date(1.3, 6.5, 10, 11, 2018);
    Complex* c = new ComString("5,4+8,3*i");
    Complex* d = new ComString("15,5+22,3*i");

    List<Complex> list_1;

    list_1.push_back(a);
    list_1.push_back(b);
    list_1.push_back(c);

    cout << "  Вывод всех элементов списка" << endl;
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Вставка элемента 15,5+22,3*i по индексу 1 " << endl;
    list_1.insertAt(d, 1);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление элемента 15,5+22,3*i по индексу 1 " << endl;
    list_1.removeAt(1);
    list_1.show();
    print_line();
    cout << endl;
```

```
    cout << "  Добавление элемента 15,5+22,3*i в конец списка " << endl;
    list_1.push_back(d);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление последнего элемента списка " << endl;
    list_1.pop_back();
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Добавление элемента 15,5+22,3*i в начало списка " << endl;
    list_1.push_front(d);
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Удаление первого элемента списка " << endl;
    list_1.pop_front();
    list_1.show();
    print_line();
    cout << endl;

    cout << "  Ввод данных в элемент по индексу 0 " << endl;
    list_1.input(0);
    list_1.show();
    print_line();
    cout << endl;

    system("pause");
    system("cls");

    List<int> list_2;
    cout << "  Тестирование специализации " << endl;
    list_2.push_back(new int(2));
    list_2.push_back(new int(10));
    list_2.push_back(new int(15));
    list_2.push_back(new int(25));
    list_2.push_back(new int(222));
    list_2.push_back(new int(50));
    list_2.show();

    system("pause");
    return 0;
}

void print_line()
{
    cout << "\n  ";
    for (int i = 0; i < 74; i++)
    {
        cout << "-";
    }
    cout << "\n";
}
```

Файл my_list.cpp

```
#include <string>
#include <iostream>
using namespace std;
template <typename T>
class List
{

public:

    List();
    ~List();

    // Методы
    void push_back(T* data); // добавление в конец списка
    void pop_back();// удаление последнего элемента

    void push_front(T* data); // добавление в начало списка
    void pop_front(); // удаление первого элемента списка

    void removeAt(int); // удаление по индексу
    void insertAt(T* data, int index); // вставка по индексу

    void clear(); // удаление всех элементов списка
    int GetSize(); // количество элементов

    void show(); // просмотр всей структуры
    void input(int n); // изменение элемента по индексу

    // Перегрузка операторов
    //T& operator[](const int index);

private:

    template <typename T>
    class Node
    {
    public:
        Node* pNext;
        T* data;

        Node(T* data = nullptr, Node* pNext = nullptr)
        {
            this->data = data;
            this->pNext = pNext;
        }
    };
    int Size;
    Node<T>* head;
};

template <typename T>
List<T>::List()
{
    Size = 0;
    head = nullptr;
}

template <typename T>
List<T>::~List()
{
    //cout << "Вызван деструктор List" << endl;
    clear();
}
```

```
// Методы
// добавление в начало списка
template<typename T>
inline void List<T>::push_front(T* data)
{
    head = new Node<T>(data, head);// создаем  новый head
    Size++;
}
// удаление всех элементов списка
template<typename T>
void List<T>::clear()
{
    while (Size)
    {
        pop_front();
    }
}
// удаление последнего элемента
template<typename T>
void List<T>::pop_back()
{
    removeAt(Size - 1);
}
// удаление по индексу
template<typename T>
void List<T>::removeAt(int index)
{
    if (index == 0)
    {
        pop_front();
    }
    else
    {
        Node<T>* previous = this->head;
        // находим елемент предшествующий тому элементу который мы ищем
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }

        Node<T>* toDelete = previous->pNext;
        previous->pNext = toDelete->pNext;
        delete toDelete;
        Size--;
    }
}
// вставка по индексу
template<typename T>
void List<T>::insertAt(T* value, int index)
{
    if (index == 0)
    {
        push_front(value);
    }
    else
    {
        Node<T>* previous = this->head;
        // находим елемент предшествующий тому элементу который мы ищем
        for (int i = 0; i < index - 1; i++)
        {
            previous = previous->pNext;
        }
        // создаем новый элемент
        Node<T>* newNode = new Node<T>(value, previous->pNext);
        previous->pNext = newNode;
        Size++;
    }
```

```
}
// удаление первого элемента списка
template<typename T>
void List<T>::pop_front()
{
    Node<T>* temp = head;

    head = head->pNext;

    delete temp;

    Size--;
}
// добавление в конец списка
template<typename T>
void List<T>::push_back(T* data)
{
    if (head == nullptr)
    {
        head = new Node<T>(data);
    }
    else
    {
        Node<T>* current = this->head;

        while (current->pNext)
        {
            current = current->pNext;
        }
        current->pNext = new Node<T>(data);
    }
    Size++;
}
// количество элементов
template<typename T>
int List<T>::GetSize()
{
    return Size;
}
// просмотр всей структуры
template<typename T>
inline void List<T>::show()
{
    Node<T>* current = this->head;
    while (current != nullptr)
    {
        current->data->Show(cout);
        current = current->pNext;
    }
}

// специализация show
template <>
void List <int>::show() {
       Node<int>* tmp = head;
       while (tmp != NULL) {
              cout << *tmp->data;
              tmp =tmp->pNext;
       }
}

// изменение элемента по индексу
template<typename T>
inline void List<T>::input(int n)
{
    Node<T>* current = this->head;
    int i = 0;
```

```
    while (i != n) {
        current = current->pNext;
        i++;
    }
    current->data->Input(cin);
}

// специализация input
template <>
void List <int> ::input(int n) {
      Node<int>* tmp = head;
      int i = 0;
      while (i != n) {
            tmp = tmp->pNext;
            i++;
      }
      cin >> *tmp->data;
}
```