

Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курский государственный университет»

Кафедра программного
обеспечения и администрирования
информационных систем

Направление подготовки
математическое обеспечение и
администрирование
информационных систем

Форма обучения очная

Отчет
по лабораторной работе №4
«Шаблоны классов»

Выполнил:

студент группы 213

Тихонов Е.Е.

Проверил:

старший преподаватель кафедры ПОиАИС

Ураева Е.Е.

Курск, 2021

Цель работы: изучить особенности реализации шаблонов классов на языке C++.

Задание

Задача 1. Создать шаблонный класс List для работы с двусвязным списком элементов любого типа. В качестве членов-данных рекомендуется брать два элемента (определяющие начало и конец списка) самоссылочного класса Node (должен быть другом основному классу) следующего вида:

```
class Node { Type data; Node *next; Node *prev; };
```

Класс List должен содержать конструктор по умолчанию, основной конструктор и конструктор копирования. Определить в этом классе функции-члены класса, обеспечивающие: добавление элементов в список, удаление элемента из списка, проверку, не является ли список пустым, распечатку элементов списка на экране дисплея. Дополнительно перегрузить операторную функцию для операции - (вычитание), которая удаляет 2-й элемент списка. Новый список отобразить на экране.

Разработка алгоритма

Задача 1

Входные данные: нет.

Выходные данные: текстовые данные в консоли.

Текст программы

Текст программы для решения задачи 1

```
#include <iostream>
#include <string>

using namespace std;
```

```
template <typename DataType>
class Node;
```

```
template <typename T>
class List
```

```
{
    Node<T>* First;
    Node<T>* Last;
    int Count = 0;
```

```
public:
```

```
    int getCount()
    {
        return Count;
    }
```

```
    List()
    {
        First = Last = nullptr;
    }
```

```
    List(T* Array, int Count)
    {
        for (size_t i = 0; i < Count; i++)
        {
            Add(Array[i]);
        }
    }
```

```
    List(List<T>& ListObj)
    {
        int Count = ListObj.getCount();
        for (size_t i = 0; i < Count; i++)
```

```

        {
            Add(ListObj.At(i));
        }
    }

```

```

void Add(T Element)
{
    Node<T>* temp = new Node<T>(Element);

    if (First == nullptr)
    {
        First = temp;
        Last = temp;
    }
    else
    {
        Last->Next = temp;
        temp->Prev = Last;
        Last = temp;
    }
    Count++;
}

```

```

private: Node<T>* NodeAt(int Indx)
{
    Node<T>* Elem = First;
    for (size_t i = 0; i < Indx; i++)
    {
        Elem = Elem->Next;
    }
    return Elem;
}

```

```

public:
T At(int Indx)
{
    NodeAt(Indx)->Data;
}

void Remove(int Indx)
{
    Node<T>* Elem = NodeAt(Indx);
    Node<T>* Before = Elem->Prev;
    Node<T>* After = Elem->Next;

    if (Before != nullptr)
    {
        Before->Next = After;
    }
    else
    {
        First = After;
    }

    if (After != nullptr)
    {
        After->Prev = Before;
    }
    else
    {
        Last = Before;
    }

    delete Elem;
    Count--;
}

```

```

bool IsEmpty()
{
    if (First == nullptr)
    {
        return true;
    }
    return false;
}

void ToString()
{
    Node<T>* Elem = First;
    while (Elem != nullptr)
    {
        cout << Elem->Data << endl;
        Elem = Elem->Next;
    }
}

void operator-(int Num)
{
    Remove(1);
}

};

template <typename DataType>
class Node
{
    friend class List<DataType>;
    DataType Data;
    Node* Next;
    Node* Prev;

```

```

Node(DataType Data = 0)
{
    this->Data = Data;
    this->Next = nullptr;
    this->Prev = nullptr;
}
};

int main()
{
    List<int> list1;

    cout << "List(empty): " << endl;
    list1.ToString();
    cout << "Is empty: " + to_string(list1.IsEmpty())<< endl;

    cout << "List(add 3 elem): " << endl;
    list1.Add(5);
    list1.Add(2);
    list1.Add(3);

    list1.ToString();
    cout << "Is empty: " + to_string(list1.IsEmpty()) << endl;

    cout << "List(remove 2nd elm): " << endl;
    list1.Remove(1);

    list1.ToString();
    cout << "Is empty: " + to_string(list1.IsEmpty()) << endl;

    cout << "List(remove all): " << endl;
    while(!list1.IsEmpty())
    {

```

```

        list1.Remove(0);
    }
    list1.ToString();
    cout << "Is empty: " + to_string(list1.IsEmpty()) << endl;

    cout << "List(from array): " << endl;

    string* Array = new string[3];
    Array[0] = "aaa";
    Array[1] = "bbb";
    Array[2] = "ccc";

    List<string> list2(Array, 3);
    list2.ToString();

    cout << "List(copy): " << endl;
    List<string> list3(list2);
    list3.ToString();

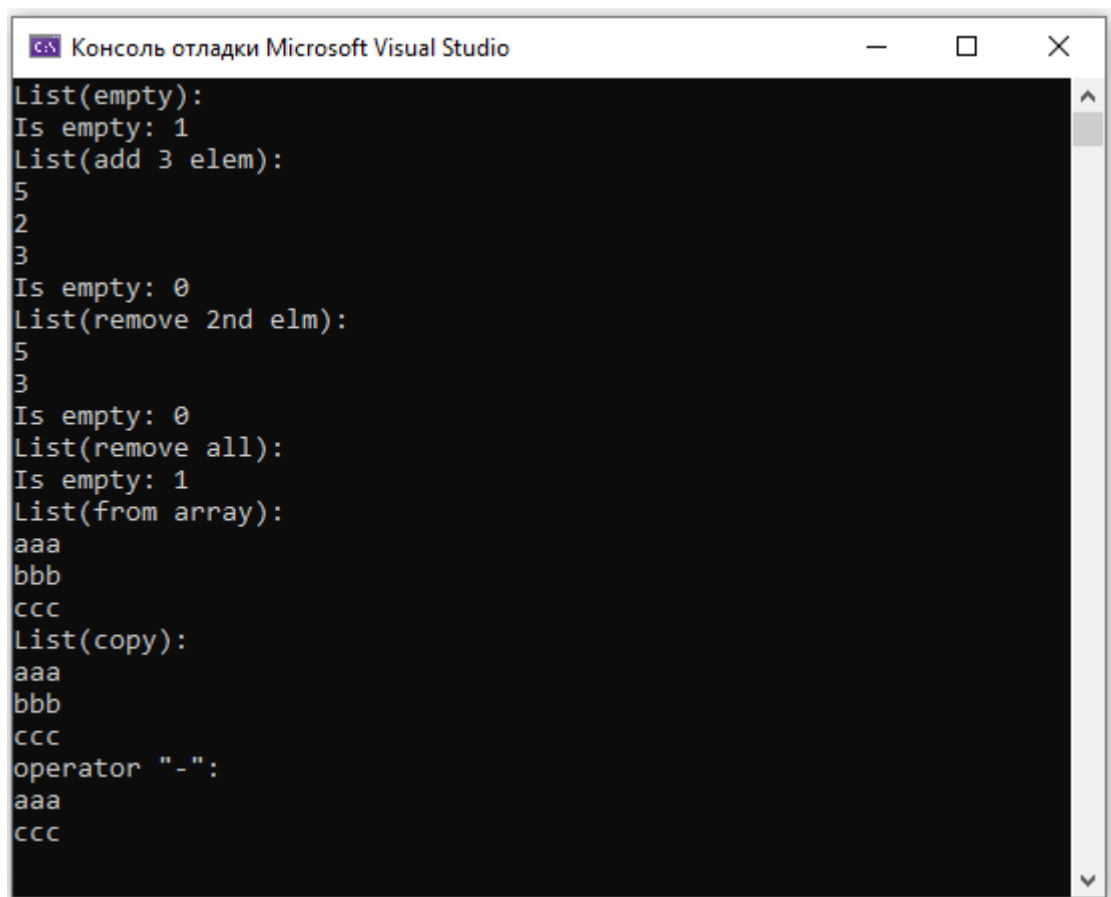
    cout << "operator \"-\": " << endl;
    list3-0;
    list3.ToString();

    return 0;
}

```

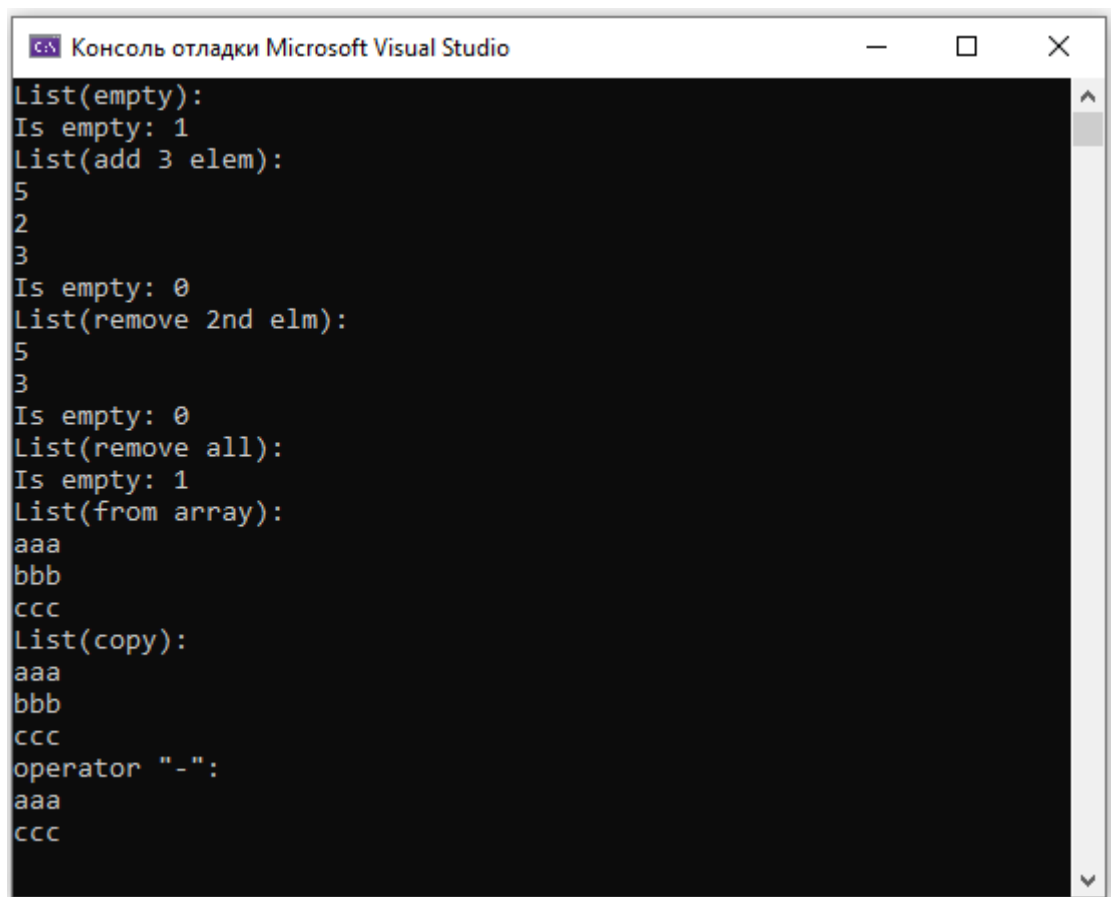
Тестирование программы

Тестирование задачи 1 представлено на рисунках 1, 2.



```
Консоль отладки Microsoft Visual Studio
List(empty):
Is empty: 1
List(add 3 elem):
5
2
3
Is empty: 0
List(remove 2nd elm):
5
3
Is empty: 0
List(remove all):
Is empty: 1
List(from array):
aaa
bbb
ccc
List(copy):
aaa
bbb
ccc
operator "-":
aaa
ccc
```

Рисунок 1 - Тест 1 задачи 1



```
Консоль отладки Microsoft Visual Studio
List(empty):
Is empty: 1
List(add 3 elem):
5
2
3
Is empty: 0
List(remove 2nd elm):
5
3
Is empty: 0
List(remove all):
Is empty: 1
List(from array):
aaa
bbb
ccc
List(copy):
aaa
bbb
ccc
operator "-":
aaa
ccc
```

Рисунок 2 - Тест 2 задачи 1