Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

Пермский национальный исследовательский политехнический университет

Факультет электротехнический

Кафедра ИТАС

ОТЧЁТ

о лабораторной работе №7 по классам

Выполнил:
Студент группы ИВТ-23-1Б
Пискунов Д. А.

_____

Проверил:
Доцент кафедры ИТАС
Яруллин Д.В.

_____

_____

Пермь 2024

Задача:

15	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера списка; * вектор – умножение элементов списков a[i]*b[i]; Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

Текст программы
файл Lab_Main_7.cpp

```cpp
#include <iostream>
#include "List.h"
#include "Pair.h"

using namespace std;

int main() {
    system("chcp 1251 > null");

    List <int> l;
    l << 2;
    l << 4;
    cout << "List<int>:\n";
    cout << l << '\n';

    List <Pair> p;
    Pair c;
    c.SetFirst(14);
    c.SetSecond(8.8);
    p << c;
    cout << "List<Pair>:\n";
    cout << p << '\n';

    return 0;
}
```

Файл List.h

```
#pragma once

#include<iostream>
#include<string>

using namespace std;

template<class T>
struct Node {
	T key;
	Node<T>* next = nullptr;
};

template <class T>
class List {
private:
	Node<T>* lastNd, * current, * head;
	int n = 0;
public:
	List() {};
	List(int count) {
		n = count;
		head = new Node<T>;
		head->key = 0;
		lastNd = head;
		for (int i = 1; i < n; i++) {
			current = new Node<T>;
			current->key = 0;
			lastNd->next = current;
			lastNd = current;
		}
		lastNd->next = NULL;
	};
	~List() {
		lastNd = head;
		while (lastNd != NULL) {
			current = lastNd->next;
			delete lastNd;
			lastNd = current;
		}
		n = 0;
	};
	List& operator = (List<T>& l) {
		if (this != &l) {
			if (this != 0) {
				lastNd = head;
```

```
46                                      lastNd = head;
47                                      while (lastNd != NULL) {
48                                              current = lastNd->next;
49                                              delete lastNd;
50                                              lastNd = current;
51                                      }
52                                      n = 0;
53                              }
54                              lastNd = head = new Node<T>;
55                              l.lastNd = l.head->next;
56                              lastNd->key = l.head->key;
57                              while (l.lastNd != NULL) {
58                                      lastNd->next = new Node<T>;
59                                      lastNd->next->key = l.lastNd->key;
60                                      l.lastNd = l.lastNd->next;
61                                      lastNd = lastNd->next;
62                              }
63                              lastNd->next = NULL;
64                              n = l.n;
65                      }
66                      return *this;
67              };
68              int& operator[] (int index) {
69                      if (index < n) {
70                              lastNd = head;
71                              for (int i = 0; i < index; i++) {
72                                      lastNd = lastNd->next;
73                              }
74                              return lastNd->key;
75                      }
76                      else {
77                              cout << "Запредельный индекс";
78                      }
79              };
80              int operator () () {
81                      return n;
82              };
83              friend ostream& operator << (ostream& out, List<T>& l) {
84                      if (l.n) {
85                              l.lastNd = l.head;
86                              while (l.lastNd != NULL) {
87                                      out << l.lastNd->key << ' ';
88                                      l.lastNd = l.lastNd->next;
89                              }
90                      }
91                      else {
```

```
91                      else {
92                              out << "Пусто";
93                      }
94                      return out;
95              };
96              friend istream& operator >> (istream& in, List<T>& l) {
97                      l.lastNd = l.head;
98                      while (l.lastNd != NULL) {
99                              in >> l.lastNd->key;
100                             l.lastNd = l.lastNd->next;
101                     }
102                     return in;
103             };
104             void operator << (T t) {
105                     if (head == NULL) {
106                             head = new Node<T>;
107                             head->key = t;
108                             head->next = NULL;
109                     }
110                     else {
111                             lastNd = head;
112                             while (lastNd->next != NULL) {
113                                     lastNd = lastNd->next;
114                             }
115                             lastNd->next = new Node<T>;
116                             lastNd->next->key = t;
117                             lastNd = lastNd->next;
118                             lastNd->next = NULL;
119                     }
120                     n += 1;
121             };
122     };
```

Файл Pair.cpp

```cpp
#include "Pair.h"

Pair::Pair(int first, double second) {
	this->first = first;
	this->second = second;
}
istream& operator>>(istream& in, Pair& m_pair) {
	in >> m_pair.first;
	in >> m_pair.second;
	return in;
}
ostream& operator<<(ostream& out, const Pair& m_pair) {
	return (out << m_pair.first << " : " << m_pair.second);
}
Pair::Pair() {
	first = 0;
	second = 0;
}
Pair::~Pair() {
}
Pair::Pair(const Pair& m_pair) {
	first = m_pair.first;
	second = m_pair.second;
}
Pair& Pair::operator=(const Pair& m_pair) {
	if (&m_pair != this) {
		first = m_pair.first;
		second = m_pair.second;
	}
	return *this;
}
Pair Pair::operator-(const Pair& m_pair) {
	Pair result(first - m_pair.first, second - m_pair.second);
	return result;

}
Pair& Pair::operator+(int first) {
	this->first += first;
	return *this;
}
Pair& Pair::operator+(double second) {
	this->second += second;
	return *this;
}
```

```
44      }
45      Pair& Pair::operator++() {
46              ++first;
47              ++second;
48              return *this;
49      }
50      Pair Pair::operator ++(int) {
51              Pair temp = *this;
52              this->first++;
53              this->second++;
54              return temp;
55      }
56
57      int Pair::GetFirst() const {
58              return first;
59      }
60
61      double Pair::GetSecond() const {
62              return second;
63      }
64
65      void Pair::SetFirst(int m_first) {
66              first = m_first;
67      }
68
69      void Pair::SetSecond(double m_second) {
70              second = m_second;
71      }
```

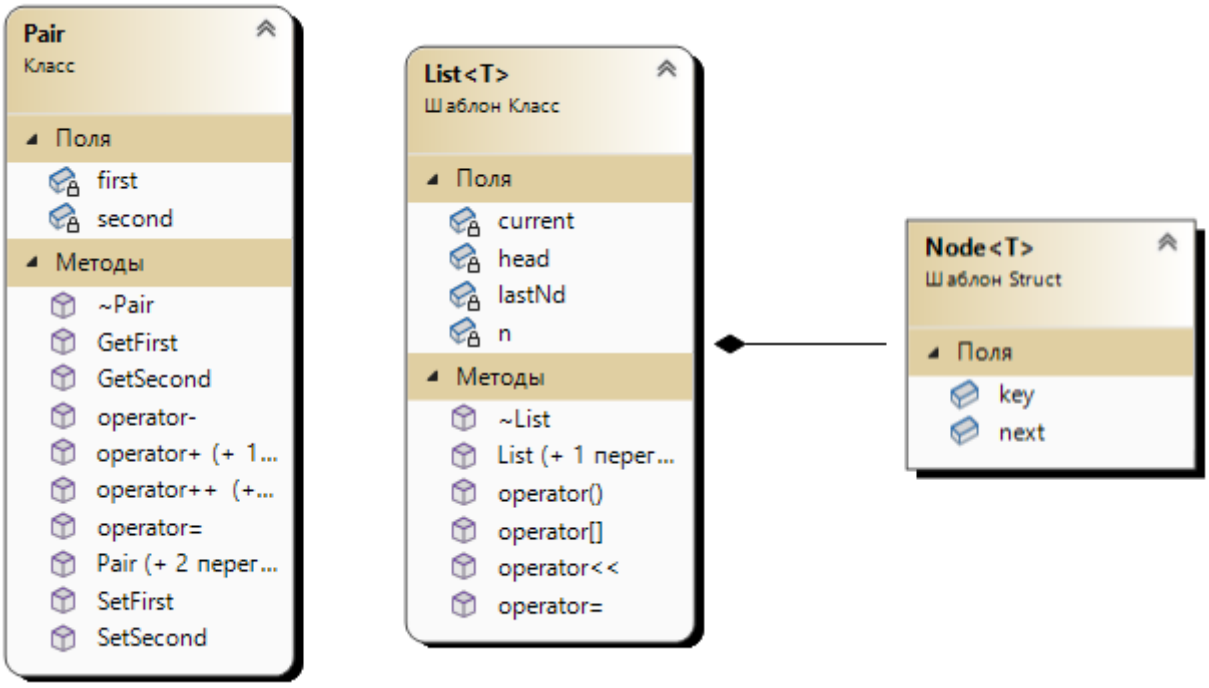
Файл Pair.h

```
#pragma once
#include <iostream>

using namespace std;

class Pair{
private:
        int first;
        double second;
public:
        friend istream& operator>>(istream& in, Pair& p);
        friend ostream& operator<<(ostream& out, const Pair& p);
        Pair(int, double);
        Pair();
        Pair(const Pair&);
        ~Pair();
        Pair& operator=(const Pair&);
        Pair operator-(const Pair&);
        Pair& operator+(int);
        Pair& operator+(double);
        Pair& operator++();
        Pair operator++(int);
        int GetFirst() const;
        double GetSecond() const;
        void SetFirst(int m_first);
        void SetSecond(double m_second);
};
```

UML-диаграмма



Тест

```
List<int>:
2 4
List<Pair>:
14 : 8.8

C:\Users\MOkASiH\Desktop\Test\x64\Debug\Test.exe (процесс 14884) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно…
```

Ответы на вопросы

1. В чем смысл использования шаблонов?

Ответ: возможность использовать один класс или функцию для различных типов данных

2. Каковы синтаксис/семантика шаблонов функций?

Ответ: template<параметры> возвращаемое значение имя (параметры) {...}

3. Каковы синтаксис/семантика шаблонов классов?

Ответ: template<параметры> class имя {...}

4. Что такое параметры шаблона функции?

Ответ: Типы данных, которые будут передаваться в функцию

5. Перечислите основные свойства параметров шаблона функции.

Ответ:

1. В списке параметров шаблона может быть несколько параметров, каждому предшествует ключевое слово *typename* или *class*. `//template <class T1, T2>`
2. Имя параметра шаблона имеет в определяемой шаблоном функции все права имени типа, то есть с его помощью могут специализироваться формальные параметры, определяться тип возвращаемого функцией значения и типы любых объектов, локализованных в теле функции.

3. Параметризированная функция может иметь сколь угодно непараметризированных формальных параметров.

```
template <class T1>
void function4(T1 a, int b, bool c) {}
```

4. В списке описания прототипа шаблона имена параметров не обязаны совпадать с именами в описании шаблона.

5. При конкретизации шаблонного определения функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковыми.

6. Как записывать параметр шаблона?

Ответ: template <class/typename TYPE>

7. Можно ли перегружать параметризованные функции?

Ответ: Да

8. Перечислите основные свойства параметризованных классов.

Ответ:

1. Компонентные функции параметризованного класса автоматически являются параметризованными.
2. Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по шаблону.
3. Если дружественная функция содержит в своем описании параметр типа параметризованного класса, то каждый класс, организованный по шаблону имеет собственную параметризованную дружественную функцию.
4. В параметризованном классе нельзя определить дружественные параметризованные классы.
5. Шаблоны могут быть базовыми классами. Производными классами от такого класса могут быть обычные или шаблонные классы. Шаблоны могут наследоваться как от обычных, так и от шаблонных классов.
6. Шаблоны функций-членов нельзя описывать как виртуальные.
7. Локальные (вложенные) классы не могут содержать шаблоны в качестве своих элементов
8. Определённые пользователем имена в описании шаблона рассматриваются как идентификаторы переменных. Чтобы имя рассматривалось как идентификатор типа, оно должно быть определено через ключевое слово typename.

9. Все ли компонентные функции параметризованного класса являются параметризованными?

Ответ: Да

10. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

Ответ: Нет

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Ответ: Нет

12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?

Ответ:

Если компонентная функция описывается вне шаблона класса:

template <список_параметров> тип_функции имя_класса <список_имен_параметров> :: имя_функции(список параметров)
{тело функции}

13. Что такое инстанцирование шаблона?

Ответ: Процесс генерации компилятором конкретного класса по шаблону при создании экземпляра параметризованного класса.

14. На каком этапе происходит генерирование определения класса по шаблону?

Ответ: На этапе создания экземпляра класса