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 №9 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]; +n - переход вправо к элементу с номером n.	3 , 1

Текст программы
файл Error.h

```cpp
#pragma once
#pragma once
#include <string>
#include <iostream>
using namespace std;

class Error {
private:
	string code[4] = { "Error: can't shift, going off the list!", "Error: imposible get first element empty list!", "Error: imposible get last element empty list!", "Error: index outside the list!" };
	string str;
public:
	Error(int i) {
		str = code[i];
	}
	void What() {
		cout << str << '\n';
	}
};
```

Файл List.cpp

```cpp
#include "List.h"

List::List(int count) {
	n = count;
	head = new Node;
	head->key = 0;
	lastNd = head;
	for (int i = 1; i < n; i++) {
		current = new Node;
		current->key = 0;
		lastNd->next = current;
		lastNd = current;
	}
	lastNd->next = NULL;
}

List::~List() {
	lastNd = head;
	while (lastNd != NULL) {
		current = lastNd->next;
		delete lastNd;
		lastNd = current;
	}
	n = 0;
}

List& List::operator=(List& l) {
	if (this != &l) {
		if (this != 0) {
			lastNd = head;
			while (lastNd != NULL) {
				current = lastNd->next;
				delete lastNd;
				lastNd = current;
			}
			n = 0;
		}
		lastNd = head = new Node;
		l.lastNd = l.head->next;/////
		lastNd->key = l.head->key;
		while (l.lastNd != NULL) {
			lastNd->next = new Node;
			lastNd->next->key = l.lastNd->key;
			l.lastNd = l.lastNd->next;
			lastNd = lastNd->next;
		}
		lastNd->next = NULL;
		n = l.n;
```

```
48                      n = l.n;
49              }
50              return *this;
51      }
52
53      int& List::operator[](int index) {
54              if (index >= n) {
55                      throw Error(3);
56              }
57              lastNd = head;
58              for (int i = 0; i < index; i++) {
59                      lastNd = lastNd->next;
60              }
61              return lastNd->key;
62      }
63
64      int List::operator () () {
65              return n;
66      }
67
68      Iterator List::first() {
69              if (!n) {
70                      throw Error(1);
71              }
72              beg.current = head;
73              return beg;
74      }
75
76      Iterator List::last() {
77              if (!n) {
78                      throw Error(2);
79              }
80              lastNd = head;
81              while (lastNd->next != NULL) {
82                      lastNd = lastNd->next;
83              }
84              end.current = lastNd;
85              return end;
86      }
87
88      ostream& operator<<(ostream& out, List& l) {
89              if (l.n) {
90                      l.lastNd = l.head;
91                      while (l.lastNd != NULL) {
92                              out << l.lastNd->key << ' ';
93                              l.lastNd = l.lastNd->next;
94                      }
```

```
94                      }
95              }
96              else {
97                      out << "Empty";
98              }
99              return out;
100     }
101
102     istream& operator>>(istream& in, List& l) {
103             l.lastNd = l.head;
104             while (l.lastNd != NULL) {
105                     in >> l.lastNd->key;
106                     l.lastNd = l.lastNd->next;
107             }
108             return in;
109     }
110
111     void List::operator << (int number) {
112             if (head == NULL) {
113                     head = new Node;
114                     head->key = number;
115                     head->next = NULL;
116             }
117             else {
118                     lastNd = head;
119                     while (lastNd->next != NULL) {
120                             lastNd = lastNd->next;
121                     }
122                     lastNd->next = new Node;
123                     lastNd->next->key = number;
124                     lastNd = lastNd->next;
125                     lastNd->next = NULL;
126             }
127             n += 1;
128     }
129
130     List operator*(const List& x, const List& y) {
131             return x * y;
132     }
```

Файл List.h

```
#pragma once
#include <string>
#include <iostream>
using namespace std;
#include "Error.h"

struct Node {
	int key;
	Node* next;
};

class Iterator {
private:
	Node* current;
	friend class List;
public:
	Iterator() {
		current = NULL;
	};
	Iterator(Node* node) {
		current = node;
	};
	void operator ++ () {
		if (current == NULL) {
			throw Error(0);
		}
		current = current->next;
	}
	Node* operator *() const {
		return current;
	}
	void operator + (int shift) {
		Node* tmp = current;
		int i = 0;
		while (i < shift && current != NULL && current->next != NULL) {
			current = current->next;
			i++;
		}
		if (i < shift) {
			current = tmp;
			throw Error(0);
		}
	}
};

```

```
45
46  class List {
47  public:
48          List() {};
49          List(int);
50          ~List();
51          List& operator = (List&);
52          int& operator[] (int);
53          int operator () ();
54          friend List operator*(const List&, const List&);
55          friend ostream& operator << (ostream&, List&);
56          friend istream& operator >> (istream&, List&);
57          Iterator first();
58          Iterator last();
59          void operator << (int);
60  private:
61          Node* lastNd, * current, * head;
62          int n = 0;
63          Iterator beg, end;
64  };
```

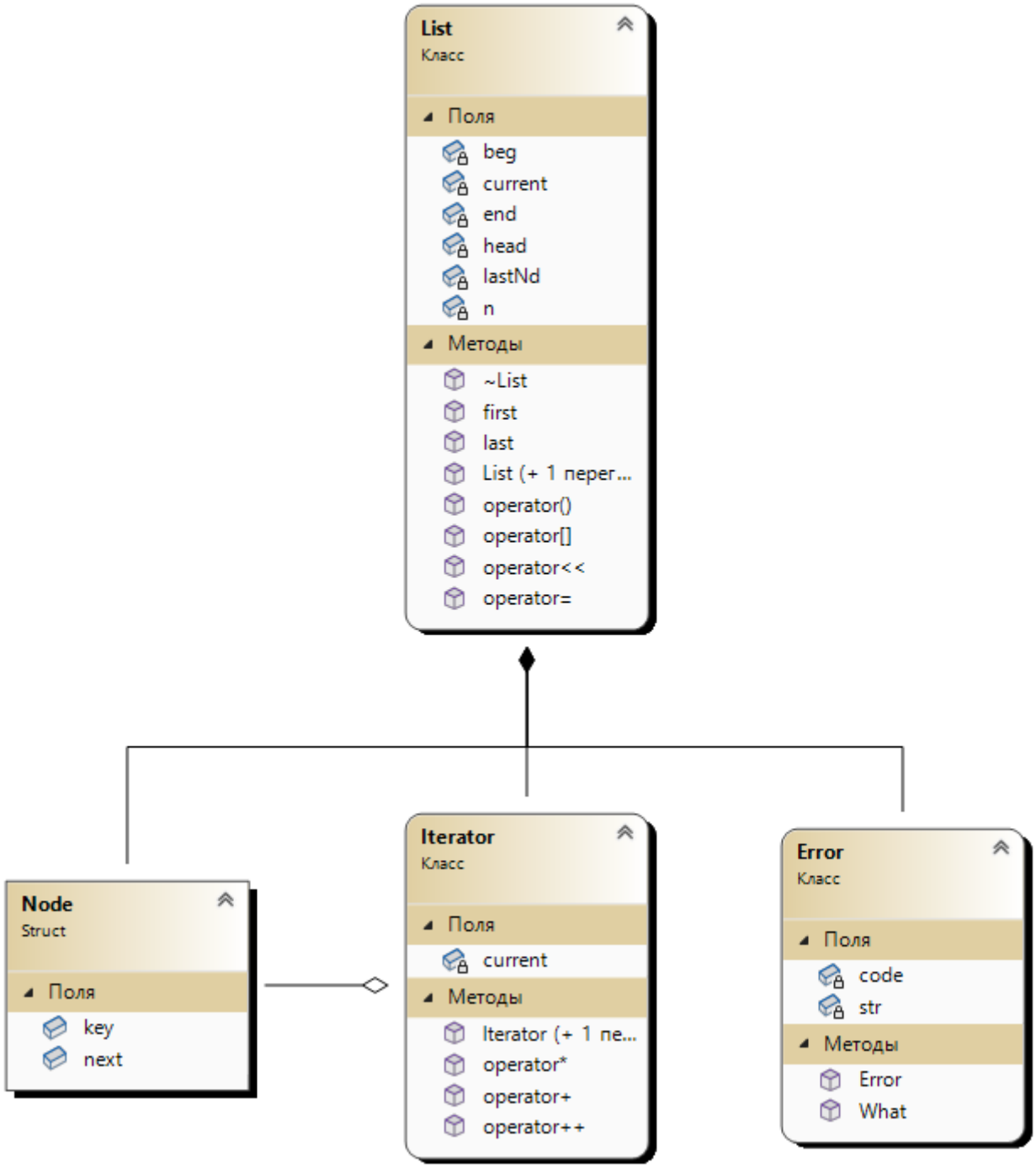
Файл infa.cpp

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

using namespace std;

int main() {
    system("chcp 1251 > NULL");
    try {
        List a(2);
        List b(3);
        a[1] = 10;
        b[2] = 20;
        cout << a[1] * b[2] << endl;
        cout << a << '\n';
        cout << b << '\n';
        cout << b[0] << endl;
        Iterator i = b.first();//Ошибка
        Iterator j = b.last();//Ошибка
        Iterator k = a.last();
        ++k;
        ++k;//Ошибка
        k + 1;//Ошибка
    }
    catch (Error& m_Error) {
        m_Error.What();
    }
    return 0;
}
```

UML-диаграмма



Тест

```
200
0 10
0 0 20
0
Error: can't shift, going off the list!

C:\Users\MOkASiH\Desktop\Test\x64\Debug\Test.exe (процесс 17628) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно…
```

Ответы на вопросы

1. Что представляет собой исключение в C++?

Ответ: исключение — объект, генерируемый системой при возникновении исключительной ситуации.

2. На какие части исключения позволяют разделить вычислительный процесс? Достоинства такого подхода?

Исключения позволяют разделить вычислительный процесс на 2 части:

- Обнаружение аварийной ситуации
- Обработка аварийной ситуации.

Достоинства:

- Удобство использования в программе из нескольких модулей
- Отсутствие требования возвращать значение в вызывающую функцию

3. Какой оператор используется для генерации исключительной ситуации?

throw

4. Что представляет собой контролируемый блок? Для чего он нужен?

Контролируемый блок try нужен для проверки помещённого кода на исключения. В нем описывается оператор throw, который передает исключения в блок обработки исключения catch

5. Что представляет собой секция-ловушка? Для чего она нужна?

Блок catch – обработчик исключения. Он нужен для перехватки исключения, которое кидает оператор throw.

6. Какие формы может иметь спецификация исключения в секции ловушке, В каких ситуациях используются эти формы?

Спецификация исключения может быть выражена типом и именем, типом и остальными исключениями. Первая и вторая формы перехватывают исключения выбранного типа. Третья форма обрабатывает все исключения, которые не попадают по выделенный тип исключения

7. Какой стандартный класс можно использовать для создания собственной иерархии исключений?

exception

8. Каким образом можно создать собственную иерархию исключений?

Описанием собственных классов

9. Если спецификация исключений имеет вид: void f1()throw(int,double); то какие исключения может порождать функция f1()?

int и double

10. Если спецификация исключений имеет вид: void f1()throw(); то какие исключения может порождать функция f1()?

Не может

11. В какой части программы может генерироваться исключение?

Внутри оператора try

12. Написать функцию, которая вычисляет площадь треугольника по трем сторонам (формула Герона).

    Функцию реализовать в 4 вариантах:

    - без спецификации исключений;

```
double Area (double a, double b, double c) {
        double p = (a + b + c) / 2;
        return sqrt(p * (p - a) * (p - b) * (p - c));
}
```

    - со спецификацией throw();

```
double Area(double a, double b, double c) throw() {
   double p = (a + b + c) / 2;
   return sqrt(p * (p - a) * (p - b) * (p - c));
```

```
}
```

- с конкретной спецификацией с подходящим стандартным исключением;

```
#include <stdexcept>
double Area(double a, double b, double c) {
   if (a <= 0 || b <= 0 || c <= 0 || a + b <= c || a + c <= b || b + c <= a) {
      throw invalid_argument("Invalid triangle sides");
   }
   double p = (a + b + c) / 2;
   return sqrt(p * (p - a) * (p - b) * (p - c));
}
```

- спецификация с собственным реализованным исключением

```
#include <exception>
class Error : public exception {
public:
   virtual const char* what() const throw() {
      return "Invalid triangle sides";
   }
};
double Area(double a, double b, double c) {
   if (a <= 0 || b <= 0 || c <= 0 || a + b <= c || a + c <= b || b + c <= a) {
      throw Error;
   }
   double p = (a + b + c) / 2;
   return sqrt(p * (p - a) * (p - b) * (p - c));
}
```