Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №18.9**

Дисциплина: «Основы теории алгоритмов и структуры данных»  
Тема: Обработка исключительных ситуаций

Вариант 14

Выполнил:

студент группы РИС-20-2б

Вичугов Алексей Дмитриевич

Проверила:

доцент кафедры ИТАС

Полякова Ольга Андреевна

Пермь, 2021

**Цель работы**

1. Создание консольного приложения, состоящего из нескольких файлов.

2. Разработка программы, обрабатывающей исключительные ситуации.

**Постановка задачи**

1. Реализовать класс, перегрузить для него необходимые операции.
2. Определить исключительные ситуации.
3. Предусмотреть генерацию исключительных ситуаций.
4. Задание варианта: класс-контейнер список с ключевыми значениями типа int. Реализовать операции: доступ по индексу, определение размера списка, умножение всех элементов на число, переход влево к элементу с номером n.

**Анализ задачи**

1. Описание классов:

struct S{

int val;

S \*sled, \*pred;

};

class Lister{

public:

S \*first, \*last;

int len;

Lister();

Lister(S \*f, S \*l);

Lister(Lister &l);

~Lister();

void Create(int r);

int operator [](int index);

int operator ()();

void operator \*(int n);

friend ostream& operator <<(ostream& ost, Lister &l);

};

class Iterator{

public:

Lister \*list;

S \*cur;

Iterator();

Iterator(Lister \*l);

Iterator(Iterator &i);

~Iterator();

S\* operator -(int n);

S\* operator --();

S\* operator ++();

};

class Error{

public:

string s;

Error(string arg){s=arg;}

};

class Err{

public:

string s;

void Er(){cout << s << endl;}

};

class RError:public Err{

public:

RError(){

s="Incorrect size of list! ";

}

};

class IntError:public Err{

public:

IntError(){

s="No number entered! ";

}

};

2. Определение компонентных функций:  
Lister::Lister(){len=0;first=NULL;last=NULL;}

Lister::Lister(S \*f, S \*l){first=f;last=l;}

Lister::Lister(Lister &l){first=l.first;last=l.last;len=l.len;}

Lister::~Lister(){}

void Lister::Create(int r){

if (r<1) throw RError();

else{

len=1;

first = new S;

first->pred=NULL;

first->sled=NULL;

S\* cur=first;

S\* nov=NULL;

cout << "1: ";

cin >> cur->val;

if (cin.fail()) throw IntError();

for (int i=1;i<r;i++){

len++;

nov = new S;

nov->pred=cur;

nov->sled=NULL;

cur->sled=nov;

cur=nov;

cout << i+1 << ": ";

cin >> cur->val;

if (cin.fail()) throw IntError();

}

last=cur;

}

}

int Lister::operator()(){

return len;

}

int Lister::operator [](int index){

bool b=0;

S \*cur=first;

if(len>index){

for (int i=0;i<index;i++){

if (cur->sled!=NULL){

cur=cur->sled;

}

else b=1;

}

}

else b=1;

if(b) throw Error("Error: the element with this index does not exist! ");

else return cur->val;

}

void Lister::operator \*(int n){

if(len>0){

S \*cur=first;

while (cur->sled!=NULL){

cur->val\*=n;

cur=cur->sled;

}

cur->val\*=n;

}

}

Iterator::Iterator(){list=NULL;cur=NULL;}

Iterator::Iterator(Lister \*l){list=l;cur=l->first;}

Iterator::Iterator(Iterator &i){list=i.list;cur=i.cur;}

Iterator::~Iterator(){}

S\* Iterator::operator -(int n){

if (cur!=NULL)

for (int i=0;i<n;i++){

if(cur->pred!=NULL) cur=cur->pred;

}

return cur;

}

S\* Iterator::operator --(){

if (cur!=NULL&&cur->pred!=NULL) cur=cur->pred;

return cur;

}

S\* Iterator::operator ++() {

if (cur!=NULL&&cur->sled!=NULL) cur=cur->sled;

return cur;

}

3. Определение глобальных функций:  
ostream& operator << (ostream& ost, Lister &l){

S\* cur=l.first;

while(cur->sled!=NULL){

ost << cur->val << " ";

cur=cur->sled;

}

ost << cur->val << endl;

return ost;

}

4. Функция main():

main(){

try{

Lister l;

int n;

cout << "Enter the size of container: "; cin >> n; if (cin.fail()||n<1) throw 1;

l.Create(n);

cout << l() << endl;

cout << l[3] << endl;

cout << l << endl;

}

catch (int){

cout << "Error was detected. " << endl;

return 0;

}

catch(Error e){

cout << e.s << endl;

return 0;

}

catch(Err e){

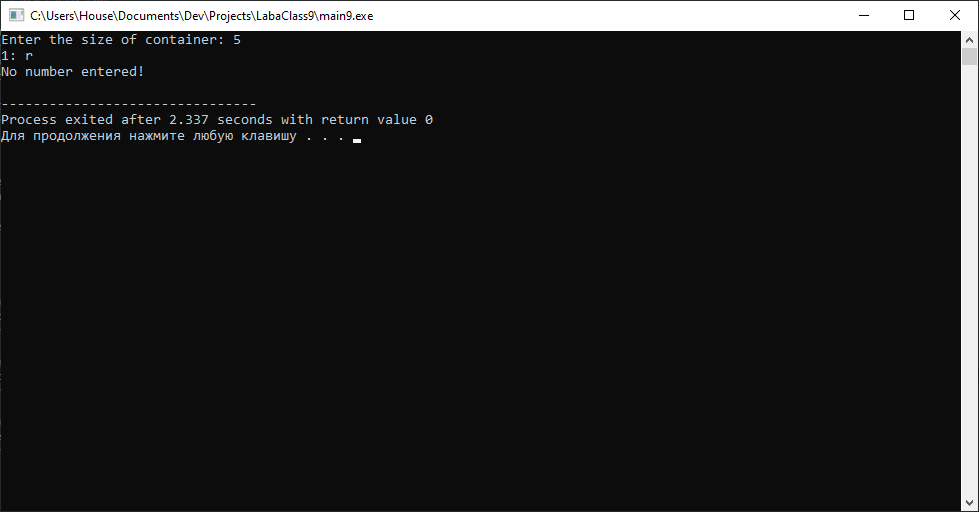
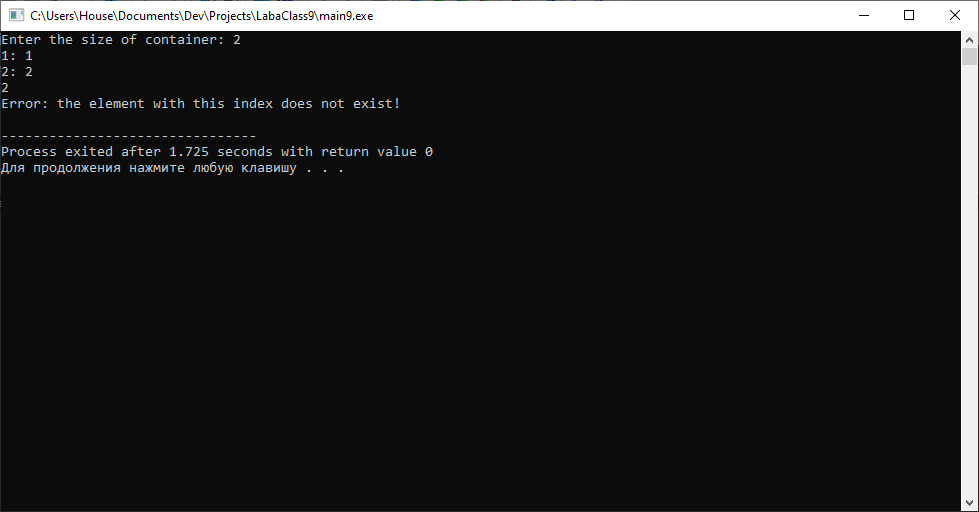
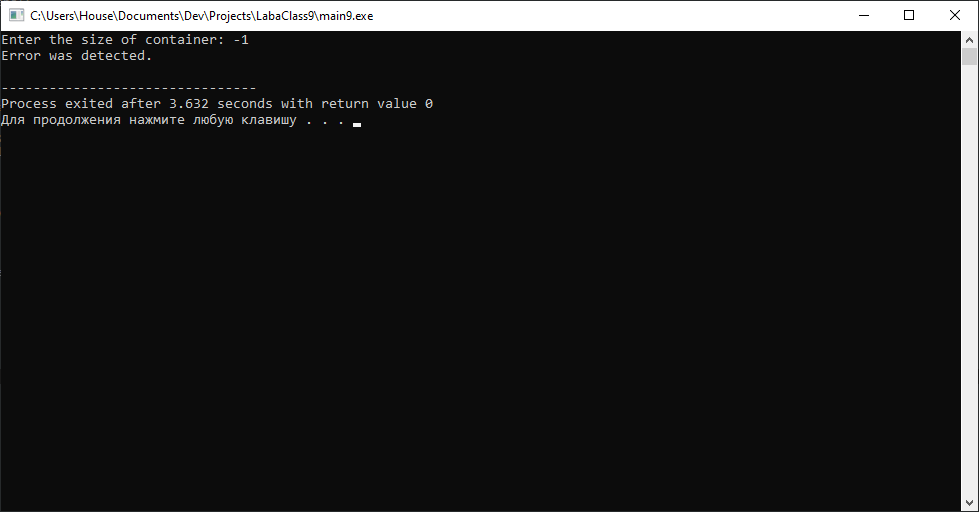
e.Er();

return 0;

}

}

**Результаты работы программы**

****

**Ответы на вопросы**

1. Непредвиденное аварийное событие.
2. Обнаружение аварийной ситуации, и её обработка. Это позволяет избежать аварийного завершения программы. Такой подход удобно использовать в программе, которая состоит из нескольких модулей. При использовании такого подхода не требуется возвращать значение в вызывающую функцию.
3. Throw.
4. Блок, в котором предположительно может произойти непредвиденная ситуация. При её возникновении выполнение блока повторяется ещё раз. При повторном возникновении непредвиденной ситуации генерируется исключение.
5. Это секция, которая срабатывает при генерации исключения определённого типа. Используется для обработки исключения или вывода сообщения об ошибке.
6. Форма catch(тип имя), как и форма cath(тип) обрабатывают определённые исключения, а форма catch(тип) перехватывает все исключения.
7. Exception.
8. Объявить базовый класс-исключение. Остальные классы исключений будут его наследниками.
9. Исключения типов int и double.
10. Данная функция не генерирует исключений.
11. В любой.
12. Double S(double a, double b, double c){int p=(a+b+c)/2; return sqrt(p\*(p-a)\*(p-b)\*(p-c));  
    Double S(double a, double b, double c) throw() {int p=(a+b+c)/2; return sqrt(p\*(p-a)\*(p-b)\*(p-c));

Double S(double a, double b, double c) throw(double){int p=(a+b+c)/2; return sqrt(p\*(p-a)\*(p-b)\*(p-c));

Class Error{};

Double S(double a, double b, double c)throw (Error) {int p=(a+b+c)/2; return sqrt(p\*(p-a)\*(p-b)\*(p-c));