Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №5  
по курсу «Объектно-ориентированное программирование»  
на тему «Механизмы исключений C++»

Выполнили студенты группы 22ВВП1:  
Беляев Д.

Захаров А.

Приняли:  
Евсеева Ю.И.  
Гудков А. А.

Пенза 2024

**Название**

Механизмы исключений C++

**Цель работы**

Изучить механизмы исключений C++

**Лабораторное задание**

1. Доработать шаблонный класс из предыдущей лабораторной работы таким образом, чтобы в программе осуществлялась генерация и обработка исключительных ситуаций.

Исключительные ситуации генерируются:

1. В конструкторе с параметром при попытке создать контейнер больше максимального

размера;

1. В операции []
2. При попытке обратиться к элементу с номером меньше 0 или больше текущего размера контейнера ;
3. В операции +
4. При попытке добавить элемент с номером больше максимального размера;
5. В операции –
6. При попытке удалить элемент из пустого контейнера.
7. Создать программу, использующую механизм обработки исключительных ситуаций, позволяющую выполнять перевод вещественного числа из одной системы счисления в другую. Перевод числа из одиннадцатеричной системы счисления в девятеричную.

**Листинг**

**main.cpp**

#include <iostream>

#include "Set.h"

#include "Calculate.h"

int main()

{

#pragma region Part1

// test 1

// Set<int> set1;

//set1.Insert(1);

//set1.Insert(2);

//set1.Insert(3);

//set1.Insert(4);

//std::cout << set1[0] << std::endl;

//std::cout << set1[50] << std::endl;

// test 2

//Set<int>\* set2 = new Set<int>(0, -1);

// test 3

/\*Set<int>\* set3 = new Set<int>(5, 2);

set3->Insert(1);

set3->Insert(3);

set3->Insert(4);\*/

#pragma endregion

#pragma region Part2

std::cout << DecToBase(143.367, 5) << std::endl;

std::cout << BaseToDec("1A3.367", 3) << std::endl;

std::cout << BaseToBase("1.AA3367", 3) << std::endl;

#pragma endregion

return 0;

}

**Set.h**

#ifndef SET

#define SET

#include <iostream>

#include <algorithm>

#include <iterator>

template<typename T>

class Set

{

public:

const unsigned int MAX\_SIZE = 20;

// Construct

Set()

{

\_size = MAX\_SIZE;

\_set = new T[\_size]{ (T)nullptr };

}

Set(T startVal, unsigned int size)

{

if (size <= 0 || size > MAX\_SIZE)

{

std::cout << "Size too small or too big" << std::endl;

throw "Size too small or too big";

}

\_size = size;

\_set = new T[\_size]{ (T)nullptr };

\_set[0] = startVal;

}

Set(T src[])

{

\_size = MAX\_SIZE;

\_set = new T[\_size]{ (T)nullptr };

for (size\_t i = 0; i < \_size; i++)

{

\_set[i] = src[i];

}

}

// Default

void Insert(T val)

{

if (this->Contains(val))

return;

int index = this->FindFreeSpace();

if (index == -1)

{

std::cout << "No free space" << std::endl;

throw "No free space";

}

\_set[index] = val;

}

void Erase(T val)

{

if (!this->Contains(val))

return;

int counter = 0;

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] == (T)nullptr)

counter++;

}

if(counter == \_size)

{

std::cout << "No space" << std::endl;

throw "No space";

}

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] == val)

\_set[i] = (T)nullptr;

}

}

bool Contains(T val)

{

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] == val)

return true;

}

return false;

}

int Size()

{

int count = 0;

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] != (T)nullptr)

count++;

}

return count;

}

void Print()

{

for (int i = 0; i < \_size; i++)

{

if (\_set[i] == (T)nullptr)

continue;

std::cout << \_set[i] << " ";

}

std::cout << std::endl;

}

T\* Values()

{

T\* tmp = new T[\_size]{ (T)nullptr };

int index = 0;

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] == (T)nullptr)

continue;

tmp[index] = \_set[i];

index++;

}

return tmp;

}

// Overload

Set operator +(T value)

{

Set<T>\* tmp = new Set<T>(this->Values());

tmp->Insert(value);

return \*tmp;

}

Set operator -(T value)

{

Set<T>\* tmp = new Set<T>(this->Values());

tmp->Erase(value);

return \*tmp;

}

Set operator \*(Set<T> secondSet)

{

T\* secVal = secondSet.Values();

Set<T>\* tmp = new Set<T>();

for (size\_t i = 0; i < \_size; i++)

{

if (secVal[i] == (T)nullptr)

continue;

if (this->Contains(secVal[i]))

tmp->Insert(secVal[i]);

}

for (int i = 0; i < \_size; i++)

{

if (\_set[i] == (T)nullptr)

continue;

if (secondSet.Contains(\_set[i]))

tmp->Insert(\_set[i]);

}

return \*tmp;

}

int operator ()()

{

return this->Size();

}

T& operator [](int index)

{

if (index < 0 || index > \_size)

{

std::cout << "Index out of range" << std::endl;

throw "Index out of range";

}

return \_set[index];

}

private:

unsigned int \_size = MAX\_SIZE;

T\* \_set;

int FindFreeSpace()

{

for (size\_t i = 0; i < \_size; i++)

{

if (\_set[i] == (T)nullptr)

return i;

}

return -1;

}

};

#endif // !

**Calculate.h**

#ifndef CALCULATE

#define CALCULATE

#include "string"

#include <algorithm>

std::string BaseToBase(std::string num, int accuracy);

float BaseToDec(std::string num, int accuracy);

std::string DecToBase(float num, int accuracy);

#endif

**Calculate.cpp**

#include "Calculate.h"

#include <cmath>

#include <iostream>

// private

char digitToString[11] = { '0' , '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A'};

char DigitToString(int num)

{

if (num < 11 && num >= 0)

return digitToString[num];

throw "Can't convert digit to string";

}

int StringToDigit(char digit)

{

int index = std::find(digitToString, digitToString + 12, digit) - digitToString;

if (index < 11)

return index;

throw "Can't convert string to digit";

}

// Get integral part of num

std::string DecToBaseInt(int intergral, int base)

{

if (base <= 0)

throw "Base less then 0";

std::string res = "";

while (intergral > base)

{

try

{

res += DigitToString(intergral % base);

}

catch (const char\* err\_message) { throw err\_message; }

intergral /= base;

}

res += DigitToString(intergral);

std::reverse(res.begin(), res.end());

return res;

}

// Get fractional part of num from 10 to p (p - base)

std::string DecToBaseFrac(float fractal, int base, int accuracy)

{

if (base <= 0)

throw "Base less then 0";

std::string res = "";

int iter = 0;

while (iter < accuracy)

{

fractal \*= base;

try

{

res += DigitToString((int)fractal);

}

catch (const char\* err\_message) { throw err\_message; }

fractal -= (int)fractal;

iter++;

}

return res;

}

// public

std::string DecToBase(float num, int accuracy)

{

std::string integral = "";

std::string fractal = "";

std::string res = "";

try

{

integral = DecToBaseInt(num, 9);

fractal = DecToBaseFrac(num - (int)num, 9, accuracy);

res = integral + '.' + fractal;

}

catch (const char\* err\_message)

{

std::cout << "Can't convert dec to base: " << err\_message << std::endl;

throw "Can't convert dec to base";

}

return res;

}

float BaseToDec(std::string num, int accuracy)

{

float res = 0;

int iter = num.find('.');

if(iter == std::string::npos)

throw "Incorrect num";

iter--;

if (num[0] == '.')

throw "Incorrect num";

for (size\_t i = 0; i < num.length(); i++)

{

if (std::string(digitToString).find(num[i]) == std::string::npos && num[i] != '.')

throw "Incorrect num";

}

for (size\_t i = 0; i < num.length(); i++)

{

char curDigitS = num[i];

if (curDigitS == '.')

continue;

int curDigitI = 0;

try

{

curDigitI = StringToDigit(curDigitS);

}

catch (const char\* err\_message) { throw err\_message; }

res += curDigitI \* pow(11, iter);

iter--;

}

return round(res \* pow(10, accuracy)) / pow(10, accuracy);

}

std::string BaseToBase(std::string num, int accuracy)

{

std::string decToBase = "";

float baseToDec = 0;

try

{

baseToDec = BaseToDec(num, accuracy);

decToBase = DecToBase(baseToDec, accuracy);

}

catch (const char\* err\_message)

{

std::cout << "Can't convert base to base: " << err\_message << std::endl;

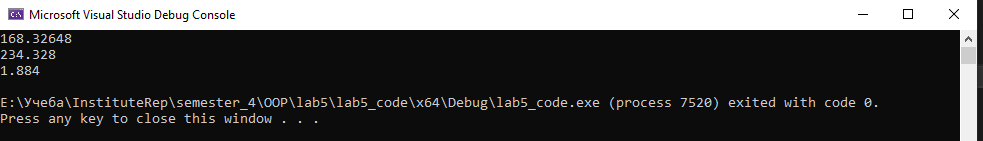
throw "Can't convert base to base";

}

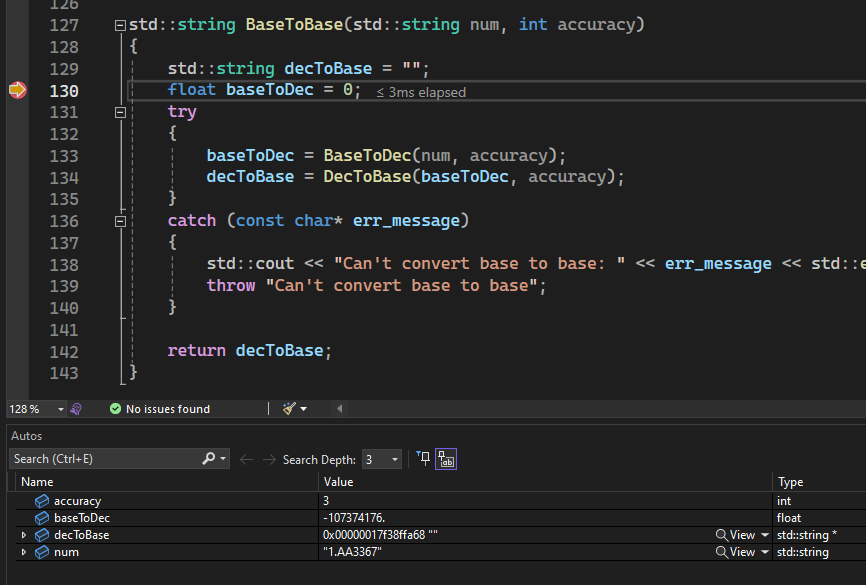
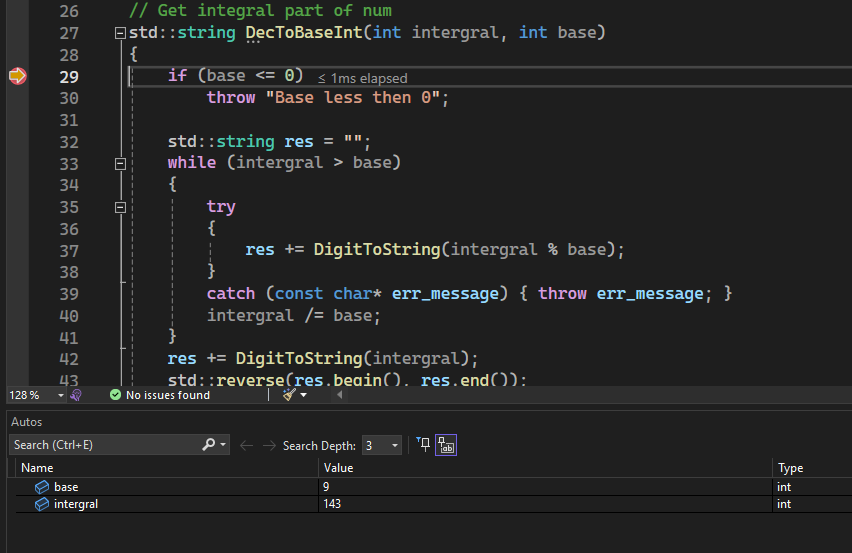
return decToBase;

}

**Результат работы программы**



**Протокол трассировки программы**



**Вывод**

Мы изучили механизмы исключений C++