Лабораторная работа №7

*Программирование обработки исключительных ситуаций*

Цель работы:

Изучить синтаксис и семантику определения и вызова исключений, синтаксис обработчика и спецификации исключений; приобрести практические навыки запуска исключений; изучить особенности применения стандартных библиотечных исключений.

Задание:

Используя модифицированные классы из 5 или 6 лабораторной работы, обработайте все возможные исключительные ситуации. Проверьте работоспособность АТД на тестовом наборе данных.

Возможные исключительные ситуации:

* ввод отрицательных значений там, где они не могут быть (скорость, дальность полёта и другие);
* попытка использования не выделенной памяти с помощью оператора new.

Ход работы

Ниже представлен блок try, в котором содержится вся работа с объектами класса. В случае исключительной ситуации управление будет передано в соответствующий блок catch. Вся конструкция try-catch зациклена, чтобы после возникновения исключительной ситуации можно было попытаться выполнить действия соответствующим образом.

do {

try

{

exception = false;

Plane<int, double> my\_plane1, my\_plane2;

cout << "Введите информацию о первом самолёте:" << endl;

my\_plane1.input();

my\_plane2 = my\_plane1;

Plane<int, double> my\_plane3(my\_plane2);

cout << endl << "Информация о первом самолёте:" << endl;

my\_plane1.print();

cout << endl << "Информация о втором самолёте (копия первого с помощью оператора =):" << endl;

my\_plane2.print();

cout << endl << "Информация о третьем самолёте (копия второго с помощью копирующего конструктора):" << endl;

my\_plane3.print();

//вызов методов базового класса

cout << endl << "Изменение информации о третьем самолёте:" << endl;

my\_plane3.Air2::input();

my\_plane3.set\_information(); //можно без полного имени - PersonalTransport::set\_information()

cout << endl << "Новая информация по третьему самолёту:" << endl;

my\_plane3.Air2::print();

my\_plane3.get\_information();

cout << endl;

}

catch (bad\_alloc& e) {

cout << "Память под строку не выделена. Введите ещё раз." << endl;

exception = true;

}

catch (bad\_value& e) {

cout << "Введено несоответствующее значение. Введите ещё раз." << endl;

exception = true;

}

} while (exception == true);

Генерация исключительных ситуаций происходит в теле конструкторов с параметрами, а также в функция ввода информации, где могут быть переданы несоответствующие значения. Ниже представлено несколько функций, где это показано:

template <class T, class P>

void Plane<T, P>::input()

{

Air2::input();

PersonalTransport::set\_information();

cout << "Введите вместимость самолёта (кол-во человек): ";

cin >> capacity;

if (capacity < 0) throw bad\_value();

}

void PersonalTransport::set\_information() {

cout << "Введите имя вашего водителя/пилота: ";

cin >> drivers\_name;

cout << "Введите траты на транспорт в месяц: ";

cin >> service\_a\_month;

if (service\_a\_month < 0) throw bad\_value();

}

template <class T, class P>

void Air2<T, P>::input()

{

cout << "Введите тип (military или public): ";

cin >> type;

cout << "Введите дистанцию полёта (км): ";

cin >> distance\_of\_flight;

if (distance\_of\_flight < 0) throw bad\_value();

cout << "Введите максимальную высоту полёта (км): ";

cin >> max\_altitude;

if (max\_altitude < 0) throw bad\_value();

cout << "Введите максимальную скорость (км/ч): ";

cin >> max\_spead;

if (max\_spead < 0) throw bad\_value();

}

Полный код программы:

#include <iostream>

#include <stdio.h>

#include <new>

using namespace std;

class bad\_value : public exception {};

template <class T, class P = int>

class Air2 {

public:

Air2(); //по умолчанию

Air2(const Air2& temp); //копирующий

Air2(const char\* str, const T d, const T a, const P s); //с параметрами

~Air2();

void input();

void print() const;

Air2<T, P> operator+(const Air2<T, P>& obj);

Air2<T, P>& operator=(const Air2<T, P>& obj);

Air2<T, P>& operator+=(const Air2<T, P>& obj);

bool operator==(const Air2<T, P>& obj) const;

bool operator!=(const Air2<T, P>& obj) const;

protected:

char\* type = new char[10]; //military or public

T distance\_of\_flight; //дистанция

T max\_altitude; //высота полета

P max\_spead;

};

template <class T, class P>

Air2<T, P>::Air2()

{

distance\_of\_flight = 425.5;

max\_altitude = 11.4;

max\_spead = 521.2;

delete[] type;

type = new char[7];

strcpy(type, "public");

}

template <class T, class P>

Air2<T, P>::Air2(const Air2<T, P>& temp)

{

if (&temp != this) {

distance\_of\_flight = temp.distance\_of\_flight;

max\_altitude = temp.max\_altitude;

max\_spead = temp.max\_spead;

delete[] type;

type = new char[strlen(temp.type) + 1];

strcpy(type, temp.type);

}

}

template <class T, class P>

Air2<T, P>::Air2(const char\* str, const T d, const T a, const P s)

{

distance\_of\_flight = d;

if (distance\_of\_flight < 0) throw bad\_value();

max\_altitude = a;

if (max\_altitude < 0) throw bad\_value();

max\_spead = s;

if (max\_spead < 0) throw bad\_value();

delete[] type;

type = new char[strlen(str) + 1];

strcpy(type, str);

}

template <class T, class P>

Air2<T, P>::~Air2()

{

delete[] type;

}

template <class T, class P>

void Air2<T, P>::input()

{

cout << "Введите тип (military или public): ";

cin >> type;

cout << "Введите дистанцию полёта (км): ";

cin >> distance\_of\_flight;

if (distance\_of\_flight < 0) throw bad\_value();

cout << "Введите максимальную высоту полёта (км): ";

cin >> max\_altitude;

if (max\_altitude < 0) throw bad\_value();

cout << "Введите максимальную скорость (км/ч): ";

cin >> max\_spead;

if (max\_spead < 0) throw bad\_value();

}

template <class T, class P>

void Air2<T, P>::print() const

{

cout << type << endl;

cout << "Дистанция полёта: " << distance\_of\_flight << " км\n";

cout << "Максимальная высота: " << max\_altitude << " км\n";

cout << "Максимальная скорость: " << max\_spead << " км/ч\n";

}

template <class T, class P>

Air2<T, P> Air2<T, P>::operator+(const Air2<T, P>& obj)

{

Air2 temp;

temp.distance\_of\_flight = distance\_of\_flight + obj.distance\_of\_flight;

temp.max\_altitude = max\_altitude + obj.max\_altitude;

temp.max\_spead = max\_spead + obj.max\_spead;

char\* tmp = new char[strlen(type) + strlen(obj.type) + 2];

strcpy(tmp, type);

strcat(tmp, "-");

delete[] temp.type;

temp.type = new char[strlen(tmp) + strlen(obj.type) + 1];

strcpy(temp.type, strcat(tmp, obj.type));

delete[] tmp;

return temp;

}

template <class T, class P>

Air2<T, P>& Air2<T, P>::operator=(const Air2<T, P>& obj)

{

distance\_of\_flight = obj.distance\_of\_flight;

max\_altitude = obj.max\_altitude;

max\_spead = obj.max\_spead;

delete[] type;

type = new char[strlen(obj.type) + 1];

strcpy(type, obj.type);

return \*this;

}

template <class T, class P>

Air2<T, P>& Air2<T, P>::operator+=(const Air2<T, P>& obj)

{

distance\_of\_flight += obj.distance\_of\_flight;

max\_altitude += obj.max\_altitude;

max\_spead += obj.max\_spead;

char\* tmp = new char[strlen(type) + strlen(obj.type) + 2];

strcpy(tmp, type);

strcat(tmp, "-");

delete[] type;

type = new char[strlen(tmp) + strlen(obj.type) + 1];

strcpy(type, strcat(tmp, obj.type));

delete[] tmp;

return \*this;

}

template <class T, class P>

bool Air2<T, P>::operator==(const Air2<T, P>& obj) const

{

return (strcmp(type, obj.type) == 0) ? true : false;

}

template <class T, class P>

bool Air2<T, P>::operator!=(const Air2<T, P>& obj) const

{

return (strcmp(type, obj.type) == 0) ? false : true;

}

//------------------------------class PersonalTransport-----------------------------

class PersonalTransport {

public:

PersonalTransport();

PersonalTransport(const char\* name, const double price);

PersonalTransport(const PersonalTransport& obj);

~PersonalTransport();

void set\_information();

void get\_information() const;

PersonalTransport& operator=(const PersonalTransport& obj);

protected:

char\* drivers\_name = new char[10];

double service\_a\_month;

};

PersonalTransport::PersonalTransport() {

delete[] drivers\_name;

drivers\_name = new char[10];

strcpy(drivers\_name, "Alex");

service\_a\_month = 10000;

}

PersonalTransport::PersonalTransport(const char\* name, const double price) {

delete[] drivers\_name;

drivers\_name = new char[strlen(name) + 1];

strcpy(drivers\_name, name);

if (price < 0) throw bad\_value();

service\_a\_month = price;

}

PersonalTransport::PersonalTransport(const PersonalTransport& obj) {

delete[] drivers\_name;

drivers\_name = new char[strlen(obj.drivers\_name) + 1];

strcpy(drivers\_name, obj.drivers\_name);

service\_a\_month = obj.service\_a\_month;

}

PersonalTransport::~PersonalTransport() {

delete[] drivers\_name;

}

void PersonalTransport::set\_information() {

cout << "Введите имя вашего водителя/пилота: ";

cin >> drivers\_name;

cout << "Введите траты на транспорт в месяц: ";

cin >> service\_a\_month;

if (service\_a\_month < 0) throw bad\_value();

}

void PersonalTransport::get\_information() const {

cout << "Имя вашего водителя/пилота: " << drivers\_name << endl;

cout << "Траты на транспорт в месяц: " << service\_a\_month << '$' << endl;

}

PersonalTransport& PersonalTransport::operator=(const PersonalTransport& obj) {

delete[] drivers\_name;

drivers\_name = new char[strlen(obj.drivers\_name) + 1];

strcpy(drivers\_name, obj.drivers\_name);

service\_a\_month = obj.service\_a\_month;

return \*this;

}

//-----------------------------class Plane<T,P>---------------------------------------

template <class T, class P>

class Plane : public Air2<int, double>, public PersonalTransport {

public:

Plane();

Plane(const char\* str, const int d, const int a, const double s, const char\* name, const double price, const int capacity);

Plane(const Plane<T, P>& obj);

~Plane();

void input();

void print() const;

Plane<T, P>& operator=(const Plane<T, P>& obj);

protected:

int capacity; //вместимость

};

template <class T, class P>

Plane<T, P>::Plane() : Air2(), PersonalTransport()

{

capacity = 2;

}

template <class T, class P>

Plane<T, P>::Plane(const char\* str, const int d, const int a, const double s, const char\* name, const double price, const int capacity) : Air2(str, d, a, s), PersonalTransport(name, price)

{

if (capacity < 0) throw bad\_value();

this->capacity = capacity;

}

template <class T, class P>

Plane<T, P>::Plane(const Plane<T, P>& obj)

{

if (&obj != this) {

//data 1st base class

distance\_of\_flight = obj.distance\_of\_flight;

max\_altitude = obj.max\_altitude;

max\_spead = obj.max\_spead;

delete[] type;

type = new char[strlen(obj.type) + 1];

strcpy(type, obj.type);

//data 2nd base class

delete[] drivers\_name;

drivers\_name = new char[strlen(obj.drivers\_name) + 1];

strcpy(drivers\_name, obj.drivers\_name);

service\_a\_month = obj.service\_a\_month;

//own data

capacity = obj.capacity;

}

}

template <class T, class P>

Plane<T, P>::~Plane()

{

Air2::~Air2();

PersonalTransport::~PersonalTransport();

}

template <class T, class P>

void Plane<T, P>::input()

{

Air2::input();

PersonalTransport::set\_information();

cout << "Введите вместимость самолёта (кол-во человек): ";

cin >> capacity;

if (capacity < 0) throw bad\_value();

}

template <class T, class P>

void Plane<T, P>::print() const

{

Air2::print();

PersonalTransport::get\_information();

cout << "Вместимость самолёта: " << capacity << " человек" << endl;

}

template <class T, class P>

Plane<T, P>& Plane<T, P>::operator=(const Plane<T, P>& obj)

{

//data 1st base class

distance\_of\_flight = obj.distance\_of\_flight;

max\_altitude = obj.max\_altitude;

max\_spead = obj.max\_spead;

delete[] type;

type = new char[strlen(obj.type) + 1];

strcpy(type, obj.type);

//data 2nd base class

delete[] drivers\_name;

drivers\_name = new char[strlen(obj.drivers\_name) + 1];

strcpy(drivers\_name, obj.drivers\_name);

service\_a\_month = obj.service\_a\_month;

//own data

capacity = obj.capacity;

return \*this;

}

int main() {

setlocale(LC\_ALL, "Russian");

bool exception;

//------------------------Plane<int, double>---------------------------

do {

try

{

exception = false;

Plane<int, double> my\_plane1, my\_plane2;

cout << "Введите информацию о первом самолёте:" << endl;

my\_plane1.input();

my\_plane2 = my\_plane1;

Plane<int, double> my\_plane3(my\_plane2);

cout << endl << "Информация о первом самолёте:" << endl;

my\_plane1.print();

cout << endl << "Информация о втором самолёте (копия первого с помощью оператора =):" << endl;

my\_plane2.print();

cout << endl << "Информация о третьем самолёте (копия второго с помощью копирующего конструктора):" << endl;

my\_plane3.print();

//вызов методов базового класса

cout << endl << "Изменение информации о третьем самолёте:" << endl;

my\_plane3.Air2::input();

my\_plane3.set\_information(); //можно без полного имени - PersonalTransport::set\_information()

cout << endl << "Новая информация по третьему самолёту:" << endl;

my\_plane3.Air2::print();

my\_plane3.get\_information();

cout << endl;

}

catch (bad\_alloc& e) {

cout << "Память под строку не выделена. Введите ещё раз." << endl;

exception = true;

}

catch (bad\_value& e) {

cout << "Введено несоответствующее значение. Введите ещё раз." << endl;

exception = true;

}

} while (exception == true);

//------------------------Plane<double, int>----------------------------

do {

try

{

exception = false;

Plane<double, int> my\_plane4, my\_plane5;

cout << "Введите информацию о четвертом самолёте:" << endl;

my\_plane4.input();

my\_plane5 = my\_plane4;

Plane<double, int> my\_plane6(my\_plane5);

cout << endl << "Информация о четвертом самолёте:" << endl;

my\_plane4.print();

cout << endl << "Информация о пятом самолёте (копия четвертого с помощью оператора =):" << endl;

my\_plane5.print();

cout << endl << "Информация о шестом самолёте (копия пятого с помощью копирующего конструктора):" << endl;

my\_plane6.print();

//вызов методов базового класса

cout << endl << "Изменение информации о шестом самолете:" << endl;

my\_plane6.Air2::input();

my\_plane6.set\_information();

cout << endl << "Новая информация о шестом самолете:" << endl;

my\_plane6.Air2::print();

my\_plane6.get\_information();

cout << endl;

}

catch (bad\_alloc& e) {

cout << "Память под строку не выделена. Введите ещё раз." << endl;

exception = true;

}

catch (bad\_value& e) {

cout << "Введено несоответствующее значение. Введите ещё раз." << endl;

exception = true;

}

} while (exception == true);

//------------------------Plane<double, double>-------------------------

do {

try

{

exception = false;

Plane<double, double> my\_plane7, my\_plane8;

cout << "Введите информацию о седьмом самолёте:" << endl;

my\_plane7.input();

my\_plane8 = my\_plane7;

Plane<double, double> my\_plane9(my\_plane8);

cout << endl << "Информация о седьмом самолёте:" << endl;

my\_plane7.print();

cout << endl << "Информация о восьмом самолёте (копия седьмого с помощью оператора =):" << endl;

my\_plane8.print();

cout << endl << "Информация о девятом самолёте (копия восьмого с помощью копирующего конструктора):" << endl;

my\_plane9.print();

//вызов методов базового класса

cout << endl << "Изменение информации о девятом самолете:" << endl;

my\_plane9.Air2::input();

my\_plane9.set\_information();

cout << endl << "Новая информация о девятом самолете:" << endl;

my\_plane9.Air2::print();

my\_plane9.get\_information();

cout << endl;

}

catch (bad\_alloc& e) {

cout << "Память под строку не выделена. Введите ещё раз." << endl;

exception = true;

}

catch (bad\_value& e) {

cout << "Введено несоответствующее значение. Введите ещё раз." << endl;

exception = true;

}

} while (exception == true);

//------------------------Plane<int, int>-------------------------------

do {

try

{

exception = false;

Plane<int, int> my\_plane10, my\_plane11;

cout << "Введите информацию о десятом самолёте:" << endl;

my\_plane10.input();

my\_plane11 = my\_plane10;

Plane<int, int> my\_plane12(my\_plane11);

cout << endl << "Информация о десятом самолёте:" << endl;

my\_plane10.print();

cout << endl << "Информация о одиннадцитом самолёте (копия десятого с помощью оператора =):" << endl;

my\_plane11.print();

cout << endl << "Информация о двенадцатом самолёте (копия одиннадцатого с помощью копирующего конструктора):" << endl;

my\_plane12.print();

//вызов методов базового класса

cout << endl << "Изменение информации о двенадцатом самолёте:" << endl;

my\_plane12.Air2::input();

my\_plane12.set\_information();

cout << endl << "Новая информация о двенадцатом самолёте:" << endl;

my\_plane12.Air2::print();

my\_plane12.get\_information();

cout << endl << "Информация о двенадцатом самолёте (после преобразования):" << endl;

my\_plane12.print();

}

catch (bad\_alloc& e) {

cout << "Память под строку не выделена. Введите ещё раз." << endl;

exception = true;

}

catch (bad\_value& e) {

cout << "Введено несоответствующее значение. Введите ещё раз." << endl;

exception = true;

}

} while (exception == true);

std::system("pause");

return 0;

}

Результат работы программы:

Ответы на контрольные вопросы

1.Дайте определение исключения.

Это возникающая в программе нештатная ситуация, с которой программа не может справиться. Например, при делении на ноль выполнение программы аварийно завершается системой. С++ дает возможность восстанавливать программу из ошибочных ситуаций и продолжать ее выполнение.

2.В каком блоке описывается список обработчиков для возбужденного исключения?

Блок try является контекстом для определения того, какие обработчики вызываются при возбуждении исключения. Порядок, в котором определены обработчики, задает очередность проверки обработчиков подходящего типа для возбуждения исключения.

3. Опишите синтаксис обработчика исключения.

*catch (формальный\_аргумент) { составная\_конструкция }*

Обработчик catch выглядит как объявление функции одного аргумента без возвращаемого типа. Например:

*catch (const char\* message)*

*{*

*cerr << message << endl;*

*exit (1);*

*}*

*catch (…) //будет выполнено действие по умолчанию*

*{*

*cerr << “Вот и все!” << endl;*

*abort();*

*}*

4.Что такое спецификация исключения?

Синтаксически спецификация исключения является частью объявления и определения функции и имеет следующий вид:

*заголовок\_функции throw (список\_типов)*

Здесь список\_типов – это список типов, которые может иметь выражение throw внутри функции. В объявлении и в определении функции спецификация исключения должна записываться одинаково. Если список пуст, компилятор полагает, что функцией не будет выполняться никакой throw (ни прямо, ни косвенно). Если спецификация исключения опущена, то предполагается, что такой функцией может быть возбуждено произвольное исключение.

5.Для каких целей используется функция unexpected?

Предоставляемая системой функция unexpected() вызывается, когда она возбудила исключение, которое отсутствует в ее списке спецификации исключений