ЛАБОРАТОРНАЯ РАБОТА № 5

Тема работы: Генерация и обработка исключительных ситуаций.

Содержание работы: Разработка собственных классов обработки исключений. Обработчики исключительных ситуаций. Перенаправление исключительных ситуаций. Исключения стандартной библиотеки.

Цель работы: Понять, как обрабатываются исключения. Использовать try, throw и catch для отслеживания, индикации и обработки исключений. Написать свои функции для проверки правильности ввода.

Теория рассмотрена в соответствующих разделах[1, 3, 4, 5].

Исключения в C++

Исключения – возникновение непредвиденных условий, например, деление на ноль, невозможность выделения памяти при создании нового объекта и т.д. Обычно эти условия завершают выполнение программы с системной ошибкой. C++ позволяет восстанавливать программу из этих условий и продолжать ее выполнение. В то же время исключение – это более общее, чем ошибка, понятие и может возникать и тогда, когда в программе нет ошибок.

Механизм обработки исключительных ситуаций является неотъемлемой частью языка C++.Этот механизм предоставляет программисту средство реагирования на нештатные события и позволяет преодолеть ряд принципиальных недостатков следующих традиционных методов обработки ошибок:

-возврат функцией кода ошибки;

-возврат значений ошибки через аргументы функций;

-использование глобальных переменных ошибки;

-использование оператора безусловного перехода goto или функций setjmp / longjmp;

-использование макроса assert.

Возврат функцией кода ошибки является самым обычным и широко применяемым методом. Однако этот метод имеет существенные недостатки. Во - первых, нужно помнить численные значения кодов ошибок. Эту проблему можно обойти, используя перечисляемые типы. Но в некоторых случаях функция может возвращать широкий диапазон допустимых(неошибочных) значений, и тогда сложно найти диапазон для возвращаемых кодов ошибки. Это и является вторым недостатком. И, в - третьих, при использовании такого механизма сигнализации об ошибках вся ответственность по их обработке ложится на программиста и могут возникнуть ситуации, когда серьезные ошибки могут остаться без внимания.

Возврат кода ошибки через аргумент функции или использование глобальной переменной ошибки снимают прежде всего вторую проблему,

однако по - прежнему остаются первая и третья. Кроме того, использование глобальных переменных не является особо позитивным фактором.

Использование оператора безусловного перехода в любых ситуациях является нежелательным, кроме того, оператор goto действует только в пределах функции. Пара функций setjmp / longjmp является довольно мощным средством, однако и этот метод имеет серьезнейший недостаток : он не обеспечивает вызов деструкторов локальных объектов при выходе из области видимости, что, естественно, влечет за собой утечку памяти.

И, наконец, макрос assert является скорее средством отладки, чем средством обработки нештатных событий, возникающих в процессе использования программы.

Таким образом, необходим некий другой способ обработки ошибок, который учитывал бы объектно - ориентированную философию. Таким способом и является механизм обработки исключительных ситуаций.

Основы обработки исключительных ситуаций

Обработка исключительных ситуаций лишена недостатков вышеназванных методов реагирования на ошибки. Этот механизм позволяет использовать для представления информации об ошибке объект любого типа. Поэтому можно, например, создать иерархию классов, которая будет предназначена для обработки аварийных событий. Это упростит, структурирует и сделает более понятной программу.

Рассмотрим пример обработки исключительных ситуаций. Функция div() возвращает частное от деления чисел, принимаемых в качестве аргументов. Если делитель равен нулю, то генерируется исключительная ситуация.

```
#include<iostream>
using namespace std;

double div(double dividend, double divisor)
{
   if (divisor == 0)  throw 1;
   return dividend / divisor;
}

int main()
{
   double result;
   try {

         result = div(77., 0.);
         cout << "Answer is " << result << endl;
   }
   catch (int) {
         cout << "Division by zero" << endl;
   }
   return 0;
```

```
}
```

Результат выполнения программы:

```
Division by zero
```

В данном примере необходимо выделить три ключевых элемента. Во - первых, вызов функции div() заключен внутрь блока, который начинается с ключевого слова try. Этот блок указывает, что внутри него могут происходить исключительные ситуации. По этой причине код, заключенный внутри блока try, иногда называют охранным.

Далее за блоком try следует блок catch, называемый обычно обработчиком исключительной ситуации. Если возникает исключительная ситуация, выполнение программы переходит к этому catch - блоку. Хотя в этом примере имеется один - единственный обработчик, их в программах может быть множество и они способны обрабатывать множество различных типов исключительных ситуаций.

Еще одним элементом процесса обработки исключительных ситуаций является оператор throw (в данном случае он находится внутри функции div()).Оператор throw сигнализирует об исключительном событии и генерирует объект исключительной ситуации, который перехватывается обработчиком catch. Этот процесс называется вызовом исключительной ситуации. В рассматриваемом примере исключительная ситуация имеет форму обычного целого числа, однако программы могут генерировать практически любой тип исключительной ситуации.

Если в инструкции if (divisor == 0) throw 1 значение 1 заменить на 1., то при выполнении будет выдана ошибка об отсутствии соответствующего обработчика catch (так как возбуждается исключительная ситуация типа double).

Одним из главных достоинств использования механизма обработки исключительных ситуаций является обеспечение развертывания стека. Развертывание стека – это процесс вызова деструкторов локальных объектов, когда исключительные ситуации выводят их из области видимости.

Сказанное рассмотрим на примере функции add() класса add_class, выполняющей сложение компонентов - данных объектов add_class и возвращающей суммарный объект. В случае, если сумма превышает максимальное значение для типа unsigned short, генерируется исключительная ситуация.

```
#include<iostream>
using namespace std;
#include<limits.h>
class Add_class

{
```

```
private:
   unsigned short num;
public:
   Add_class(unsigned short a)
   {
        num = a;
        cout << "Constructor " << num << endl;
   }
   ~Add_class() { cout << "Destructor of add_class " << num << endl;
}
   void show_num() { cout << " " << num << " "; }
   void input_num(unsigned short a) { num = a; }
   unsigned short output_num() { return num; }
};
Add_class add(add_class a, add_class b)   // суммирование компонент
num
{
   Add_class sum(0);                                    // объектов a
и b
   unsigned long s = (unsigned long)a.output_num() +
        (unsigned long)b.output_num();
   if (s > USHRT_MAX) throw 1;       // генерация исключения типа
int
   sum.input_num((unsigned short)s);
   return sum;
}
int main()
{
   add_class a(USHRT_MAX), b(1), s(0);
   try {                    // охранный блок
        s = add(a, b);
        cout << "Result";
```

```
            s.show_num();
            cout << endl;
        }
        catch (int) {          // обработчик исключения типа int
            cout << "Overflow error" << endl;
        }
        return 0;
}
```

Результат выполнения программы:

```
Constructor 65535
Constructor 1
Constructor 0
Constructor 0
Destructor of add_class 0
Destructor of add_class 65535
Destructor of add_class 1
Overflow error
Destructor of add_class 0
Destructor of add_class 1
Destructor of add_class 65535
```

Сначала вызываются конструкторы объектов a, b и s, далее происходит передача параметров по значению в функцию. В этом случае происходит вызов конструктора копий(сначала для объекта b затем для a), созданного по умолчанию, именно поэтому вызовов деструктора больше, чем конструктора, затем, используя конструктор, создается объект sum. После этого генерируется исключение и срабатывает механизм развертывания стека, т.е. вызываются деструкторы локальных объектов sum, a и b.И, наконец, вызываются деструкторы s, b и a.

Рассмотрим более подробно элементы try, catch и throw механизма обработки исключений.

Блок try. Синтаксис блока :

```
try {

охранный код

}

список обработчиков
```

Необходимо помнить, что после ключевого слова try всегда должен следовать составной оператор, т.е.после try всегда следует{ … }.Блоки try не имеют однострочной формы, как, например, операторы if, while, for.

Еще один важный момент заключается в том, что после блока try должен следовать, по крайней мере, хотя бы один обработчик. Недопустимо нахождение между блоками try и catch какого - либо кода. Например:

```
int i;

try {

   throw исключение;

}

i = 0;                        //   'try' block starting on line '
номер  ' has no catch handlers

catch (тип аргумент) {

   блок обработки исключения

}
```

В блоке try можно размещать любой код, вызовы локальных функций, функции - компоненты объектов, и любой код любой степени вложенности может генерировать исключительные ситуации. Блоки try сами могут быть вложенными.

Обработчики исключительных ситуаций catch. Обработчики исключительных ситуаций являются важнейшей частью всего механизма обработки исключений, так как именно они определяют поведение программы после генерации и перехвата исключительной ситуации .Синтаксис блока catch имеет следующий вид :

```
catch (тип 1 < аргумент > ) {

   тело обработчика

}
```

```
catch (тип 2 < аргумент > )){

тело обработчика

}

.   .   .

catch (тип N <аргумент>)){

тело обработчика

}
```

Таким образом, так же как и в случае блока try, после ключевого слова catch должен следовать составной оператор, заключенный в фигурные скобки. В аргументах обработчика можно указать только тип исключительной ситуации, необязательно объявлять имя объекта, если этого не требуется.

У каждого блока try может быть множество обработчиков, каждый из которых должен иметь свой уникальный тип исключительной ситуации. Неправильной будет следующая запись :

```
typedef int type_int;

try { . . . }


catch (type_int error1) {

   . . .

}

catch (int error2) {

   . . .

}
```

Так, в этом случае type_int и int – это одно и то же.

Таким образом пример верен.

```
class cls

{
public:

   int i;

};
```

```
try {
    . . .
}
catch (cls i1) {
    . . .
}
catch (int i2) {
}
```

В этом случае cls – это отдельный тип исключительной ситуации. Существует также абсолютный обработчик, который совместим с любым типом исключительной ситуации. Для написания такого обработчика надо вместо аргументов написать многоточие(эллипсис).

```
catch (…) {
    блок обработки исключения
}
```

Использование абсолютного обработчика исключительных ситуаций рассмотрим на примере программы, в которой происходит генерация исключительной ситуации типа char *, но обработчик такого типа отсутствует. В этом случае управление передается абсолютному обработчику.

```
#include <iostream>
using namespace std;
void int_exception(int i)
{
    if (i > 100) throw 1;      // генерация исключения типа  int
}
void string_exception()
{
    throw "Error";            // генерация исключения типа  char *
}
int main()
```

```
{
   try {          //  в блоке возможна обработка одного из двух
исключений

         int_exception(99);      //  возможно исключение типа int

         string_exception();     //  возможно исключение типа char *

   }

   catch (int) {

         cout << "Обработчик для типа int" << endl;

   }

   catch (...) {

         cout << "Абсолютный обработчик " << endl;

   }

}
```

Результат выполнения программы :

```
Абсолютный обработчик
```

Так как абсолютный обработчик перехватывает исключительные ситуации всех типов, то он должен стоять в списке обработчиков последним. Нарушение этого правила вызовет ошибку при компиляции программы.

Для того чтобы эффективно использовать механизм обработки исключительных ситуаций, необходимо грамотно построить списки обработчиков, а для этого, в свою очередь, нужно четко знать следующие правила, по которым осуществляется поиск соответствующего обработчика:

-исключительная ситуация обрабатывается первым найденным обработчиком, т.е. если есть несколько обработчиков, способных обработать данный тип исключительной ситуации, то она будет обработана первым стоящим в списке обработчиком;

-абсолютный обработчик может обработать любую исключительную ситуацию;

-исключительная ситуация может быть обработана обработчиком соответствующего типа либо обработчиком ссылки на этот тип;

-исключительная ситуация может быть обработана обработчиком базового для нее класса. Например, если класс B является производным от класса A, то обработчик класса A может обработать исключительную ситуацию класса B;

-исключительная ситуация может быть обработана обработчиком, принимающим указатель, если тип исключительной ситуации может быть

приведен к типу обработчика путем использования стандартных правил преобразования типов указателей.

Если при возникновении исключительной ситуации подходящего обработчика нет среди обработчиков данного уровня вложенности блоков try, то обработчик ищется на следующем охватывающем уровне. Если обработчик не найден вплоть до самого верхнего уровня, то программа аварийно завершается.

Следствием из правил 3 и 4 является еще одно утверждение : исключительная ситуация может быть направлена обработчику, который может принимать ссылку на объект базового для данной исключительной ситуации класса. Это значит, что если, например, класс B – производный от класса A, то обработчик ссылки на объект класса A может обрабатывать исключительную ситуацию класса B(или ссылку на объект класса B).

Рассмотрим особенности выбора соответствующего обработчика на следующем примере. Пусть имеется класс C, являющийся производным от классов A и B; показано, какими обработчиками может быть перехвачена исключительная ситуация типа C и типа указателя на C.

```
#include<iostream.h>

using namespace std;

class A {};

class B {};

class C : public A, public B {};

void f(int i)

{
   if (i) throw C();      // возбуждение исключительной ситуации

                                          // типа «объект класса C»

   else throw new C; //  возбуждение исключительной ситуации

                                           // типа «указатель на
объект класса C»

}

int main()

{
   int i;

   try {

         cin >> i;
```

```
        f(i);
    }
    catch (A) {
        cout << "A handler";
    }
    catch (B&) {
        cout << "B& handler";
    }
    catch (C) {
        cout << "C handler";
    }
    catch (C*) {
        cout << "C* handler";
    }
    catch (A*) {
        cout << "A* handler";
    }
    catch (void*) {
        cout << "void* handler";
    }
}
```

В данном примере исключительная ситуация класса C может быть направлена любому из обработчиков A, B& или C, поэтому выбирается обработчик, стоящий первым в списке. Аналогично для исключительной ситуации, имеющей тип указателя на объект класса C, выбирается первый подходящий обработчик A* или C*.Эта ситуация также может быть обработана обработчиками void*.Так как к типу void* может быть приведен любой указатель, то обработчик этого типа будет перехватывать любые исключительные ситуации типа указателя. Рассмотрим еще один пример :

```
#include <iostream.h>
using namespace std;
```

```
class base {};

class derived : public base {};

void fun()

{
   derived obj;

   base &a = obj;

   throw a;

}

int main()

{
   try {
         fun();

   }

   catch (derived) {
         cout << ”derived” << endl;

   }

   catch (base) {
         cout << ”base” << endl;

   }

}
```

Результат выполнения программы :

```
base
```

В примере генерируется исключение, имеющее тип base, хотя ссылка a имеет тип base, хотя является ссылкой на класс derived. Так происходит потому, что статический тип a равен base, а не derived.

В случае, когда генерация исключения выполняется по значению объекта или по ссылке на объект, происходит копирование значения объекта(локального) во временный(в catch).

Генерация исключительных ситуаций throw. Исключительные ситуации передаются обработчикам с помощью ключевого слова throw. Как ранее отмечалось, обеспечивается вызов деструкторов локальных объектов при выходе из области видимости, т.е. развертывание стека. Однако развертывание стека не обеспечивает уничтожения объектов, созданных

динамически. Таким образом, перед генерацией исключительной ситуации необходимо явно освободить динамически выделенные блоки памяти.

Следует отметить также, что если исключительная ситуация генерируется по значению или по ссылке, то создается скрытая временная переменная, в которой хранится копия генерируемого объекта. Когда после throw указывается локальный объект, то к моменту вызова соответствующего обработчика этот объект будет уже вне области видимости и, естественно, прекратит существование. Обработчик же получит в качестве аргумента именно эту скрытую копию. Из этого следует, что если генерируется исключительная ситуация сложного класса, то возникает необходимость снабжения этого класса конструктором копий, который бы обеспечил корректное создание копии объекта.

Если же исключительная ситуация генерируется с использованием указателя, то копия объекта не создается. В этом случае могут возникнуть проблемы. Например, если генерируется указатель на локальный объект, к моменту вызова обработчика объект уже перестанет существовать и использование указателя в обработчике может привести к ошибкам.

```
class A

{
   int i;

public:

   A() :i(0) {}

   void ff(int i) { this->i = i; }

};

void f()

{
   A obj;

   .   .   .

         throw &obj;

}

int main()

{
   try {

         f();

   }

   catch (A *a) {   //   код обработчика;
```

```
        a->ff(1);
    }
}
```

Эту проблему можно легко устранить, если поместить указатель на новый динамический объект :

```
void f()
{
    A obj;
    .   .   .
        throw new A;
}
```

При этом необходимо решить вопрос о необходимости удаления динамически созданного объекта исключения(например в блоке catch).В противном случае возможны утечки памяти.

(Продолжение)

Вопросы:

-Перенаправление исключительных ситуаций.

- Исключительная ситуация, генерируемая оператором new

- Генерация исключений в конструкторах.

Перенаправление исключительных ситуаций

Иногда возникает положение, при котором необходимо обработать исключительную ситуацию сначала на более низком уровне вложенности блока try, а затем передать ее на более высокий уровень для продолжения обработки. Для того чтобы сделать это, нужно использовать throw без аргументов. В этом случае исключительная ситуация будет перенаправлена к следующему подходящему обработчику(подходящий обработчик не ищется ниже в текущем списке − сразу осуществляется поиск на более высоком уровне). Приводимый ниже пример демонстрирует организацию такой передачи. Программа содержит вложенный блок try и соответствующий блок catch. Сначала происходит первичная обработка, затем исключительная ситуация перенаправляется на более высокий уровень для дальнейшей обработки.

```
#include<iostream>
using namespace std;
void func(int i)
```

```
{
   try {
         if (i) throw "Error";
   }
   catch (char *s) {
         cout << s << "-  выполняется первый обработчик" << endl;
         throw;
   }
}
int main()
{
   try {
         func(1);
   }
   catch (char *s) {
         cout << s << "- выполняется второй обработчик" << endl;
   }
}
```

Результат выполнения программы :

```
Error - выполняется первый обработчик
Error - выполняется второй обработчик
```

Если ключевое слово trow используется вне блока catch, то автоматически будет вызвана функция terminate(), которая по умолчанию завершает программу.

Исключительная ситуация, генерируемая оператором new

Следует отметить, что некоторые компиляторы поддерживают генерацию исключений в случае ошибки выделения памяти посредством оператора new, в частности исключения типа bad_alloc. Оно может быть перехвачено и необходимым образом обработано. Ниже в программе рассмотрен пример генерации и обработки исключительной ситуаций bad_alloc. Искусственно вызывается ошибка выделения памяти и перехватывается исключительная ситуация.

```
#include <new>

#include <iostream>

using namespace std;

int main()

{
   double *p;

   try {

         while (1) p = new double[100];  // генерация ошибки
выделения памяти

   }

   catch (bad_alloc exept) {                          // обработчик
xalloc

         cout << "Возникло исключение" << exept.what() << endl;

   }

   return 0;

}
```

В случае если компилятором не генерируется исключение bad_alloc, то можно это исключение создать искусственно :

```
#include <new>

#include <iostream>

using namespace std;

int main()

{
   double *p;

   try {

         if (!(p = new double[100000000]))  // память не выделена
p==NULL

               throw bad_alloc;                  // генерация ошибки
выделения памяти

   }

   catch (bad_alloc exept) {            // обработчик  bad_alloc
```

```
        cout << "Возникло исключение   " << exept.what() << endl;
    }
    return 0;
}
```

Результатом работы программы будет сообщение :

```
Возникло исключение   bad  allocation
```

Оператор new появился в языке C++ еще до того, как был введен механизм обработки исключительных ситуаций, поэтому первоначально в случае ошибки выделения памяти этот оператор просто возвращал NULL. Если требуется, чтобы new работал именно так, надо вызвать функцию set_new_handler() с параметром 0. Кроме того, с помощью set_new_handler() можно задать функцию, которая будет вызываться в случае ошибки выделения памяти. Функция set_new_handler(в заголовочном файле new) принимает в качестве аргумента указатель на функцию, которая не принимает никаких аргументов и возвращает void.

```
#include<new>
#include<iostream>
using namespace std;
void newhandler()
{
    cout << "The new_handler is called: ";
    throw bad_alloc();
    return;
}
int main()
{
    char* ptr;
    set_new_handler(newhandler);

    try {
        ptr = new char[~size_t(0) / 2];
        delete[] ptr;
    }
```

```
    catch (bad_alloc &ba) {

          cout << ba.what() << endl;

    }

    return 0;

}
```

Результатом работы программы является

```
The new_handler is called : bad allocation
```

В классе new также определена функция _set_new_handler, аргументом которой является указатель на функцию, возвращающую значение типа int и получающую аргумент типа size_t, указывающий размер требуемой памяти :

```
#include <new>

#include <iostream>

using namespace std;

int error_alloc(size_t) // size_t unsigned integer
                        // integer  результат sizeof  operator.

{
    cout << "ошибка выделения памяти" << endl;

    return 0;

}

int main()

{

    _set_new_handler(error_alloc);

    int *p = new int[10000000000];

    return 0;

}
```

Результатом работы функции является :

```
ошибка выделения памяти
```

В случае, если память не выделяется и не задается никакой функции - аргумента для set_new_handler, оператор new генерирует исключение bad_alloc.

Генерация исключений в конструкторах

Механизм обработки исключительных ситуаций очень удобен для обработки ошибок, возникающих в конструкторах. Так как конструктор не возвращает значения, то соответственно нельзя возвратить некий код ошибки и приходится искать альтернативу. В этом случае наилучшим решением является генерация и обработка исключительной ситуации. При генерации исключения внутри конструктора процесс создания объекта прекращается. Если к этому моменту были вызваны конструкторы базовых классов, то будет обеспечен и вызов соответствующих деструкторов. Рассмотрим на примере генерацию исключительной ситуации внутри конструктора. Пусть имеется класс B, производный от класса A и содержащий в качестве компоненты - данного объект класса local. В конструкторе класса B генерируется исключительная ситуация.

```
#include<iostream>

using namespace std;

class local

{
public:

   local() { cout << "Constructor of local" << endl; }

   ~local() { cout << "Destructor of local" << endl; }

};

class A

{
protected:

   int n;

public:

   A() { cout << "Constructor of A" << endl; }

   virtual  ~A() { cout << "Destructor of A" << endl; }

   virtual void f() = 0;

};

class B : public A

{
public:

   local ob;
```

```
    B(int i = 0)
    {
          cout << "Constructor of B" << endl;
          if (i) throw 1;
          n = i;
    }
    ~B() { cout << "Destructor of B" << endl; }
    void f() { cout << "** " << n << " **" << endl; }
};
int main()
{
    A *p;
    try {
          p = new B(1);
          p->f();
    }
    catch (int) {
          cout << "int exception handler" << endl;
          return 1;
    }
    p->f();
    delete p;
    return 0;
}
```

Результат выполнения программы :

```
Constructor of A
Constructor of local
Constructor of B
int exception handler
```

```
Destructor of local

Destructor of A
```

В программе при создании объекта производного класса B сначала вызываются конструкторы базового класса A, затем класса local, который является компонентом класса B. После этого вызывается конструктор класса B, в котором генерируется исключительная ситуация. При выходе из области видимости(try {}) деструкторы не вызываются, так как объекты являются динамическими. При выполнении инструкции

```
delete p;
```

выполняется вызов деструкторов. Видно, что при этом для всех ранее созданных объектов вызваны деструкторы, а для объекта самого класса B деструктор не вызывается, так как конструирование этого объекта не было завершено.

Если в конструкторах выполнялось динамическое выделение памяти, то при генерации исключительной ситуации выделенная память автоматически освобождена не будет, об этом необходимо заботиться самостоятельно, иначе возникнет утечка памяти.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как работают исключения?
2. Можно ли между блоком try и catch записать произвольный код?
3. Что такое Развертывание(разворачивание) стека?
4. Как повторно сгенерировать то же исключение?
5. Абсолютный обработчик исключений

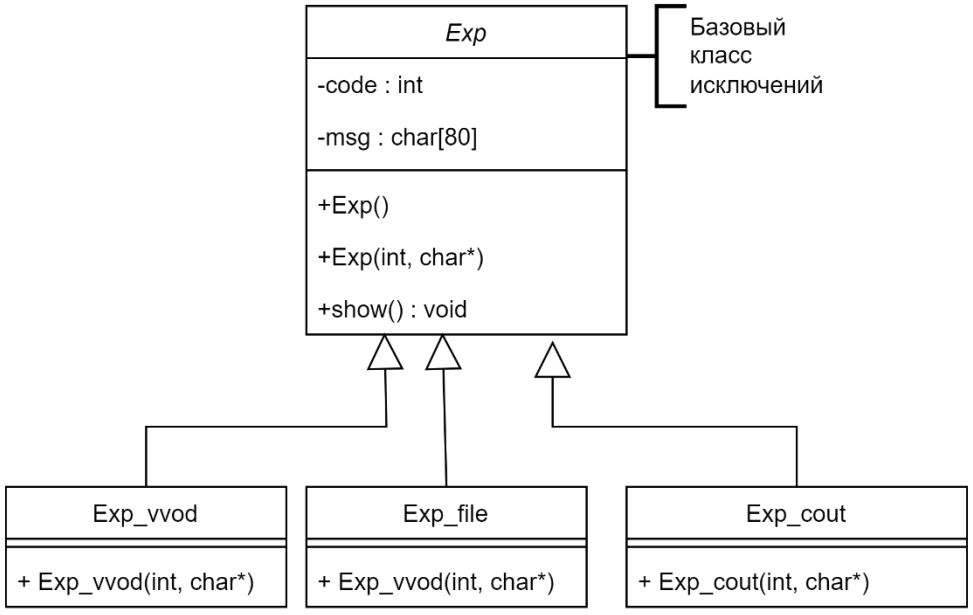
ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить краткие теоретические сведения.
2. Ознакомиться с материалами литературных источников.
3. Ответить на контрольные вопросы.
4. Разработать алгоритм программы.
5. Написать, отладить и выполнить программу.

ЗАДАНИЕ

Написать программу на основе лабораторных №3-4:

Реализовать базовый класс ошибок, от него унаследовать несколько других классов (для проверки ввода числа, ввода строк и т.д.)



;

Написать функции для проверки ввода различных полей (имя, дата, номер карты…) и добавить их вызов при заполнении полей класса. В них генерировать исключения, например при неправильном вводе

```
if(str[i]<'A'||str[i]>'Z')
throw ExpVvod(123, "Вводите только прописными латинскими буквами.");
```

При необходимости добавить генерацию ошибок на неправильное выделение памяти, выход за границы контейнера и т.п.

Продемонстрировать работу с исключениями.

ЛИТЕРАТУРА

1. Бьерн Страуструп. Язык программирования C++. Пер. с англ.- М.: «Издательство БИНОМ», 2004. – 1098 с.

2. Скляров В.А. Язык C++ и объектно-ориентированное программирование. Мн.: Выш.шк., 1997г. – 478 с.: ил.

3. Дейтел Х., Дейтел П. Как программировать на C++. Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 2001. – 1152 с.: ил.

4. Лафоре Р. Объектно-ориентированное программирование в C++.– «Питер», 2003. – 923 с.

5. Луцик Ю.А., Ковальчук А.М., Лукьянова И.В. Объектно-ориентированное программирование на языке C++. Мн.: БГУИР, 2003. – 203 с.: ил.

6. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – «Питер», Addison-Wesley, 2009. – 366 с.

7. Фримен Э., Фримен Э., Сьерра К., Бейтс Б. Паттерны проектирования. – «Питер», 2011 г. – 656 с.