Обработка исключительных ситуаций

Исключения

- Это ошибки, возникающие во время работы программы
- Они могут быть вызваны различными обстоятельствами:
  - выход за пределы памяти
  - ошибка открытия файла
  - попытка инициализировать объект недопустимым значением
  - использование индекса, выходящего за пределы вектора

Для чего нужны исключения?

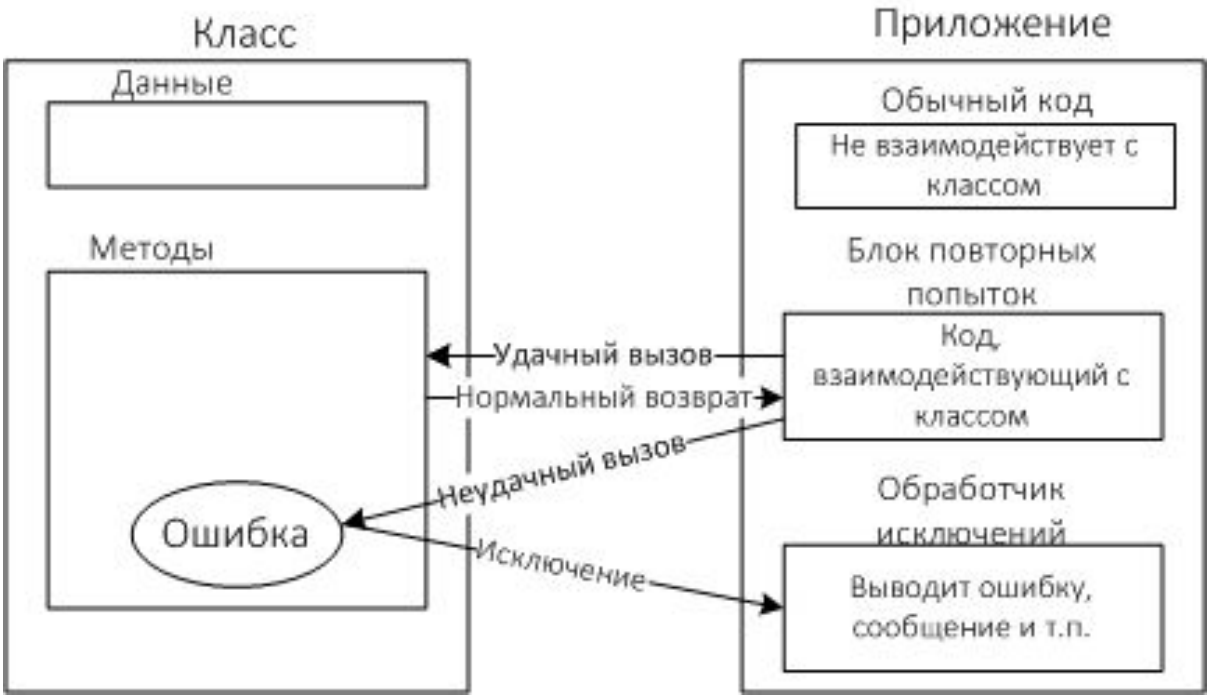
- Как обрабатываются ошибки в программе на языке С?
- Программы на С сообщают об ошибке возращением установленного значения из функции, в которой она произошла. Каждый раз при вызове этих функций происходит проверка возвращаемых значений (например, открытие файла)

```
if( somefunc() == ERROR_RETURN_VALUE )
  //обработка ошибки или вызов обработчика ошибок
else
  //нормальная работа
if( anotherfunc() == NULL )
  //обработка ошибки или вызов обработчика ошибок
else
  //нормальная работа
if( thirdfunc() == 0 )
  //обработка ошибки или вызов обработчика ошибок
else
  //нормальная работа
```

Проблемы:

- Каждый вызов функции должен проверяться программой -> добавление условий, выражений обработки ошибки -> рост кода
- При использовании классов ошибки могут возникать и при неявном вызове функции, т.е. при работе конструкторов
- Использование библиотеки классов

Механизм исключений



- Пусть приложение создает объекты некоторого кода и работает с ними
- Если метод обнаруживает ошибку, то он информирует приложение об этом или *генерирует исключительную ситуацию*
- Отдельная секция кода приложения содержит операции по обработке ошибок – *обработчик исключительных ситуаций,* который отлавливает исключения
- Код приложения, использующий объекты класса, заключается в *блок повторных попыток*

Синтаксис исключений

Механизм обработки исключительных ситуаций основан на трех ключевых словах: **try, catch, throw**

```
try {
// код, подлежащий контролю
// функции могут генерировать исключительные ситуации
// может содержать несколько оператор или целую программу
}
catch (тип1 аргумент) {
// перехват исключительных ситуаций
}
catch (тип2 аргумент) {
//
}
…
catch (типN аргумент) {
//
}
```

С одним блоком try может быть связано несколько операторов catch. Из нескольких операторов выбирается оператор catch, тип аргумента которого совпадает с типом исключительной ситуации. Аргумент может быть объектом встроенного типа или класса

Синтаксис исключений

throw исключительная ситуация;

Оператор throw генерирует указанную исключительную ситуацию. Если в программе есть ее перехват, оператор throw должен выполняться либо внутри блока try, либо внутри функции, явно или неявно вызываемой внутри блока try

Если генерируется исключительная ситуация, для которой не предусмотрена обработка, программа может прекратить свое выполнение. В этом случае вызывается стандартная функция terminate(), которая по умолчанию вызывает функцию abort()

обработки исключительной ситуации

```
int main()
{
cout << “начало \n”;
try {
   cout << “внутри блока try \n”;
   throw 100;          // генерируем ошибку
   cout << “этот оператор не выполняется”;
} catch (int i) {      // перехват ошибки

 cout << “перехват исключительной ситуации – значение равно: ”;
 cout << i << “\n”;
}
cout << “конец” ;
return 0;
}
```

При генерации исключительной ситуации управление передается оператору catch, а выполнение блока try прекращается. При этом блок catch не вызывается, а просто программа переходит к его выполнению.

Обычно оператор catch пытается исправить ошибку. Если это возможно, то выполнение программы возобновляется с оператора, следующего за блоком catch. Однако часто ошибку исправить невозможно, и блок catch прекращает выполнение программы, вызывая exit() или abort()

- Исключение может генерироваться вне блока try только в том случае, если оно генерируется функцией, которая содержит оператор catch и вызывается внутри блока try
- Блок try может находиться внутри функции. В этом случае при каждом входе в функцию обработка исключительной ситуации выполняется заново.
- Код блока catch выполняется только при перехвате исключительной ситуации
- Исключительная ситуация может иметь любой тип, в том числе быть объектом класса, определенного пользователем
- Если исключительные ситуации описываются с помощью базового и производных классов, оператор catch, соответствующий базовому классу, одновременно соответствует всем производным классам
- Если необходимо обрабатывать не отдельные типы исключительных ситуаций, а перехватывать все подряд, то применяется следующий вид оператора catch:

```
catch (…)
{
// обработка всех исключительных ситуаций
}
```

Скелет класса, в котором могут возникнуть ошибки

```
// эта программа не работает!
class AClass              // просто класс
  {
  public:
  class AnError           // класс exception
    {
    };
  void Func()             //какой-то метод
    {
    if( /* условие ошибки */ )
      throw AnError();    // генерировать исключение
    }
  };
int main()                // приложение как бы
  {
  try                     // блок повторных попыток
    {
    AClass obj1;          // взаимодействие с объектами AClass
    obj1.Func();          // тут может возникнуть ошибка
    }
  catch(AClass::AnError)  //обработчик ошибок
    {                     // (улавливающий блок)
    }
  return 0;
  }
```

Демонстрация механизма исключений

```
#include <iostream>
using namespace std;
const int MAX = 3;   //в стеке максимум 3 целых числа
class Stack
 {
 private:
   int st[MAX];  //стек: целочисл. массив
   int top;         //индекс вершины стека
 public:
   class Range   //класс исключений для Stack
     {                //внимание: тело класса пусто
     };
----------------------------------------------------
   Stack()              //конструктор
     { top = -1; }
----------------------------------------------------
   void push(int var)
     {
     if(top >= MAX-1)    //если стек заполнен,
       throw Range();    //генерировать исключение
     st[++top] = var;    //внести число в стек
     }
----------------------------------------------------
   int pop()
     {
     if(top < 0)         //если стек пуст,
       throw Range();    //исключение
     return st[top--];   //взять число из стека
     }
 };
int main()
 {
 Stack s1;

 try
```

Демонстрация обработчика двух исключений

```
#include <iostream>
using namespace std;
const int MAX = 3; //в стеке может быть до трех целых чисел
/
class Stack
 {
 private:
   int st[MAX];         //стек: массив целых чисел
   int top;             //индекс верхушки стека
 public:
   class Full { };      //класс исключения
   class Empty { };     //класс исключения
//--------------------------------------------------------
   Stack()              //конструктор
    { top = -1; }
//--------------------------------------------------------
   void push(int var)   //занести число в стек
    {
    if(top >= MAX-1)    //если стек полон,
      throw Full();     //генерировать исключение Full
    st[++top] = var;
    }
//--------------------------------------------------------
   int pop()            //взять число из стека
    {
    if(top < 0)         //если стек пуст,
      throw Empty();    //генерировать исключение Empty
    return st[top--];
    }
 };
////////////////////////////////////////////////////////////
int main()
 {
 Stack s1;

 try
```

Исключения и класс Distance

```
#include <iostream>
using namespace std;
class Distance {
  private:
    int feet;
    float inches;
  public:
    class InchesEx { };   //класс исключений
-----------------------------------------------
    Distance()      //конструктор (без аргументов)
     { feet = 0; inches = 0.0; }
-----------------------------------------------
    Distance(int ft, float in) {
     if(in >= 12.0)  //если дюймы указаны неверно,
       throw InchesEx();      //генерировать исключение
     feet = ft;
     inches = in;
     }
-----------------------------------------------
    void getdist()       //получить длину от пользователя
     {
     cout << "\nEnter feet: ";  cin >> feet;
     cout << "Enter inches: ";  cin >> inches;
     if(inches >= 12.0)  //если дюймы неправильные,
       throw InchesEx(); //генерировать исключение
     }
-----------------------------------------------
    void showdist()
{ cout << feet << "\'-" << inches << '\"'; }
  };
int main()
  {
  try
    {
```

Исключения с аргументами

```
#include <iostream>
#include <string>
using namespace std;
class Distance {
  private:
    int feet;
    float inches;
  public:
-------------------------------------------------------
    class InchesEx        //класс исключений
      {
      public:
        string origin;   //для имени функции
        float iValue;    //для хранения ошибочного значения
        InchesEx(string or, float in) {
          origin = or;     //сохраненная строка с именем виновника ошибки
          iValue = in;     //сохраненное неправильно значение дюймов
          }
      };
-------------------------------------------------------
    Distance()
{ feet = 0; inches = 0.0; }
-------------------------------------------------------
    Distance(int ft, float in)
      {
      if(in >= 12.0)
        throw InchesEx("Конструктор с двумя аргументами", in);
      feet = ft;
      inches = in;
      }
-------------------------------------------------------
    void getdist()
{
      cout << "\nВведите футы: ";  cin >> feet;
      cout << "Введите дюймы: ";  cin >> inches;
      if(inches >= 12.0)
        throw InchesEx("функция getdist()", inches);
      }
```

Демонстрация исключения bad_alloc

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

int main()
  {
  const unsigned long SIZE = 10000; //объем памяти
  char* ptr;              //указатель на адрес в памяти


  try
    {
    ptr = new char[SIZE]; //разместить в памяти SIZE
байт
    }
  catch(bad_alloc)       //обработчик исключений
    {
    cout << "\n Исключение bad_alloc:
невозможно разместить данные в памяти.\n";
    return(1);
    }
  delete[] ptr;          //освободить память
  cout << "\nПамять используется без сбоев.
\n";
  return 0;
  }
```