Исключения

Как **не** допустить логических ошибок при выполнении программы?

Некорректное создание объектов классов(1)

```
class Table{
    int *l,*w,*h;
    char* color;
    bool Init(int l1,int w1, int h1,char* col);
  public:
    Table(int len, int wid, int hei ,char* col, bool& err);
    void Print();
    ~Table();
};
```

Некорректное создание объектов классов(2)

```
bool Table::Init(int l1,int w1, int h1,char* col)
{    cout<<"bool Table::Init(int l1,int w1, int h1,char* col)\n";
   if( (l1>0) && (h1>0) && (w1>0) )
       {
       *l=l1;*h=h1;*w=w1;
       strcpy(color,col);
       return true;
       }
   else{
       cout<<"\tError, incorreect data init\n";
       return false;    }    }
```

Некорректное создание объектов классов(3)

```
Table::Table (int len,int wid, int hei,char* col,bool& err)
{
cout<<"Table::Table(int len,int wid, int hei,char col,bool& err)\n";
    h=new int;  l=new int;  w=new int;
    color=new char[20];
    err=Init(len, wid, hei, col);
    if(!err)
        {
         delete this;
        }
    else cout<<"Create Table\n";
}
```

Некорректное создание объектов классов(4)

```
void Table::Print()
{   cout<<"void Table::Print()\n";
    cout<<"\tTable information:\n";
    cout<<"\t\t h = "<<*h<<"\n";
    cout<<"\t\t w = "<<*w<<"\n";
    cout<<"\t\t l = "<<*l<<"\n";
    cout<<"\t\t color: "<<color<<"\n\n";
}
Table::~Table()
{
    cout<<"Table::~Table()\n";
    delete h; delete l; delete w; delete[]color;
}
```

Некорректное создание объектов классов(5)

```
int main()
{
    bool err=false;
    Table *T= new Table(-10,10,10,"black",err);

    if(!err) {  T=NULL;   return -1; }
    T->Print();
    delete T;
    return 0;
}
```

Механизм исключений

- **throw**

  для обозначения кода ошибки

- **try**

  для начала блока, в котором может возникнуть «аварийная» ситуация

- **catch**

  для обработки кода ошибки, выкидываемой throw.

Что такое исключение

- **Исключение** – это объект, а не ситуация.
- С исключением можно работать как с переменной.
- Тип объекта исключения может быть любым.

Определение исключения

Синтаксис:

throw выражение_генерации_исключения;

Примеры:

- **throw “Ошибка: деление на ноль”;**
- **throw 1;**
- **throw s[i];**

Объект - исключение

- **throw MyException(1,"Error 1");**

  // оператор-ловушка - catch(MyException m)

- **throw new MyException(1,"Error 1");**

  // оператор-ловушка - catch(MyException *m)

Перехват исключения

Синтаксис:

```
try {
    /* контролируемый блок */
      }
catch (спецификация исключения № 1)
    {  /* блок обработки  1*/  }
catch (спецификация исключения № 2)
    {  /* блок обработки 2*/  }
…
catch (спецификация исключения № n)
    {  /* блок обработки  n*/  }
```

Спецификация исключения

- **(тип имя)**

  Если необходимо объект – исключение использовать в блоке обработки

- **(тип)**

  Если в блоке обработки объект-исключение не используется

- **(...)**

  Такой обработчик перехватывает все исключительные ситуации.

Правила работы с **try**

- Если сработал хоть один блок обработки исключений, то последующие не выполнятся.

Правила работы с **try**

- Если сработал хоть один блок обработки исключений, то последующие не выполнятся.
- Последним обработчиком должен быть самый общий вариант, для обработки всех исключений.

Правила работы с **try**

- Если сработал хоть один блок обработки исключений, то последующие не выполнятся.
- Последним обработчиком должен быть самый общий вариант, для обработки всех исключений.
- Если в блоке **try** не выпало исключительной ситуации, то все блоки **catch** пропускаются и выполняются те действия, которые дальше идут по коду.

Правила работы с **try**

- Если сработал хоть один блок обработки исключений, то последующие не выполнятся.
- Последним обработчиком должен быть самый общий вариант, для обработки всех исключений.
- Если в блоке **try** не выпало исключительной ситуации, то все блоки **catch** пропускаются и выполняются те действия, которые дальше идут по коду.
- Блоки **try** и **catch** могут быть вложенными.

Пример: класс MyEx (1)

```
class MyEx{
  public:
    int code;
    char* message;
     MyEx(int a, char* b);
    ~MyEx();
};
```

Пример : класс MyEx (2)

```
MyEx::MyEx (int a, char* b)
{
    cout<<"MyEx::MyEx(int a,char* b)\n";
    code=a;
    message=new char [ strlen(b) + 1 ];
    strcpy ( message, b );
}
```

Пример : класс MyEx (3)

```
MyEx::~MyEx()
{
    cout<<"MyEx::~MyEx()\n";
    delete[] message;
}
```

Пример : генерация исключений (1)

```
void Table::Init (int len ,int wid, int hei, char* col)
{
    *l=len; *w=wid; *h=hei; strcpy(color,col);
}
```

Пример : генерация исключений (2)

```
Table::Table (int len, int wid, int hei, char *col)
{
    if(len>0&&wid>0&&hei>0)
        {  h=new int; l=new int; w=new int;
           color=new char[20];
  Init(len,wid,hei,col);
        }
    else {
        throw(MyEx(1,"some parameters are not
            positive"));
        }   }
```

Пример : генерация исключений (3)

```
int main()
{
    try
     {
          Table* T=new Table(-1,5,6,“green”);
          T->Print();
          delete T;
           }
    catch(MyEx e)
     {
           cout<<e.code<<" - error!"<<e.message<<endl;
    }
```

Пример : генерация исключений (3)

```
catch(...)
    {
        cout<<"I catch everything"<<endl;
    }
    return 0;
}
```

Зачем нужны исключения

- Исключение вынуждает вызывающий код признать состояние ошибки и обработать его.
- Исключение обеспечивает четкое разделение между кодом, который обнаруживает ошибку, и кодом, который обрабатывает ошибку.