# 异常处理机制

苏州大学计算机科学与技术学院
面向对象与C++程序设计课程组

# 异常处理的必要

- 程序的正常结束
- 程序的非正常结束
  - 除数为0
  - 指针越界
- 为了提供安全和稳定的程序
- 保证程序在环境条件出现异外或用户操作不当的时候程序也有正确合理的表现，避免出现灾难性后果

# C错误处理方式

- C的处理方法
  - 返回值是C常用判断方法
  - **if ((p = malloc(n)) == NULL)**
    **/* ... */**

# C++异常处理

- 异常处理的步骤：
  - 发生异常则暂停正常程序
  - 搜寻处理此异常的代码
  - 执行异常处理代码
- 将异常处理与正常代码分离，提高程序的可读性、可维护性
- 当在函数体中检测到异常条件存在，但无法确定相应的处理方法时，将引发一个异常，并由函数的直接或间接调用者处理这个异常

# 异常的实现

- throw
  - 抛出异常
  - 在被调用函数中
- try
  - 放入监视异常的语句检测是否触发异常
  - 如果有异常就用throw抛出
- catch
  - 捕获匹配的异常
  - 在上层调用函数中
- try与catch语句总是结合使用

# 异常处理器

- 每个异常处理器，都包含：
  - 一个try
  - 一个或多个throw
  - 一个或多个catch

# 一个简单的异常处理器

```
int main(int argc,char *argv[])
{
cout<<”开始”<<endl;
try{
    cout<<”进入try语句块.”<<endl;
    throw 200;
    cout<<” 不会被执行”<<endl;
}
catch(int i){
    cout<<”捕获一个异常，它的值为：”<<i<<endl;
}
    cout<<”结束”<<endl;
    return 0;
}
```

# throw和catch可以不在同一函数中

```
int Div(int x,int y)
{
  if(y==0)
      throw y;
  return x/y;
}
```

```
int main(int argc ,char *argv[])
{
    try
    {
        cout<<"5/2="<<Div(5,2)<<endl;
        cout<<"8/0="<<Div(8,0)<<endl;
        cout<<"7/3="<<Div(7,3)<<endl;
    }
    catch(int i)
    {
        if(i==0)
            cout<<"Exception of dividing zero.\n";
    }
    cout<<"end";
    return 0;
}
```

# 对异常使用...

- catch(...)
- 可以捕获一个try抛出的所有异常
- 不能区分异常的类型
- 通常与正常catch结合使用于捕获不知类型的异常

# 异常接口声明

- 异常声明
  - 函数返回值类型　函数名（含数形参列表）throw(异常类型列表);
- 例如：

  void fun() throw(int,string,float);

  抛出的异常对象的类型有int 、string、float。

# 异常接口声明（续）

- 当为throw()的形式时，此函数不抛出任何类型的异常对象。
  - void fun() throw();
- 函数后面没有throw(异常类型列表)子句时
  - 抛出任何类型的异常对象
  - 即抛出的异常对象的类型不定
  - void fun();

# 异常处理的不唤醒机制

- 一旦在保护段执行期间发生异常，则立即抛掷，由相应的catch子句捕获处理
- 抛掷——捕获间的代码被越过而不执行
- 程序从try块后跟随的最后一个catch子句后面的语句继续执行下去。
- 异常处理将检测与处理分离
  - 以便各司其职、灵活搭配地工作
  - 它们的联系靠类型

# 异常的捕获

- 异常抛掷后总是沿着函数调用链往上，直到被某个函数捉住，因此，异常抛掷、捕捉以及处理都依附于函数，函数承载着异常。
- 不处理的异常
  - 被系统默认的“强制捕捉器” terminate捕获
  - Terminate是系统资源，默认操作是系统的abort函数，从而无条件地终止程序的执行

# 异常的总结

- 异常是一种程序控制机制，与函数机制独立和互补
- 函数是一种以栈结构展开的上下函数衔接的程序控制系统,
- 异常是另一种控制结构, 它依附于栈结构, 却可以同时设置多个异常类型作为网捕条件, 从而以类型匹配在栈机制中跳跃回馈.

# 异常的高级使用

- 异常也已经成为了一种有效的控制手段。
- 异常并非一定是针对错误时刻处理
  - 一个例子：
  - 在递归中找到数据之后快速返回
    - 函数调用返回效率低
    - 有时走异常则非常巧妙