第8讲 C++工具

8.1 异常处理

8.1.1 异常处理的任务

程序中常见的错误有两大类: 语法错误和运行错误。在编译时，编译系统能发现程序中的语法错误。

有的程序虽然能通过编译，也能投入运行。但是在运行过程中会出现异常，得不到正确的运行结果，甚至导致程序不正常终止，或出现死机现象。

所谓异常处理指的是对运行时出现的差错以及其他例外情况的处理。

8.1.2 异常处理的方法

C++采取的办法是: 如果在执行一个函数过程中出现异常，可以不在本函数中立即处理，而是发出一个信息，传给它的上一级(即调用它的函数)，它的上级捕捉到这个信息后进行处理。

如果上一级的函数也不能处理，就再传给其上一级，由其上一级处理。如此逐级上送，如果到最高一级还无法处理，最后只好异常终止程序的执行。

这样做的好处是使底层的函数专门用于解决实际任务，而不必再承担处理异常的任务，以减轻底层函数的负担，而把处理异常的任务上移到某一层去处理。这样可以提高效率。

C++处理异常的机制是由3个部分组成的，即检查(try)、抛出(throw)和捕捉(catch)。

需要检查的语句放在try块中

throw发出一个异常信息

catch捕捉异常信息

例8.1 给出三角形的三边a,b,c，求三角形的面积。只有a+b>c,b+c>a,c+a>b时才能构成三角形。设置异常处理，对不符合三角形条件的输出警告信息，不予计算。

先写出没有异常处理时的程序:

```
#include <iostream>
#include <cmath>
using namespace std;
int main( )
{double triangle(double,double,double);
 double a,b,c;
 cin>>a>>b>>c;
 while(a>0 && b>0 && c>0)
  {cout<<triangle(a,b,c)<<endl;
   cin>>a>>b>>c;
  }
  return 0;
 }
```

```
double triangle(double a,double b,double c)
{double area;
 double s=(a+b+c)/2;
 area=sqrt(s*(s-a)*(s-b)*(s-c));
 return area;
}
```

运行情况如下:

6 5 4↙ (输入a,b,c的值)

9.92157 (输出三角形的面积)

1 1.5 2↙ (输入a,b,c的值)

0.726184 (输出三角形的面积)

1 2 1↙ (输入a,b,c的值)

0 (输出三角形的面积，此结果显然不对,因为不是三角形)

1 0 6↙ (输入a,b,c的值)

(结束)

修改程序：在主函数中的try-catch块中调用traingle函数，检测有无异常信息，并作相应处理。

```
#include <iostream>
#include <cmath>
using namespace std;
void main( )
{double triangle(double,double,double);
 double a,b,c;
 cin>>a>>b>>c;
 try//在try块中包含要检查的函数
  {while(a>0 && b>0 && c>0)
    {cout<<triangle(a,b,c) <<endl;
   cin>>a>>b>>c;
    }
  }
catch(double) //用catch捕捉异常信息并作相应处理
  {cout<<a<<",,"<<b<<",,"<<c<<",,not a triangle!"<<endl;}
 cout<<"end"<<endl;
}
```

需要检查的语句放在try后面的花括号中

异常信息的类型名

系统根据throw抛出的异常信息寻找与之匹配的catch子句

异常处理后，继续执行catch后面的语句

```
double triangle( double a,double b,double c)  //计算三角形的面积的函数
{double s=(a+b+c)/2;
 if (a+b<=c||b+c<=a||c+a<=b) throw a;   //当不符合三角形条件抛出异常
 return sqrt(s*(s-a)*(s-b)*(s-c));
}
```

throw抛出异常信息后，流程立即转到上一级的函数(main 函数)

程序运行结果如下:

```
6 5 4↙ (输入a,b,c的值)
9.92157          (计算出三角形的面积)
1 1.5 2↙         (输入a,b,c的值)
0.726184         (计算出三角形的面积)
1 2 1↙           (输入a,b,c的值)
a=1,b=2,c=1, that is not a triangle!  (异常处理)
end
```

说明:

(1) 被检测的函数必须放在try块中，否则不起作用。

(2) catch块是try-catch结构中的一部分，必须紧跟在try块之后，在二者之间也不能插入其他语句。

(3) try和catch块，即使花括号内只有一个语句，也不能省略花括号。

(4) 一个try-catch结构中只能有一个try块，但却可以有多个catch块，以便与不同的异常信息匹配。

(5) catch后面的圆括号中，一般只写异常信息的类型名， 如：catch(double)

8.1.3 在函数声明中进行异常情况指定

为便于阅读程序，C++允许在声明函数时列出可能抛出的异常类型，如例8.1中函数声明改写为

```
double triangle(double,double,double) throw(double);
```

表示triangle函数只能抛出double类型的异常信息。

如果写成

```
double triangle(double,double,double)
throw(int,double,float,char);
```

则表示triangle函数可以抛出int,double,float或char类型的异常信息。

如果在声明函数时未列出可能抛出的异常类型，则该函数可以抛出任何类型的异常信息。

异常指定必须同时出现在函数声明和函数定义的首行。

8.1.4 在异常处理中处理析构函数

C++的异常处理机制会在throw抛出异常信息被catch捕获时，对有关的局部对象进行析构，然后执行与异常信息匹配的catch块中的语句。

例8.3 在异常处理中处理析构函数。

```
#include <iostream>
#include <string>
using namespace std;
class Student
{public:
   Student(int n,string nam)//定义构造函数
{cout<<"constructor-"<<n<<endl;
num=n;name=nam;}
~Student( ){cout<<"destructor-"<<num<<endl;}//定义析构函数
 void get_data( );                          //成员函数声明
private:
int num;
string name;
 };
```

```
void Student::get_data( )                    //定义成员函数
{if(num==0) throw num;                       //如num=0,抛出int型变量num
  else cout<<num<<" "<<name<<endl; //若num≠0，输出num,name
  cout<<"in get_data()"<<endl; //输出信息，表示目前在get_data函数中
 }
void fun( )
{Student stud1(1101,"Tan");          //建立对象stud1
stud1.get_data( );                   //调用stud1的get_data函数
Student stud2(0,"Li");               //建立对象stud2
stud2.get_data( );                   //调用stud2的get_data函数
}
int main( )
{cout<<"main begin"<<endl;           //表示主函数开始了
cout<<"call fun( )"<<endl;           //表示调用fun函数
try
  {fun( );}                          //调用fun函数
```

```
catch (int n)
  {cout<<"num="<<n<<",error!"<<endl;}      //表示num=0出错
cout<<"main end"<<endl;                    //表示主函数结束
return 0;
}
```

程序运行结果如下:

```
main begin
call fun( )
constructor-1101
1101 tan
in get_data()
constructor-0
destructor-0
destructor-1101
num=0,error!
main end
```

8.2 使用早期的函数库

C++程序中可以使用C语言的函数库。如果要用函数库中的函数，就必须在程序文件中包含有关的头文件。在C++中使用这些头文件有两种方法。

(1) 用C语言的传统方法。头文件名包括后缀.h，如stdio.h，math.h等。如

```
#include <math.h>
```

(2) C++标准要求系统提供的头文件不包括后缀.h，例如iostream、string。在程序中要对命名空间std作声明。如

```
#include <cstdio>
#include <cmath>
using namespace std;
```

目前所用的大多数C++编译系统既保留了C的用法，又提供了C++的新方法。下面两种用法等价，可以任选：

C传统方法

```
#include <stdio.h>
#include <math.h>
#include <string.h>
```

C++新方法

```
#include <cstdio>
#include <cmath>
#include <cstring>
using namespace std;
```

可以使用传统的C方法，但提倡使用C++的新方法。