Exception handling in C++

The process of exception handling allows us to treat problems that can occur in a program in a more organized way. The advantage of exception handling is a higher automation code.

Exception handling is made by *try* ... *catch* blocks and can be thrown by *throw*.

try – detects the part of the code where we are looking for exceptions,
catch – is used to determine the type of exceptions and how the caught exceptions will be handled
throw – it can be called from anywhere in the program code in a *try* ... *catch* block or function.

If an error is thrown and not caught in a try ... catch block it will cause an abrupt termination of the program.

Exercises: [Jamsa & Klander, Stroustrup]

1. Writing a simple try-catch

```
int  main(void)
 {
   cout << "Start" << endl;
   try {
      cout << "Inside try block." << endl;
      throw 100;
      cout << "This will not execute.";
    }
   catch(int i) {
      cout << "Caught an exception -- value is: ";
      cout << i << endl;
    }
   cout << "End";
   return 0;
 }
```

Instead of waiting for the program to generate an error, we use *throw* in order to produce the error.

After the block *try* throws the error, the block *catch* catches it and processes the value passed by *throw*.

2. **Exceptions are specific to each type**

```
int  main(void)
 {
   cout << "Start" << endl;
   try {
      cout << "Inside try block." << endl;
      throw 100;
      cout << "This will not execute.";
    }
   catch(double d) {
      cout << "Caught a double exception -- value is: ";
      cout << d << endl;
    }
   cout << "End";
   return 0;
 }
```

The exception which is handled in the block *try* has to have the same type as the type specified by *catch*.

The program above catches a *double* exception and try throws an *int* exception. ➔ Abrupt program termination

3. **Throwing exceptions by using a function inside the body of the block try**

```
void XHandler(int test)
 {
    cout << "Inside XHandler, test is:" << test << endl;
    if(test) throw test;
 }

int main(void)
 {
   cout << "Start: " << endl;
   try {
      cout << "Inside try block." << endl;
      XHandler(1);
      XHandler(2);
      XHandler(0);
    }
   catch(int i) {
```

```
            cout << "Caught an exception. Value is: ";
            cout << i << endl;
         }
        cout << "End ";
        return 0;
         }
```

4. The bloc try inside the body of a function

```
void XHandler(int test)
 {
   try {
      if(test) throw test;
    }
   catch(int i)
    {
      cout << "Caught exception #: " << i << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(1);
   XHandler(2);
   XHandler(0);
   XHandler(3);
   cout << "End";
   return 0;
 }
```

When we have a block try inside the body of a function, the C++ language reinitializes the block each time we access that function.

The program throws only 3 exceptions although we call the function 4 times because the call XHandler(0) is evaluated to false.

5. When the instruction *catch* is executed?

```
int main(void)
 {
   cout << "Start" << endl;
   try
   {
      cout << "Inside try block." << endl;
      cout << "Still inside try block." << endl;
   }
   catch(int i)
   {
      cout << "Caught an exception--value is: " << endl;
      cout << i << endl;
   }
   cout << "End";
   return 0;
   }
```

The instructions from the block *catch* are executed only if the program throws an exception in the block *try*.

6. Several *catch* blocks and one single block *try*

```
void XHandler(int test)
 {
   try
   {
      if(test==0) throw test;
      if(test==1) throw "String";
      if(test==2) throw 123.23;
    }
   catch(int i)
    {
      cout << "Caught exception #: " << i << endl;
    }
   catch(char *str)
    {
      cout << "Caught string exception: " << str << endl;
    }
   catch(double d)
    {
      cout << "Caught exception #: " << d << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(0);
   XHandler(1);
   XHandler(2);
```

```
    cout << "End";
  return 0;
 }
```

7. (…) with exceptions

Syntax:

```
 try
 {
    // instructions
  }
 catch(...)
  {
    //exception handling
  }
```

Example:

```
void XHandler(int test)
 {
   try
   {
      if(test==0) throw test;
      if(test==1) throw 'a';
      if(test==2) throw 123.23;
    }
   catch(...)
    {
      cout << "Caught one." << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(0);
   XHandler(1);
   XHandler(2);
   cout << "End";
  return 0;
 }
```

8. Handling explicit exceptions and generic exceptions

```
void XHandler(int test)
 {
   try
   {
      if(test==0) throw test;
      if(test==1) throw 'a';
      if(test==2) throw 123.23;
    }
   catch(int i)
    {
      cout << "Caught an integer." << endl;
    }
   catch(...)
    {
      cout << "Caught one." << endl;
    }
 }

int main(void)
 {
   cout << "Start: " << endl;
   XHandler(0);
   XHandler(1);
   XHandler(2);
   cout << "End";
  return 0;
 }
```

9. Exceptions restrictions

```
returned_type function-name (list-of-parameters) throw (types-
list)
      {
        //the body of the function
      }
```

To restrict the exceptions that are thrown by our functions we add a throw clause in the definition of the function.

When calling a function with the clause *throw* this function can throw only that types which are found in the types-list. If the function throws any other type of exception this will produce an abrupt termination of the program.

If we do not want to throw any exception, then we use in the function the types-list as the empty list.

```
void XHandler(int test) throw()
 {
   if(test==0)
     throw test;
   if(test==1)
     throw 'a';
   if(test==2)
     throw 123.23;
 }

int main(void)
 {
   cout << "Start: " << endl;
   try
    {
      XHandler(0);     // try passing 1 and 2 for different
                       // responses
     }
   catch(int i)
    {
      cout << "Caught an integer." << endl;
    }
   catch(char c)
    {
      cout << "Caught a character." << endl;
    }
   catch(double d)
    {
      cout << "Caught a double." << endl;
    }
   cout << "End ";
   return 0;
 }
```

Another example:

```
void XHandler(int test) throw(int, char, double)
 {
   if(test==0) throw test;
   if(test==1) throw 'a';
   if(test==2) throw 123.23;
 }

int main(void)
 {
   cout << "Start: " << endl;
   try {
      XHandler(0);                      // try passing 1 and 2
                                      // for different responses
    }
   catch(int i) {
      cout << "Caught an integer." << endl;
    }
   catch(char c) {
      cout << "Caught a character." << endl;
    }
   catch(double d) {
      cout << "Caught a double." << endl;
    }
   cout << "End ";
    return 0;
 }
```

What is the difference between these two examples?

10.Re-throw an exception

```
void XHandler(void)
 {
   try {
      throw "hello";
    }
   catch(char *) {
      cout << "Caught char * inside XHandler." << endl;
      throw;
    }
 }


int main(void)
 {
   cout << "Start: " << endl;
   try {
      XHandler();
    }
   catch(char *)
```

```
     {
        cout << "Caught char * inside main." << endl;
      }
     cout << "End ";
     return 0;
   }
```

Problems:

P1. Division by zero

```
void divide(double a,double b)
{
     try
     {
          if (!b) throw b; //check division by zero
          cout<<"the result is  "<<a/b<<endl;
     }
     catch(double b)
     {
          cout<<"Division by zero is impossible"<<endl;
     }
}
int main()
{
     double i,j;
     do
     {
          cout<<"introduce the numerator"<<endl;
          cin>>i;
          cout<<"introduce the denominator (0 for
stop)"<<endl;
          cin>>j;
          divide(i,j);
     }
     while (i!=0);
     return 0;
}
```

P2. Study the following problem

```
#define MAXX 80
#define MAXY 25
class Point
{
public:
      class xZero {};
      class xOutOfScreenBounds {};
      Point(unsigned __x, unsigned __y)
      {
      x = __x;
      y = __y;
      }
      unsigned GetX()
      {
      return x;
      }
      unsigned GetY()
      {
      return y;
      }

void SetX(unsigned __x)
{
if(__x > 0)
      if(__x <= MAXX)
            x = __x;
      else
      throw xOutOfScreenBounds();
else
throw xZero();
}

void SetY(unsigned __y)
{
if(__y > 0)
      if(__y <= MAXY)
            y = __y;
      else
      throw xOutOfScreenBounds();
else
      throw xZero();
}
protected:
	int x, y;
};

int main()
{
Point p(1, 1);
try {
p.SetX(5); // CORRECT!
// p.SetX(0); // throws an xZero exception
cout << "p.x successfully set to " << p.GetX() << "."<<endl;
// throws an xOutOfScreenBounds exception
p.SetX(100);
}

catch(Point::xZero)
{
cout << "Zero value!\n";
}

catch(Point::xOutOfScreenBounds)
{
cout << "Out of screen bounds!\n";
}

catch(...)
{
cout << "Unknown exception!\n";
}
return 0;
}
```

Because an exception is an instance of a class, the derivation can create hierarchies of exception handling.

Attention! There exists the possibility of an exception to occur even in the code where we handle an exception! Such situations must be avoided!