Lecture 9

Exception Handling

Topics

- Exceptions (Runtime errors)
- Try-catch statement
- Throwing an exception
- Exceptions and class constructors

Exceptions (Runtime errors)

- An exception (exceptional event) is a run-time error.
- Exceptions may cause program termination (program crash) during program execution.
- Exception Handling is the process of run-time error management.

- Examples of runtime errors:
  - Division by zero
  - Insufficient memory
  - Invalid index of an array
  - Null pointer
  - File not found
  - Arithmetic overflow

- In most cases, IF statements should be used to prevent runtime errors, before the occurance of exceptions.

C++ Exceptions

- C++ exception mechanism provides an object-oriented approach to handle **runtime errors** generated by C++ classes.

- For example, a **constructor** in a user-written String class might generate an exception, if the application tries to initialize an object with a string that's too long.

- Similarly, a program can check whether a file was opened or written successfully, and generate an exception if it was not.

Error Handling with IF statements

In C language programs, an error is often signaled by returning a particular value from the function in which it occurred.

```
int main()
{
   if  ( func1() == ERROR_RETURN_VALUE )
          // handle the error
   else   // proceed normally


   if  ( func2() == NULL )
          // handle the error
   else   // proceed normally


   if  ( func3() == 0 )
           // handle the error
   else   // proceed normally
}
```

5

Example : Handling Divide-by-zero error with IF statement

The division function below takes the numerator and denominator as parameters, calculates the result and returns it.

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

float  division  (int num,  int denom)  {
    return  float(num)  /  denom;
}

//--------------------------------------------------------------------
int main()  {
    int  numerator, denominator;
    cout << "Enter numerator : ";     cin >> numerator;
    cout << "Enter denominator : ";  cin >> denominator;

     if   (denominator == 0 )               // Check for zero
        cout << "Divide by zero error \n";   // Don't call the function
     else
         cout << division (numerator, denominator); // Call the function
}
```

6

Limitations of Error Handling with IF Statements

- The disadvantage of this approach is that every single call to a function must be examined by the program.
- Also, it is not practical for some functions to return an error value.
- Another disadvantage is that, it can not handle runtime errors occured in **class constructors**.
  The application can not find out whether an error occurred in the class constructor, because there is no return value (flag) to be checked.

Topics

- Exceptions (Runtime errors)
- Try-catch statement
- Throwing an exception
- Exceptions and class constructors

Try-catch statement

- The exception mechanism uses three C++ keywords: **throw**, **try**, and **catch**.

General Syntax

```
try  {
   Statements
}
catch  (object1)  {
  Block1
}
catch  (object2)  {
  Block2
}
catch  (object3)  {
  Block3
}
```

- If an error is detected in a function (class member or nonmember), this function informs the application that an error has occurred. This process is called throwing an exception**.**
- Any code in application that uses objects of a class is enclosed in a ***try block***.
- A try block should be followed by at least one ***catch block.*** It catches the exceptions thrown by the function.
- Each catch block is called an exception handler**.**

Example: Catching Memory Allocation Errors

- The **new** operator automatically throws an exception at run-time, if the specified RAM memory byte size is too large for the computer being used.
- The defult catch block below can detect runtime memory allocation errors.

```
#include <iostream>
using namespace std;
int main()
{
  double  * dizi;
  try
  {
        dizi = new  double [300000000];
  }
   catch  (...)  // Default catching block
   {
        cout << "Memory allocation error \n";
   }
}
```

$3 \times 10^8 \times 8$ bytes
≈ 2 GB memory

Topics

- Exceptions (Runtime errors)
- Try-catch statement
- Throwing an exception
- Exceptions and class constructors

The throw command

- The thrown exception object can be any variable or constant of any built-in data type (char, char *, int, float, etc.).
- It can also be a class object that defines the exception.

```
return_type    function_name ( parameters )
{
       if  ( exception_condition )
           throw   exception_object ;
           //Throw command causes exit from function immediately.

       // Normal operations
       return  expression ;
   }
```

Example : Handling Divide-by-zero error with try-catch and throw statements

The division function below throws an error message string as exception, if the denominator is zero.

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

float  division  (int num,  int denom)  {
     if (denom == 0)  throw "Divide by zero";  // Throw message as exception
     else  return  float(num)  /  denom;             // Normal operation
}

//----------------------------------------------------------------------------------------------------
int main()  {
     int  numerator, denominator;
     cout << "Enter numerator : ";      cin >> numerator;
     cout << "Enter denominator : ";  cin >> denominator;
      try  {
         cout << division (numerator, denominator);  // Call the function
     }
     catch  (const  char *  msg)  {
          cout << msg << endl;  // Display the thrown error message
    }
}
```

Multiple throw commands

- A function may throw more than one exception.
- The thrown exceptions can be different data types (char *, int, object, etc.)

```
float  division  (int num,  int denom)
{
    if (denom == 0)    throw  "Divide by zero";             // throws char *

    if (denom < 0)      throw  "Negative denominator";  // throws char *

    if (denom > 1000)  throw   -1;                               // throws int

    return  float(num)  /  denom;      // normal operation
}
```

Catch Block

- In a catch block, the type of the specified exception can be caught.
- The thrown parameter variable name can be omitted (not written), if it will not be used such as displaying.

```
catch  (const  char  * )
{
   cout << "ERROR";
}
```

The thrown char parameter is not used in the catch block.

Multiple Catch Blocks

If a function throws multiple exceptions of different data types, then a separate catch block must be written for each exception type.

```
int main()  {
  try  {
         cout << division (numerator , denominator);
  }

  catch  (const char * msg)  {   // Catch block for exceptions of type char *
         cout << msg << endl;
  }

  catch  (int)  {  // Catch block for exceptions of type int  (value is not taken)
         cout << "ERROR \n";
  }

  catch  ( ... )  {  // Ellipses indicate the default catching block (written at last)
         cout << "Default catching \n";
  }
}
```

Example: Using throw without try-catch

- If a throw command is written, but no try-catch statement is written, then program terminates at runtime.
- The Operating System displays its own error message.

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

int main()
{
   cout << "Program started. \n";  // Displayed

   throw  "Exception thrown. \n";  // Program terminates (this message not displayed)

   cout << "Program finished. \n"; // Not displayed
}
```

Screen Output

```
Program started.
Terminate called after throwing an instance of 'char const*'
```

Throwing an Object of User-written class

- Class **objects** can be thrown and caught as exceptions.
- Example: The user-written Stack class below has two member functions.
- If an error occurs, the push and pop member functions throw an object of user-written Error class.

```
class  Stack  {
        int  st [MAX];
        int   top;
      public:
        Stack ();
        void   push (int  data);
        int     pop ();
    };
```

```
void   Stack  ::  push (int data)
{
   if  (top > MAX-1)   // Check if stack is full
       throw  Error ("Stack is full");

   st [top]  =  data;
   top++;
}
```

```
int   Stack  ::  pop ()
{
  if  ( top <= 0 )  // Check if stack is empty
      throw   Error ("Stack is empty");
  else
      return  st [-- top];
}
```

User-written Error class and Main program

```
// Objects to be thrown
class  Error
{
  private:
    string   error_msg;  // Error message

  public:
    Error (string  m)  // Constructor
    {
          error_msg = m;
    }

    void print()
    {
          cout << error_msg << endl ;
    }
};
```

```
int main()
{
  Stack  s1;

  try
  {
     s1 . pop();
   }
   catch (const Error &  obj)
   // Exception handler
   {
       obj . print();
    }

}
```

Example: Chaining the Exceptions and Re-throwing

The f1 function below catches an exception thrown by f2 function.

Then, f1 calls the throw function for re-throwing the exception.

```
int  main ()  {
   try  {
      f1 ();
  }
catch (const char* msg)  {
   cout << "Main catches : "
          << msg << endl;
}
 return 0;
}
```

```
void  f1 ()  {
  try  {
     f2 ();
  }
 catch (const char * msg)  {
    cout << "f1 catches : "
           << msg << endl;
    throw msg; // Rethrows the exception
 }
}
```

Screen output

```
f1 catches : (Error thrown by f2)
Main catches : (Error thrown by f2)
```

```
void  f2 ()  {
     throw "(Error thrown by f2)";
}
```

Topics

- Exceptions (Runtime errors)
- Try-catch statement
- Throwing an exception
- Exceptions and class constructors

Exceptions and Class Constructors

- Exceptions are necessary to find out if an error occurred in the class constructor functions.
- Constructor functions are called implicitly and there's no return value to be checked.
- **Example:** The constructor of the String class does not allow the contents to be longer than 10 characters.

```
class  String
{
      const  int  MAX_SIZE = 10;       // MAX_SIZE is a constant
      int      size;
      char * contents;
 public:
      String (const char *);           // Constructor
      void print() const;
      ~String();                       // Destructor
};
```

Example: String class constructor

If a string is longer than MAX_SIZE (10) characters, an exception is thrown in constructor function, and the object is not created.

```
String  ::  String (const char *  in_data)   // Constructor
{
      size  =  strlen (in_data);

      if  (size  >  MAX_SIZE)
         throw   "Too long string" ;
         //Throw command exits from constructor function immediately.

      // Proceed below (normal operations) if there was no throw.
      contents  =  new  char [size +1];
      strcpy (contents,  in_data);
 }
```

Example: Main program

```
int main()
{
        char  input [20];   // To take chars from keyboard
        String  *str;        // Pointer to String object

           cout << " Enter a string: ";
           cin >> input;   // Reads as chars

           try
           {
               str  =  new  String (input);   // Calls the constructor
           }
           catch (const char * msg)
           {
                cout << "An exception caught : " << msg << endl;
                return  0;  // Stops the program
           }

          str -> print();
          delete  str;
}
```