C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

Exception Handling

- If we give wrong input to the application then it generates runtime error/exception.
- Exception is an object, which is used to send notification to the end user of the system if any exceptional situation occurs in the program.
- To handle exception then we should use 3 keywords:
- 1**. try**
  - try is keyword in C++.
  - If we want to inspect exception then we should put statements inside try block/handler.
  - Try block may have multiple catch block but it must have at least one catch block.
- **2. catch**
  - If we want to handle exception then we should use catch block/handler.
  - Single try block may have multiple catch block.
  - Catch block can handle exception thrown from try block only.
  - A catch block, which can handle any type of exception is called generic catch block / catch-all handler.
  - For each type of exception, we can write specific catch block or we can write single catch block which can handle all types of exception. A catch block which can handle all type of exception is called generic catch block.
- **3. throw**
  - throw is keyword in C++.
  - If we want to generate exception explicitly then we should use throw keyword.
  - "throw statement" is a jump statement.
  - To generate new exception, we should use throw keyword. Throw statement is jump statement.

Note : For thrown exception, if we do not provide matching catch block then C++ runtime gives call to the std::terminate() function which implicitly gives call to the std::abort() function.

Consider the following code

- In C++, try, catch and throw keyword is used to handle exception.

```
int num1;
accept_record( num1 );
int num2;
accept_record( num1 );
try {
   if( num2 == 0 )
   throw "/ by zero  exception";
   int result = num1 / num2;
   print_record( result )
}
catch( const char *ex ){
   cout<<ex<<endl;
}
catch(...)
{
cout<<"Genenric catch  handler"<<endl;
}
```

Consider the following code

In this code, int type exception is thrown but matching catch block is not available. Even generic catch block is also not available. Hence program will terminate. Because, if we throw exception from try block then catch block can handle it. But with the help of function we can throw exception from outside of the try block.

```
int main( void ){
    int num1;
    accept_record(num1);
    int num2;
    accept_record(num2);
try {
    If( num2 == 0 )
                throw 0;
    int result = num1 / num2;
    print_record(result);
}
catch( const char *ex ){
    cout<<ex<<endl;  }
    return 0;
}
```

Consider the following code

If we are throwing exception from function, then implementer of function should specify “exception specification list”. The exception specification list is used to specify type of exception function
may throw.
If type of thrown exception is not available in exception specification list and if exception is raised then C++ do execute catch block rather it invokes std::unexpected() function.

```
int calculate(int num1,int num2) throw(const char* ){
    if( num2 == 0 )
                throw "/ by zero  exception";
    return num1 / num2;
}
int main( void ){
    int num1;
    accept_record(num1);
    int num2;
    accept_record(num2);
try{
   int result = calculate(num1, num2 );
    print_record(result);
}
catch( const char *ex ){
cout<<ex<<endl; }
return 0;  }
```

Run-Time Type Information

- Run-time type information (RTTI) is a mechanism that allows the type of an object to be determined during program execution.
- There are three main C++ language elements to run-time type information:
- The dynamic_cast operator.
  - Used for conversion of polymorphic types.
- The typeid operator.
  - Used for identifying the exact type of an object.
  - typeid(Base *).name();
  - If Base ptr holds null then type_id throws bad_typeid exception
- The type_info class.
  - Used to hold the type information returned by the **typeid** operator.

Casting Operators

- dynamic_cast Used for conversion of polymorphic types.
  - dynamic_cast < type-id > (expression)
  - dynamic cast returns NULL if the cast to a pointer type fails
  - The **bad_cast** exception is thrown by the **dynamic_cast** operator as the result of a failed cast to a reference type.
- static_cast Used for conversion of nonpolymorphic types.
  - Risky conversion not be used, should only be used in performance-critical code when you are certain it will work correctly
  - The **static_cast** operator can be used for operations such as converting a pointer to a base class to a pointer to a derived class. Such conversions are not always safe.
- const_cast Used to remove the **const**, **volatile**, and **__unaligned** attributes.
  - const_cast<class *> (this)->membername = value;
- reinterpret_cast Used for simple reinterpretation of bits.
  - Allows any pointer to be converted into any other pointer type.
  - The **reinterpret_cast** operator can be used for conversions such as **char*** to **int***, or One_class* to Unrelated_class*, which are inherently unsafe.

Thank You