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;  }
```

Template

- If we want to write generic program in C++, then we should use template.
- This feature is mainly designed for implementing generic data structure and algorithm.
- If we want to write generic program, then we should pass data type as a argument. And to catch that type we should define template.
- Using template we can not reduce code size or execution time but we can reduce developers effort.

Template

int num1 = 10, num2 = 20; swap_object<**int**>**(num1, num2);** string str1="Pune", str2="Karad"; swap_object<string>(str1, str2);	In this code, <int> and <string> is considered as type argument.

template<typename T> //or
template<class T> //T : Type Parameter
void swap(b obj1, **T** obj2 **)**
{
T temp = obj1;
obj1 = obj2;
obj2 = temp;
}
	template and typename is keyword in
	C++. By passing datatype as argument
	we can write generic code hence parameterized type is called template

- **Types of Template**
  - Function Template
  - Class Template

Example of Function Template

```
//template<typename T>//T : Type Parameter
template<class T> //T : Type Parameter
void swap_number( T &o1, T &o2 )
{   T temp = o1;
    o1 = o2;
    o2 = temp;
}
 int main( void )
{
    int num1 = 10;
    int num2 = 20;
    swap_number<int>( num1, num2 );     //Here int is type argument
    cout<<"Num1 : "<<num1<<endl;
    cout<<"Num2 : "<<num2<<endl;
    return 0;
}
```

Example of Class Template

```
template<class T>
class Array // Parameterized type
{
private:
   int size;
   T *arr;
   public:
   Array( void ) : size( 0 ), arr( NULL )
   {
   }
   Array( int size )
   {
   this->size = size;
   this->arr = new T[ this->size ];
   }
   void acceptRecord( void ){}
   void printRecord( void ){ }
   ~Array( void ){ }
};
```

```
int main(void)
{
Array<char> a1( 3 );
a1.acceptRecord();
a1.printRecord();
return 0;
}
```

Thank You