Academic Year: 2020

Semester: 2nd

Course Code: CS-241L

Course Title: Object Oriented Programming

CS-241L Object Oriented Programming Lab 10

Type of Lab: Open Ended **Weightage: 5%**

CLO 3: Select programming API functionality and incorporate them into object design.

Student understand the concept of Templates and Exception handling	**Cognitive/Understanding**	CLO3	Rubric A

Rubric A: Cognitive Domain

Evaluation Method: GA shall evaluate the students for Question according to following rubrics.

CLO	**0**	**1**	**2**	**3**	**4**	**5**
CLO3	Unable to understand and implement	Student understand composition	Student understand Templates and exception handling	Implement Templates and Exception handling	Student solve class tasks partial	Student solve class tasks completely

Lab 10

BS-Computer Science

Object Oriented Programming

Target: Templates and Exception Handling in C++

Templates: Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.

Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

A function template works in a similar to a normal function, with one key difference.

A single function template can work with different data types at once but, a single normal function can only work with one set of data types.

Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration.

However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

Function Template:

A function template starts with the keyword template followed by template parameter/s inside **< >** which is followed by function declaration.

How to declare a Function template?

```
template <class T>
T someFunction(T arg)
{
    ... .. ...
}
```

- In the above code, T is a template argument that accepts different data types (int, float), and class is a keyword.
- You can also use keyword typename instead of class in the above example.
- When, an argument of a data type is passed to someFunction( ), compiler generates a new version of someFunction() for the given data type.

Class Templates:

Like function templates, you can also create class templates for generic class operations.

Sometimes, you need a class implementation that is same for all classes, only the data types used are different.

Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class.

This will unnecessarily bloat your code base and will be hard to maintain, as a change is one class/function should be performed on all classes/functions.

However, class templates make it easy to reuse the same code for all data types.

How to declare a class template?

```
template <class T>
class className
{
    ... .. ...
    public:
        T var;
```

```
        T someOperation(T arg);
        ... .. ...
};
```

- In the above declaration, T is the template argument which is a placeholder for the data type used.
- Inside the class body, a member variable var and a member function someOperation() are both of type T.

How to create a class template object?

To create a class template object, you need to define the data type inside a <> when creation.

className<dataType> classObject;

For example:

className<int> classObject;

className<float> classObject;

className<string> classObject;

Exception Handling:

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

- throw − A program throws an exception when a problem shows up. This is done using a throw keyword.
- catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- try − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Why Exception Handling?

- Following are main advantages of exception handling over traditional error handling.
  1. Separation of Error Handling code from Normal Code: In traditional error handling codes, there are always if else conditions to handle errors. These conditions and the code to handle errors get mixed up with the normal flow. This makes the code less readable and maintainable. With try catch blocks, the code for error handling becomes separate from the normal flow.
  2. Functions/Methods can handle any exceptions they choose: A function can throw many exceptions, but may choose to handle some of them. The other exceptions which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them, then the exceptions are handled by caller of the caller. In C++, a function can specify the exceptions that it throws using the throw keyword. The caller of this function must handle the exception in some way (either by specifying it again or catching it)
  3. Grouping of Error Types: In C++, both basic types and objects can be thrown as exception. We can create a hierarchy of exception objects, group exceptions in namespaces or classes, categorize them according to types.

Code within a try/catch block is referred to as **protected code**, and the syntax for using try/catch as follows.

```
try {
  // protected code
} catch( ExceptionName e1 ) {
  // catch block
} catch( ExceptionName e2 ) {
  // catch block
} catch( ExceptionName eN ) {
  // catch block
}
```

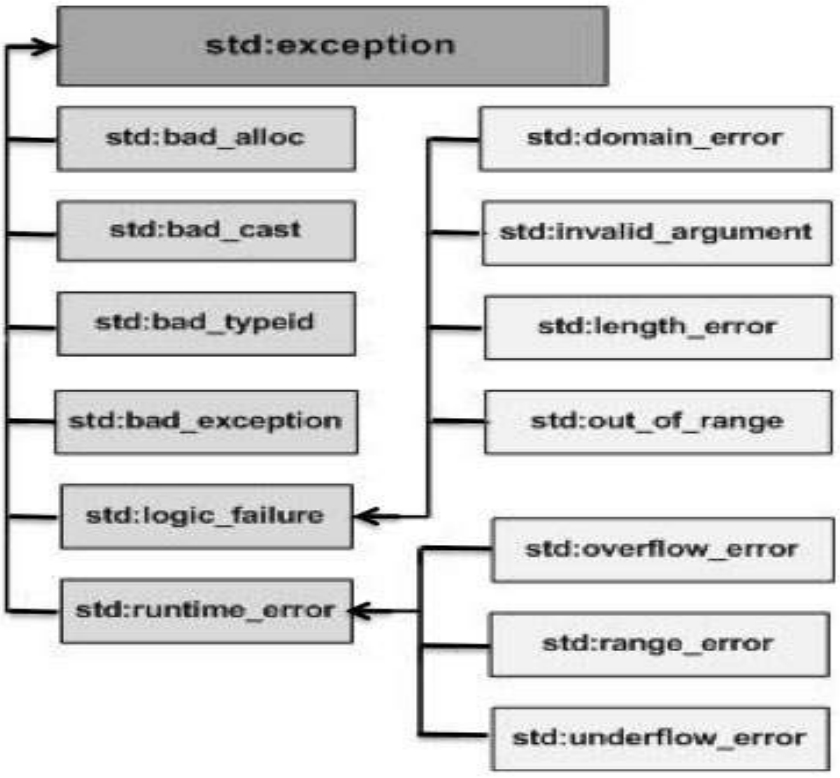
You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations. The following is an example, which throws a division by zero exception and we catch it in catch block.

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

```
double division(int a, int b) {
  if( b == 0 ) {
     throw "Division by zero condition!";
  }
  return (a/b);
}

int main () {
  int x = 50;
  int y = 0;
  double z = 0;

  try {
     z = division(x, y);
     cout << z << endl;
  }
  catch (const char* msg) {
    cerr << msg << endl;
  }

  return 0;
}
```

C++ Standard Exceptions

C++ provides a list of standard exceptions defined in <exception> which we can use in our programs. These are arranged in a parent-child class hierarchy shown below.



Home Task

Task 1:

Write a Program to display largest among two numbers using function templates.

Task 2:

Write a Program to add, subtract, multiply and divide two numbers using class template

Task 3:

Write any example for exception handling using c++ standard exception. For example std::domain_error (This is an exception thrown when a mathematically invalid domain is used).

Submission:

1. Three .cpp files with name task 1.cpp , task2. cpp, task 3.cpp.
2. Copy all .cpp files and paste in a folder and rename that folder with your registration number as 2019-BS-CS-001.
3. Upload on EDUKO before deadline.