CA3 Set-B

Course Code - CAP444

Course Title - Object Oriented Programming using C++

Name - Sirjanpreet Kaur

Section - D2112

Roll no - B56

Reg. no - 12107974

Name: Sanjanpreet Kaur Roll no: B56

Reg. no: 12107974 Ques no. 1

Q: Explain the exception handling mechanism in detail. Discuss the multiple catch statements with the help of program.

Ans The errors that occur at run time is known as exception. They occur due to different conditions such as division by zero, unable to open a file, running out of memory, accessing element that is out of bound of an array and many more. Apart from this, there are other type of exceptions like errors associated with events like disk I/O completion etc. So for handling these type of exceptions, Excepting Handling is there. Exception Handling is a mechanism to handle such errors. But Exception Handling is only used to handle synchronous exceptions. Synchronous exceptions like division by zero, element out of bound of an array, running out of memory etc. The main objective of exception handling is a way to detect and report the exception so that it can be handle on time without troubling.

Exception Handling Mechanism

The exception handler is required to handle exception whenever any exception occurs, the part of program in which exception occur, there may be applied some mechanism that detects the exception. The portion of program that detect the exception can inform that

exception has occured by throwing it. On throwing an exception, the program immediately stops the execution of the code and jumps to the separate code of Block that is known as Exception Handler. Exception Handler handles the exception and proces it without unnecessarily troubling the user, otherwise the program terminates abnormally and may cause system failure.

There are three constructs for implementing exception Handling i.e try, throw and catch. Firstly there is a try block. try block always use with catch block. In try block, the code is placed in try block that needs to be tested for exception. If there is issue in code in try block, then we use the throw statement to invoke an exception handler. Exception Handler is catch block. The error that occur in try block immediately follows the catch block. The catch block handles that exception. The syntax of try-catch block is:-

```
try
{
  // testing condition
    throw excep ;
}
catch (datatype arg)
{ // code for handeing exception
}
```

In the above syntax, there is a keyword try. The try keyword is followed by a set of statements. The statements that will cause exceptions at run time may be put in try block. On detecting an error or exception, it is thrown using a throw statement. The throw statement is in try block. The throw keyword is used to throw an exception and excep represents the type of value to be thrown ie whether it is of int type, char type etc. When try block throw the exception, that exception is handled by catch block. The catch block starts with keyword catch that is followed by set of statements. The catch keyword provide some parameter. The arguments must be passed in parameters of catch. In catch, an exception parameter arg is passed along with datatype. If the type of value (excep) thrown matches the datatype of arg in catch block, then the catch block is executed for processing the exception. If no such mechanism passes, then the program will terminate abnormally. If there is no exception occurs in try block, then the program skips the catch block and continues with the next statements of code after try-catch block.

Name: Sirjanpreet kaur Roll no: B56

Reg. no: 12107974 Quesno: 1

Program:-

```
// Program to illustrate how an exception is handled using
  try-catch block and throw statement
#include<iostream>
using namespace std;
int main()
{
   int num1, num2;
    float res;
    cout<<"Enter the dividend :";
    cin>>num1;
    cout<<"\n";
    cout<<"Enter the divisor :";
    cout>>num2;
    cout<<"\n";
//exception handling begins here
try //try block, it will execute the statements of try block
{
if(num2 !=0) //checking if divisor is zero, if it is not zero then
                //the program execute successfully
{
res = num1/num2; //it will store the result of division of
                  //num1 and num2
cout<<"Result is :"<<res<<endl; //if user pass the value of
//num2 other than 0 then Result is shown
}
```

```
else
{
 throw 10;// if user put the value of divisor to 0, then it will
          // throw an exception
 // throwing the exception, it matches with the data type of
   excep in catch block
 }
}
//catch block
catch (int excep) // catches the exception
{
 cout<<"Division by zero is not possible"; // it will print
                    // this message if exception occurs
}
}
```

Output:

```
Enter the dividend : 34
Enter the divisor : 0
Division by zero is not possible
```

Name: Sirjanpreet Kaur Roll no: B56

Reg. no 12107974 Ques no: 1

In this program, the purpose of program is to display the result of division of two numbers and check for division by 0 exception. The program on execution first prompts the user to input the value of dividend and divisor both of type int. In try block, we chect the value of divisor. If it is 0, an exception is detected and thrown using the throw statement with an integer value 10. The catch block catches the exception as the type of value thrown matches with the parameter type of catch statement. In the catch block, we display the appropriate message and exit from the program. If the value of divisor is non zero, then program ignores the catch block and continues with statement after catch block if any statement is there.

MULTIPLE CATCH BLOCKS

In program, they may be different type of exceptions occur in the program. In order to handle multi-ple exceptions of different types, we need to define multiple catch blocks associated with a single try block. We can have as many catch blocks with a single try block. There can not be multiple try blocks. The multiple catch blocks will be executed for handling different type of exceptions. But which

catch block will handle which type of exception is determined by the type of value thrown from the try block. The syntax is :-

```
try
{
  // testing condition
 throw except1;
  // testing condition
   throw except2;
  // testing condition
   throw except3;
};
catch (datatype1 arg1)
 {
  // code for handling exception
 }
catch (datatype2 arg2)
 {
  // code for handling exception
 }
catch (datatype3 arg3)
 {
  // code for handling exception
 }
```

Name: Harjanpreet Kaur Roll no: B56

Reg no: 12307974 Ques no: 1

The multiple catch blocks takes paramsters of different data types and each must match a different type of exception. When an exception is thrown from the statement in try block, then multiple catch blocks that follow the try block is searched in order they appear in program. If the type of value thrown matches with the parameter type of catch statement, then that catch block is selected to handle the exception. Rest of the catch blocks are ignored if the specified catch block found. Then the program continues with the next statements following last catch block. If catch blocks are not matched, then the thrown exception will not be proceded and it will terminate abnormally.

Program:

```
// Program to print table of any number using multiple
// catch statements

#include<iostream>
#include<math.h>
using namespace std;
```

Name: Sirjanpreet Kaur Roll no: B56

Reg no: 12107974 Ques no: 1

```
int main()
{
  int num;
  float res;
  cout<<"Enter a number :"; //number entered by user
  cin>>num;
  try
  {
    if (num<0)  // if number is less than 0
      throw 10; //throw int type exception that matches with
                //datatype in catch parameters

    if(num>0xffff) //if number entered is out of scope because
                   //the accepted ranges is 0x0000 to 0xFFFF
      throw 'E';
      cout<<"Table of "<<num<<" is "<<endl;

    for(int i=1; i<=10; i++) //loop continues till i is less
                             //than or equal to 10
    {
      cout<<num<<"*"<<i<<"="<<num*i<<endl; //print
                //the table of number entered by user
    }
  }
  catch (int) // catches the exception of integer type
  {
    cout<<"Table of negative number is not possible"<<endl;
  }
```

```
catch (char) //catches the exception of character type
{
 cout<<"Out of Range "<<endl;
 }
 return 0;
}
```

Output:

1.

```
Enter a number : -4
Table of negative number is not possible
```

2.

```
Enter a number : 5
Table of 5 is
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Name: Shivanpreet Kaur Roll no: B56

Reg no: 12107974 Ques no: 1, 2

In the above program, the table of any number is to be print. In this we deal with two type of exceptions. The first exception occurs when a number entered by the user is negative number. When user inputs a negative number then exception is detected and we throw a exception with a value of int type. Similarly, the second exception occurs when a number entered by the user is out of range. When the user inputs any number out of range, then exception is detected and we throw the exception with a value of int type.

2. Write a program to achieve the following:
(a) Implementing class template
(b) Implementing swap operation using function template.

Ans (a) Implementing class Template

Like function templates, we can create generic class templates that provide same functionality for different data types. Once the code is written as a class template, it can support multiple data types. The Syntax is:

```
template < class TY
class class_name
{
    // class definition
};
```

Name: Harjanpreet Kaur Roll no: B56

Reg no: 12107974 Ques no: 2

The class template always begin with keyword template followed by angular brackets < > which specify the type parameter T. The keyword class preceded by type parameter T indicates that T represents built-in or user-defined data type.

Program:-

```
// Program to implement class Template
#include <iostream>
using namespace std;
template <class T> // template class
class CTemp // create class template
{
  public :
    T temp[2];
    CTemp (T a1, T a2) // created constructor and pass parameter
                       // of T type
    {
      temp[0] = a1;
      temp[1] = a2;
    }
    void show()  // create normal function
    {
      cout << "value of a is : " << temp[0] << endl;
      cout << "value of b is : " << temp[1] << endl;
    }
};
```

Name: Harjanpreet Kaur Roll no: B56

Reg. no: 12107974 Ques no: 2

```
int main()
{
  cout<<"Passing Integer type values"<<endl;
  CTemp<int> ct1(32,45); //created object and pass int
                         //type values

  ct1.show(); //calling show function
  cout<<"\n";

  cout<<"Passing String type values"<<endl;
  CTemp<string> ct2("c1","c2"); //created object and
                                //pass string type values

  ct2.show(); //calling show function

  return 0;
}
```

Output:

```
Passing Integer type values
Value of a is : 32
Value of b is : 45

Passing String type values
Value of a is : c1
Value of b is : c2
```

In this program, CTemp<int> ct1(32,45) creates an object ct1 of type CTemp that can store int values. We pass parameters of int type that will pass to the constructor parameters and on calling the show function, it will be displayed. Similarly, CTemp<string> ct2("c1","c2") creates an object ct2 of type CTemp that will store string type values. On passing values to parameterized constructor, it will store the values of string type and on calling the show function, it will be displayed on the output screen.

(b) Implementing Swap operation using function template

function templates are the generic functions which can handle different data types without the need of writing separate code. Using function template, the same code need to be repeatedly written for different data types, reduces a lot of manual effort in development.

The syntax of function template is:-

```
template <class T>
T function_name(T par1, T par2, ...)
{
  // template function body
}
```

Name: Sirjanpreet Kaur
Roll no: B56
Reg no: 12107974
Ques no: 2

Function template begin with template that informs the compiler that we are about to define a function template. It is followed by template type parameter T in angled brackets and parameter T must be preceded by keyword class which establish that T is a datatype

Program:

```
//Program to implement swap operation using function
//template
#include<iostream>
using namespace std;
template<class T>                    //declare template
void swapping(T &x, T &y) //create function template and
{                                    //pass parameters of T type
   T temp;
   temp=x;
   x=y;
   y=temp;
}
int main()
{
   int x,y;
   cout<<"Enter Integer Type Values:";
   cout<<"Enter value of x:"; //value entered by user
   cin>>x;
```

```
cout<< "Enter value of y:";
cin>>y;
cout<<"\n Before Swapping :";
cout<<"\n The value of x is :"<<x;
cout<<"\n The value of y is :"<<y;
swapping (x,y);   //calls function template for int
cout<<"\n";
cout<<"\n After Swapping :";
cout<<"\n The value of x is :"<<x;
cout<<"\n The value of y is :"<<y;
cout<<"\n";
float p,q;
cout<<"\n Enter Float Type Values :";
cout<<"\n Enter value of p:";
cin>>p;
cout<<"\n Enter value of q:";
cin>>q;
cout<<"\n Before Swapping :";
cout<<"\n The value of p is :"<<p;
cout<<"\n The value of q is :"<<q;
swapping (p,q);   //calls function template for float
cout<<"\n";
cout<<"\n After swapping :";
cout<<"\n The value of p is :"<<p;
```

```
cout<<"\n The value of q is :"<<q;
cout<<"\n";
char c1,c2;
cout<<"\n Enter character Type values :";
cout<<"\n Enter value of c1 :";
cin>>c1;
cout<<"\n Enter value of c2:";
cin>>c2;
cout<<"\n Before Swapping :";
cout<<"\n The value of c1 is :"<<c1;
cout<<"\n The value of c2 is :"<<c2;
swapping (c1,c2);   //calls function template for char
cout<<"\n";
cout<<"\n After Swapping :";
cout<<"\n The value of c1 is :"<<c1;
cout<<"\n The value of c2 is :"<<c2;

return 0;
}
```

Output:-

```
Enter Integer Type values:
Enter value of x : 2
Enter value of y : 8

Before Swapping :
The value of x is : 2
The value of y is : 8
```

```
After Swapping:
The value of x is :8
The value of y is :2

Enter Float Type values:
Enter value of p : 5.8
Enter value of q : 9.7
Before Swapping:
The value of p is : 5.8
The value of q is : 9.7
After Swapping:
The value of p is : 9.7
The value of q is : 5.8

Enter Character Type values:
Enter value of c1 : s
Enter value of c2 : k

Before Swapping:
The value of c1 is : s
The value of c2 is : k

After Swapping:
The value of c1 is :k
The value of c2 is : s
```

Name: Sirjanpreet Kaur Roll no: B56

Reg. no: 12107974 Ques no: 3

3. Discuss recursion with template function and difference between templates and macros.

Ans The process in which a function call by itself is called recursion. Recursion is useful for writing repetitive problems where each action is stated in terms of previous result. The need of recursion is important if logic of program is such that the solution of problem depends upon the repetition of set of statements and end with a condition. The requirements of recursion are:-

1. The function must call itself again and again
2. It must have an exit condition

In templates also, we can apply Recursion. In this a template will be created and it will called by itself. We can create Recursive template function also. Function Templates are function that are written with placeholder to return any data type. In order to do recursion with template function, we have to first make template function and pass parameters in that function. Then in the body of the template function, we have to write the code that will repetedly doing calling itself. This can be understanded by the following program:-

Name: Sirjanpreet Kaur
Roll no: B56
Reg no: 12107974
Ques no: 3

Program:-

```
// To find the sum of all even number in an array using
// recursive function template

#include <iostream>
using namespace std;

template <class T>
T sumEven(T arr[], T i, T sum) //template function to
   //return sum of even numbers in an array of size n
{
 if (i<0)
 {
   cout<<"Sum of even number is "<< sum;
   return 0;
 }
 if ((arr[i])%2==0) //checking that element is divisible by 2
 {
   sum+=(arr[i]); //if true then elements are
                  //added to sum
 }
 sumEven(arr, i-1, sum);
}

int main()
{
```

Name : Sirjanpreet Kaur Role no: B56

Reg no: 12107974 Ques no: 3

```
int arr[] = { 3, 4, 5, 34, 5, 8, 10, 2 }; // passing values
int n = sizeof(arr)/sizeof(arr[0]);
int sum = 0;
SumEven(arr, n-1, sum);
return 0;
}
```

Output :-

```
Sum of even number is 58
```

In this program, we have define template and pass parameters, T arr[] means we can pass any array. We print sum of even numbers. In this, we check condition if a number is divisible by 2, if it is divisible by 2, then the element is added to sum, again the condition is checked and if condition is true, then element is added to sum. In this way, sum of even numbers are printed by using recursive template function

Name: Sirjanpreet kaur
Reg no: 12107974
Roll no: B56
Ques no: 3

Difference between Template and Macros

	TEMPLATES	MACROS
1.	Templates are generic function that can be used for different data types	Macros are preprocessor statements that gives instructions to compiler.
2.	Template function is compiled means compiler checks types before creating a new class.	Macro is preprocessed that means it preprocess the information before actual compilation starts.
3.	Templates takes longer time to process	Macros take less time to process
4.	The function template used to calculate the minimum of two values can be:-	The macros used to calculate the minimum of two values can be:-

```
template < class T >
T min (T a, T b)
{
  return (a < b) ? a : b;
}
```

```
#define min(a,b) ((a<b)?a:b)
```

On calling this macro :-

```
min(5,2);
min(20.5,2.5);
```

Name: Sirjanpreet Kaur
Roll no: B56
Reg. no: 12107974
Ques no:- 3

	TEMPLATES	MACROS
5.	In function Template, full type checking is performed by compiler	In Macros, no type checking is performed by the compiler and preprocessor dumbly replaces any call to the macros with their expansion
6.	In Templates, always correct results are obtained.	In macros, it can cause unexpected result.
7.	Template functions keeps the code length unaffected	Macro increases the code length
8.	We can have recursive template. So manual time of development decreases	While in Macros, macros cannot call themselves and it is limited to a single expansion
9.	Templates are used for writing large codes when program need more lines of code	Macros are used in programs where small code is required.
10.	During function call, transfer of control takes place	While in macros, macro name is replaced by macro value

Name: Sirjanpreet Kaur
Reg. no: 12107974

Roll no: B56
Ques no: 3

	TEMPLATES	MACROS
1.	The speed of template functions are lower for execution	The speed of Macros is faster for execution
12	The templates are of two types :- function templates and class templates	# is used to define a macro and it is actually a piece of code
13	Using templates, we can declare a class and define it. We can create other classes from template class In case of inheritence	In case of Macros, we cannot do these functionalities
14	Speed of execution is slower than macros	Macros are faster than templates
15	The templates are function Template and class Template	The predefined macros are -LINE-, -FILE-, -DATE-, -TIME- et