OOP USING C++
EXCEPTION HANDLING

EXCEPTION HANDLING

- When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

- When an error occurs, C++ will normally stop and generate an error message. The technical term for this is: C++ will throw an exception (throw an error).

- Exception is an event which occurs during execution of program that disrupts normal flow of program

HOW TO HANDLE THE EXCEPTION: EXCEPTION HANDLING

- Exception Handling is a process to handle runtime errors.
- We perform exception handling so the normal flow of the application can be maintained even after runtime errors.
- exception is an event or object which is thrown at runtime.
- All exceptions are derived from **exception** class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

EXCEPTION HANDLING: CONTINUED

- Example of Exception like Divide by zero, Accessing array element beyond its limit, running out of memory etc.

- Exception handling mechanism consists of followin parts:

1) Find the problem(Hit the Exception)
2) Inform about its occurrence(Throw the exception)
3) Receive error information(Catch the exception)
4) Take proper action(Handle the exception)

WHY EXCEPTION HANDLING

- Seperation of Error handling code from normal code
- functions can handle any exceptions they choose
- Grouping of error types

EXCEPTION HANDLING : CONTINUED

- **Exception Handling can be done in 3 ways:**

1) **try block**: try block is used to place the code that may occur exception. Exception are thrown inside the try block.

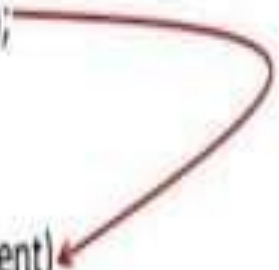
2) **catch block**: catches an exception which is thrown from try block. The catch keyword indicates the catching of an exception. It is used to handle the exception. It defines action to be taken when exception occur

3) **throw keyword:** throws an exception when a problem is detected.

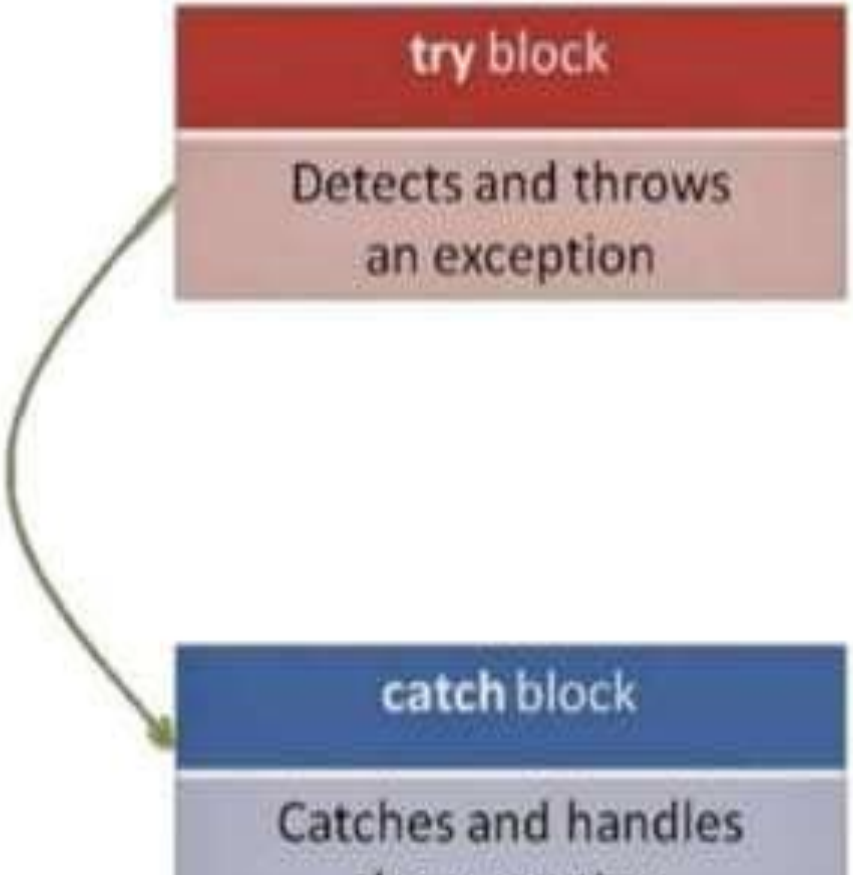
SYNTAX OF EXCEPTION HANDLING

```
try
{
  statements;
  ... ... ...
  throw exception;
}

catch (type argument)
{
  statements;
  ... ... ...
}
```

```
try
{
  statements;
  ... ... ...
  throw exception;
}

catch (type argument)
{
  statements;
  ... ... ...
}
```



EXCEPTION HANDLING MECHANISM



EXAMPLE WITHOUT EXCEPTION HANDLING

```
#include<iostream>
void main()
{
    int a,b;
    a=10;
    b=a/0;  //exception occurs
     cout<<"result: "<<b;
}
```

Here program will be terminated you will not get output because exception occurs at line 6 and flow of program comes out of main() function without executing line 7.

Exception Handling

```
#include<iostream>
void main()
{
   int a,b;
   a=10;
  try
  {
   b=a/0;  //exception occurs
   }
   catch(const char* e)
   {
    cout<<"Divide by zero error" //exception handler code
    }
```

EXAMPLE OF EXCEPTION HANDLING

```
#include<iostream>
void main()
{
    int n1,n2,result;
    cout<<"Enter first number: ";
    cin>>n1; n1=12
    cout<<"Enter second number: "
    cin>>n2; n2=20
    try
    {
        if(n2==0)
        {
            throw n2;
        }
        else
        {
            result=n1/n2;   12/20=0.....
        }
    }
    catch(int x) //x=0 n2=0
    {
        cout<<"Can't divide by  "<<x;
    }
    cout<"End of the program";
}
```

Note: Guess the output

1) n1=45, n2=0

2) n1=12, n2=20

EXAMPLE OF EXCEPTION HANDLING

```
#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;
}
```

MULTIPLE CATCH EXCEPTION

- Multiple catch statements are used in case of more than one exceptions.
- For handling multiple exceptions we can write multiple catch statements with different declarations.
- syntax of multiple catch statements:

```
try {
   body of try block
}
catch (type1 argument1) {
   statements;
   ... ... ...
}
catch (type2 argument2) {
   statements;
   ... ... ...
}
... ... ...
... ... ...
catch (typeN argumentN) {
   statements;
   ... ... ...
```

CATCH ALL STATEMENTS

- In some cases it is not feasible to write multiple catch blocks for each kind of exception in such cases we can write single catch block which catches all the exceptions

- syntax:

```
catch(....)
{
    statements;
    ..................
}
```

EXAMPLE OF EXCEPTION HANDLING 2

```
int main() {
  int n1,n2;
  cout << "Enter 1st number: ";
  cin >> n1;
  cout << "Enter 2nd number: ";
  cin >> n2;
  try {
    if (n2 != n1) {
      float div = (float)n1/n2;
      if (div < 0)
        throw 'e';
      cout << "n1/n2 = " << div;
    }
    else
```

```
catch (int e)
  {
    cout << "Exception: Division by zero";
  }
  catch (char st)
  {
    cout << "Exception: Division is less than 1";
  }
  catch(...)
  {
    cout << "Exception: Unknown";
  }
  return 0;
```

EXAMPLE CONTINUED

- GUESS THE OUTPUT FOR FOLLOWING INPUT:

1) n1=20, n2=5

2) n1=5, n2=20

3) n1=-1, n2=20

EXAMPLE OF MULTIPLE CATCH BLOCKS

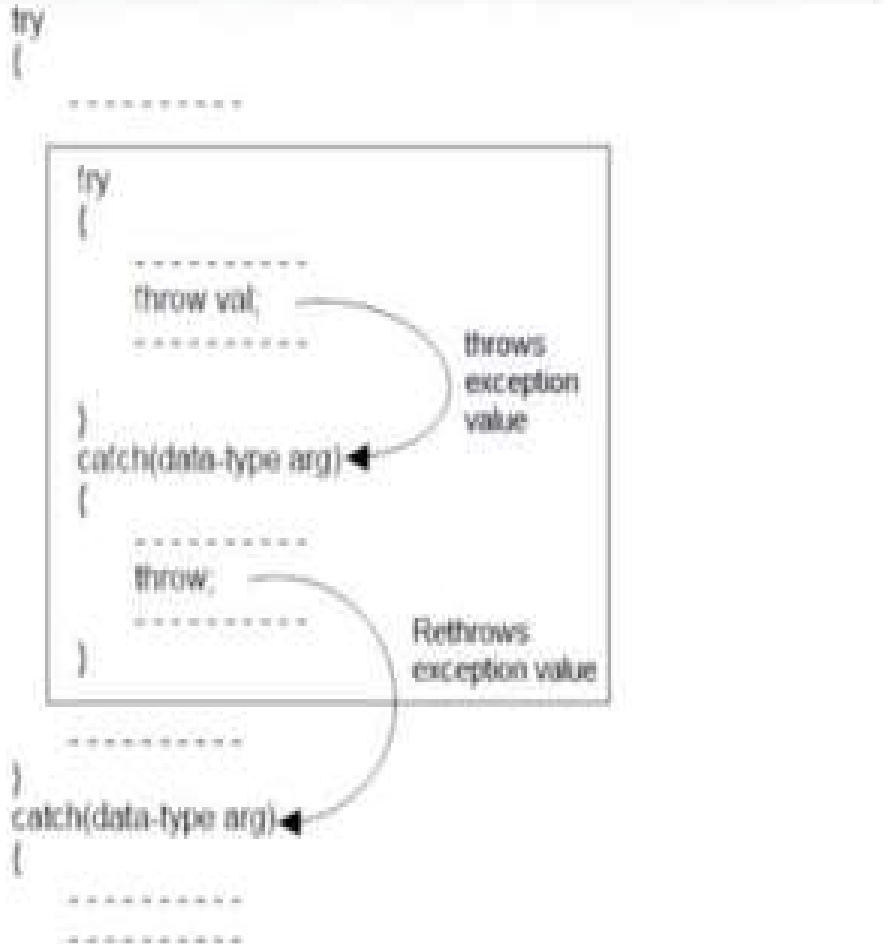
```
int main()
{
  int a=2;
      try
      {
          if(a==1)
              throw a;  //throwing integer exception
          else if(a==2)
              throw 'A'; //throwing character exception
          else if(a==3)
              throw 4.5; //throwing float exception
```

```
catch(int a)
  {
     cout<<"\nInteger exception caught.";
  }
catch(char ch)
  {
     cout<<"\nCharacter exception caught.";
  }
catch(double d)
  {
     cout<<"\nDouble exception caught.";
  }
cout<<"\nEnd of program.";
```

RETHROWING EXCEPTION

- We can rethrow the exception where we can have inner and outer try-catch statements(nested try-catch).

- Throwing of exception from inner catch block to outer catch block is called **Rethrowing Exception**



EXAMPLE OF RETHROWING EXCEPTION

```
int main()
{
  int a=1;
  try
  {
    try
    {
      throw a;
    }
    catch(int x)
    {
      cout<<"\nException in inner try-catch block.";
      throw x;
    }
```

```
catch(int n)
  {
    cout<<"\nException in outer try-catch block.";
  }
cout<<"\nEnd of program.";
}
```

DEFINE NEW EXCEPTION

- In this user can create and throw its own exception with the help of throw statement

EXAMPLE OF CUSTOM EXCEPTION

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

class Demo {
   int num;

public:
   demo(int x)
   {
      try {
            if (x == 0){
            throw "Zero not allowed ";
            }
        num = x;
        show();
      }
catch (const char* exp) {
        cout << "Exception caught \n ";
        cout << exp << endl;
      }
   }
   void show()
   {
      cout << "Num = " << num << endl;
   }
};

void main()
{
   Demo d = demo(0);
   Demo d1= demo(1);
}
```

Q1. Write a C++ program that takes an integer input from the user to access an element from an array of fixed size. Implement exception handling to catch an "out of bounds" error. If the user tries to access an index that is outside the valid range of the array, throw an exception and display an appropriate error message. The program should prevent the user from crashing the program due to invalid array access.

Q2. Create a C++ program that simulates a simple bank transaction system. The user can deposit or withdraw money. If the user tries to withdraw more money than the current balance, the program should throw an exception. Use exception handling to prevent the withdrawal when the balance is insufficient, and display a message indicating the error. The program should also throw exceptions for invalid inputs like negative deposit amounts and handle them accordingly.

THANK

YOU