UNIT IV - TEMPLATES AND EXCEPTION HANDLING

- **Topics to be discussed,**
  - Function Template and Class Template
  - Namespaces
  - Casting
  - **Exception Handling**

Exception Handling

- **What is Exception?**
  - The errors that occur at run-time are known as exceptions.
  - An exception is an unexpected problem that arises during the execution of a program our program terminates suddenly with some errors/issues.
  - **Types of C++ Exception**
    - There are two types of exceptions in C++
      - **Synchronous**
      - **Asynchronous**

Exception Handling – Cont'd

- **Synchronous:**
  - Exceptions that happen when something goes wrong because of a mistake in the input data or when the program is not equipped to handle the current type of data it's working with
  - For example, they occur due to different conditions such as division by zero, accessing an element out of bounds of an array, unable to open a file, running out of memory and many more.
- **Asynchronous**:
  - Exceptions that are beyond the program's control, such as disc failure, keyboard interrupts, etc.

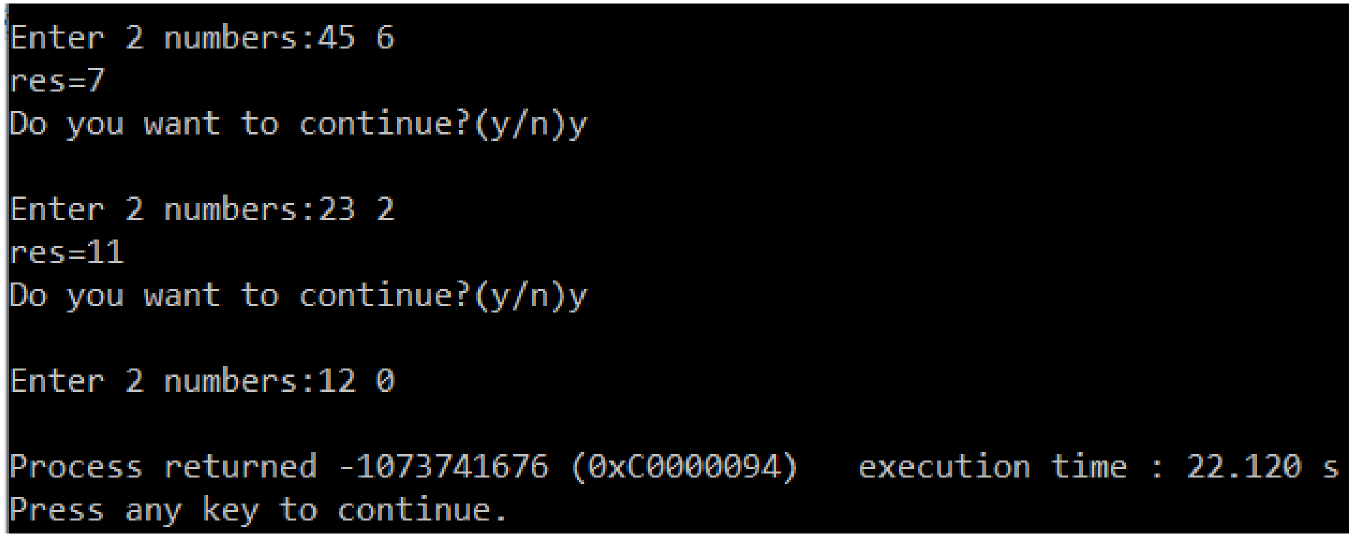
Exception Handling – Cont'd

- **Exception Handling in C++ is a process to handle runtime errors.**
- If we don't handle the exception, it prints exception message and terminates the program.
- **The main objective of exception handling is to provide a way to detect and report the exception** condition so that necessary action can be taken without troubling the user.
- We perform exception handling so the normal flow of the application can be maintained even after runtime errors.
- In C++, exception handling is designed to handle only synchronized exceptions.
- In C++, exception is an event or object which is thrown at runtime.
- All exceptions are derived from **std::exception class**.

Exception - Example

```
#include<iostream>
using namespace std;
int main()
{
   int n1,n2;
   float res;
   char ch;
   while(true)
   {
      cout<<"\nEnter 2 numbers:";
      cin>>n1>>n2;
      res=n1/n2;
      cout<<"res="<<res;
      cout<<"\nDo you want to continue?(y/n)";
      cin>>ch;
      if(ch!='y')
         break;
   }
}
```

Output:



```
Enter 2 numbers:45 6
res=7
Do you want to continue?(y/n)y

Enter 2 numbers:23 2
res=11
Do you want to continue?(y/n)y

Enter 2 numbers:12 0

Process returned -1073741676 (0xC0000094)   execution time : 22.120 s
Press any key to continue.
```

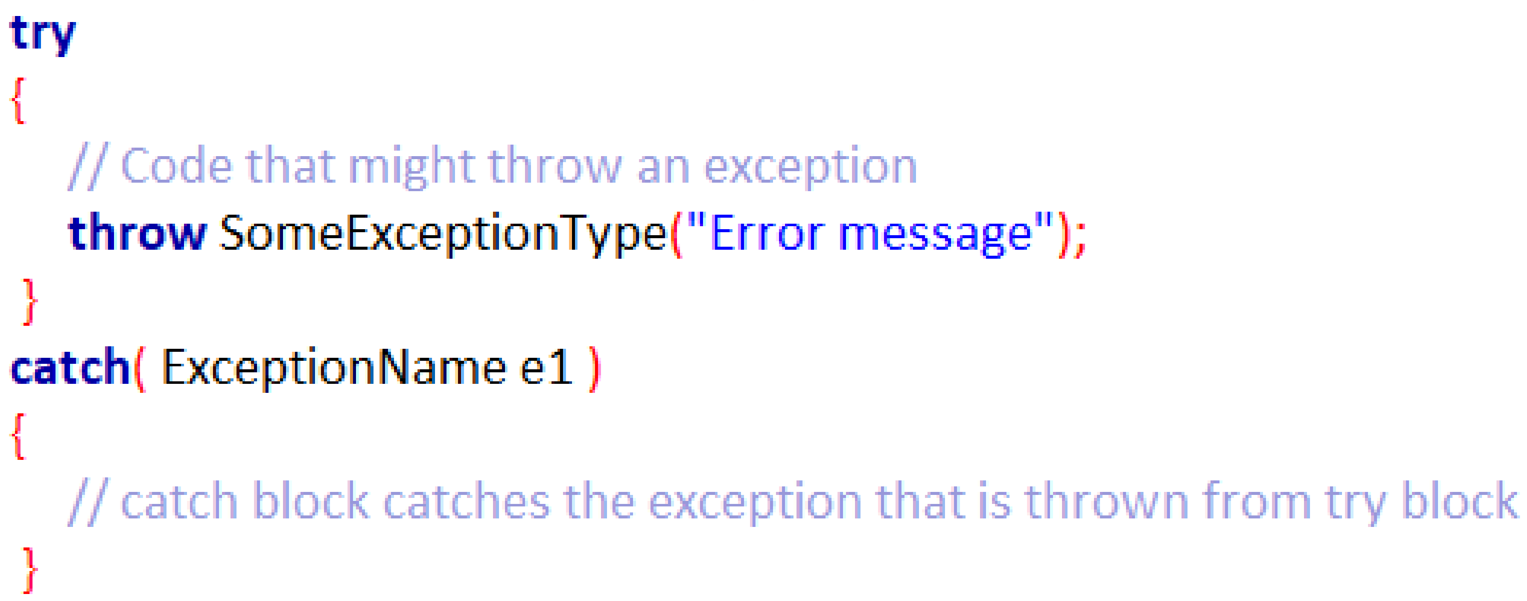
Exception Handling – Cont'd

- **Exception Handling Mechanism**
  - Whenever an exception occurs in a C++ program, the **portion the program that detects the exception can inform that exception has occurred by throwing it**
  - On throwing an exception, the program control immediately stops the step by step execution of the code and jumps to the separate block of code known as an exception handler.
  - **The exception handler catches the exception and processes it without troubling the user**.
  - However, if there is no exception handler, the program terminates abnormally.
  - **C++ provides three constructs try, throw and catch, for implementing exception handling.**

Exception Handling – Cont'd

C++ try and catch

Syntax:



```
try
{
    // Code that might throw an exception
    throw SomeExceptionType("Error message");
 }
catch( ExceptionName e1 )
{
    // catch block catches the exception that is thrown from try block
 }
```

- **try**
  - The try keyword represents a block of code that may throw an exception placed inside the try block.
  - It's followed by one or more catch blocks.
  - If an exception occurs, try block throws that exception.

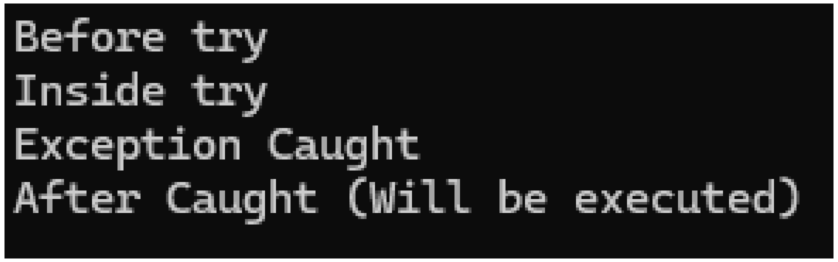
Exception Handling – Cont'd

- **catch**
  - The catch statement represents a block of code that is executed when a particular exception is thrown from the try block.
  - The code to handle the exception is written inside the catch block.
- **throw**
  - An exception in C++ can be thrown using the throw keyword.
  - When a program encounters a throw statement, then it immediately terminates the current function and starts finding a matching catch block to handle the thrown exception.

Exception Handling – Example 1

```cpp
#include <iostream>
using namespace std;
int main()
{
    int x = -1;
    cout << "Before try \n";
    try
    {
        cout << "Inside try \n";
        if (x < 0)
        {
            throw x;
            cout << "After throw (Never executed) \n";
        }
    }
    catch (int x)
    {
        cout << "Exception Caught \n";
    }
    cout << "After Caught (Will be executed) \n";
    return 0;
}
```

Output:

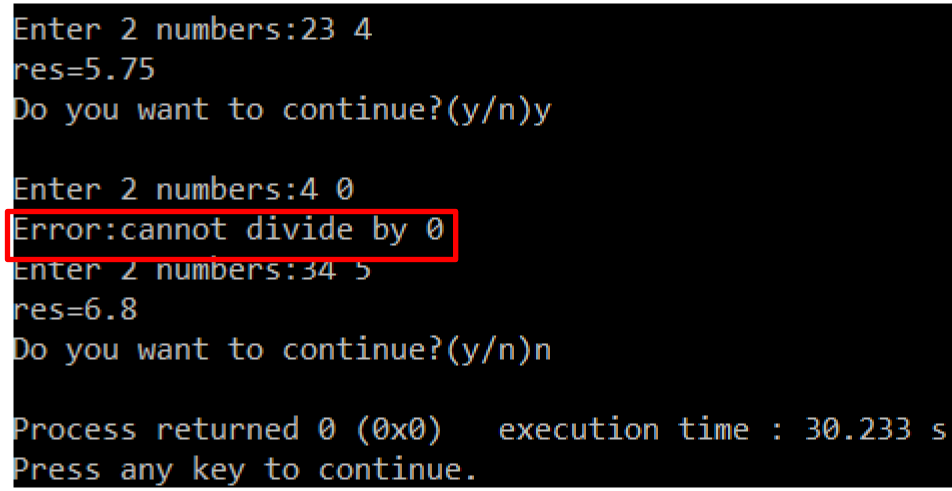


```
Before try
Inside try
Exception Caught
After Caught (Will be executed)
```

Exception Handling – Example 2

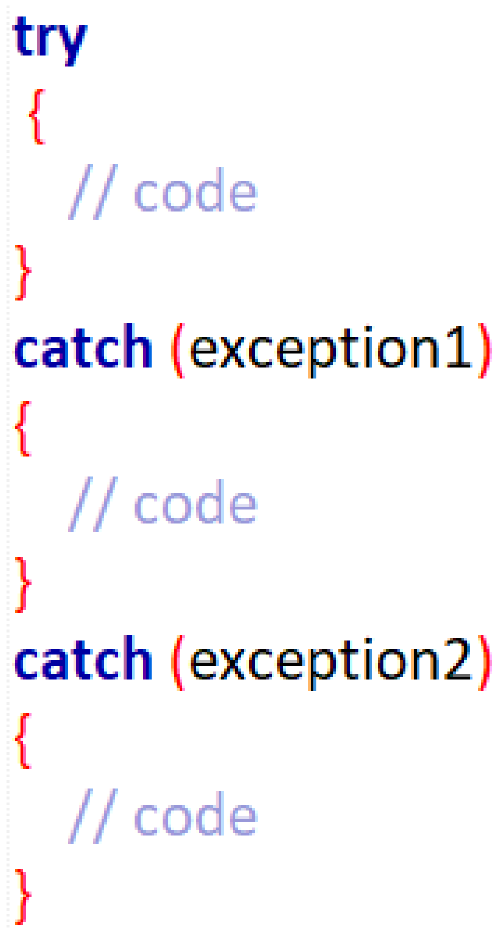
```
#include<iostream>
using namespace std;
int main()
{
   int n1,n2;
   float res;
   char ch;
   while(true)
   {
      cout<<"\nEnter 2 numbers:";
      cin>>n1>>n2;
      try
      {
         if (n2==0)
            throw 0;
         res=static_cast<float>(n1)/n2;
         cout<<"res="<<res;
         cout<<"\nDo you want to continue?(y/n)";
         cin>>ch;
         if(ch!='y')
            break;
      }
      catch(int exp)
      {
         cout<<"Error:cannot divide by "<<exp;
      }
   }
}
```

Output:



```
Enter 2 numbers:23 4
res=5.75
Do you want to continue?(y/n)y

Enter 2 numbers:4 0
Error:cannot divide by 0
Enter 2 numbers:34 5
res=6.8
Do you want to continue?(y/n)n

Process returned 0 (0x0)   execution time : 30.233 s
Press any key to continue.
```

Exception Handling – Cont’d



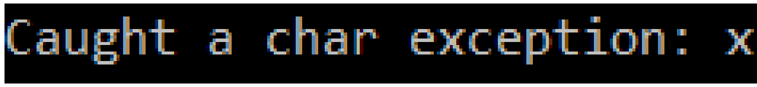
```
try
 {
   // code
}
catch (exception1)
{
   // code
}
catch (exception2)
{
   // code
}
```

- **Multiple catch Statements**
  - In C++, we can use multiple catch statements for different kinds of exceptions that can result from a single block of code.

Multiple catch Statements – Example 1

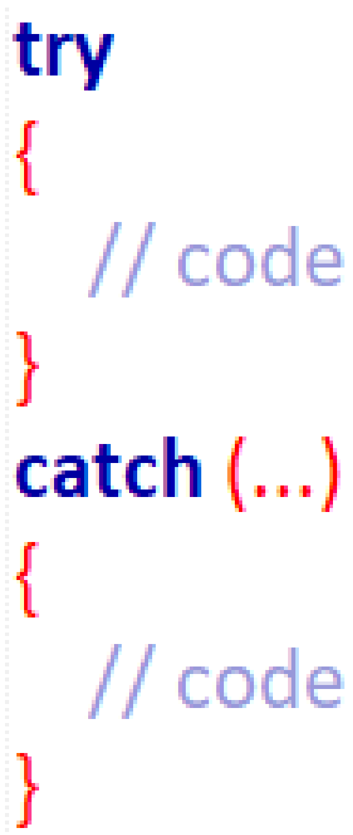
```cpp
#include <stdexcept>
using namespace std;
int x = 5;
int main()
{
    try
    {
        if (x == 0)
            throw x;
        else if (x > 0)
            throw 'x';
        else
            throw "x is negative";
    }
    catch (int i)
    {
        cout << "Caught an int exception: " << i << endl;
    }
    catch (char c)
    {
        cout << "Caught a char exception: " << c << endl;
    }
    catch (char* str)
    {
        cout << "Caught a string exception: " << str << endl;
    }
}
```

Output:



```
Caught a char exception: x
```

Exception Handling – Cont'd



```
try
{
   // code
}
catch (...)
{
   // code
}
```

- **Catching All Types of Exceptions**
  - In exception handling, it is important that we know the types of exceptions that can occur due to the code in our try statement.
  - This is so that we can use the appropriate catch parameters.
  - Otherwise, the try...catch statements might not work properly.
  - If we do not know the types of exceptions that can occur in our try block, then we can use the ellipsis symbol ... as our catch parameter.

Exception Handling – Cont’d

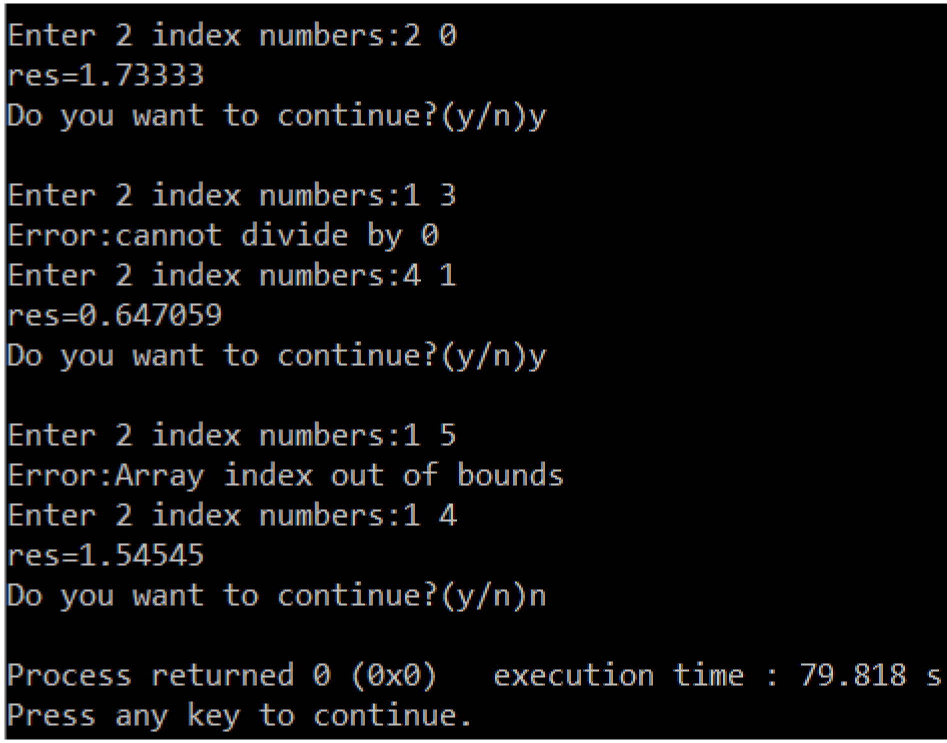
```
try
{
   // code
}
catch (exception1)
 {
   // code
}
catch (exception2)
 {
   // code
}
catch (...)
{
   // code
}
```

- Our program catches exception1 if that exception occurs.
- If not, it will catch exception2 if it occurs.
- If there is an error that is neither exception1 nor exception2, then the code inside of catch (...) {} is executed.
- **Note:**
- catch (...) {} should always be the final block in our try...catch statement.
- This is because this block catches all possible exceptions and acts as the default catch block
- It is not compulsory to include the default catch block in our code.

Multiple catch Statements - Example

```
#include<iostream>
using namespace std;
int main()
{
   int ind1,ind2;
   int arr[5]={45,34,78,0,22};
   float res;
   char ch;
   while(true)
   {
      cout<<"\nEnter 2 index numbers:";
      cin>>ind1>>ind2;
      try
      {
         if (ind1>4 || ind2>4)
            throw "Error:Array index out of bounds";
         if(arr[ind2]==0)
            throw 0;
        res=static_cast<float>(arr[ind1])/arr[ind2];
        cout<<"res="<<res;
        cout<<"\nDo you want to continue?(y/n)";
        cin>>ch;
         if(ch!='y')
            break;
     }
     catch(const char* emsg)
     {
        cout<<emsg;
     }
     catch(int exp)
     {
        cout<<"Error:cannot divide by "<<exp;
     }
     catch (...)
     {
        cout << "Unexpected exception!" << endl;
     }
  }
}
```

Output:



```
Enter 2 index numbers:2 0
res=1.73333
Do you want to continue?(y/n)y

Enter 2 index numbers:1 3
Error:cannot divide by 0
Enter 2 index numbers:4 1
res=0.647059
Do you want to continue?(y/n)y

Enter 2 index numbers:1 5
Error:Array index out of bounds
Enter 2 index numbers:1 4
res=1.54545
Do you want to continue?(y/n)n

Process returned 0 (0x0)   execution time : 79.818 s
Press any key to continue.
```

Exception Handling – Cont’d

Throwing Exceptions from C++ constructors

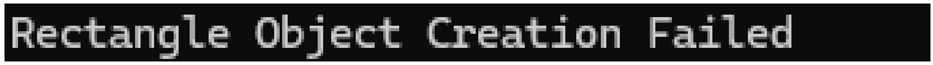
- An exception should be thrown from a C++ constructor whenever an object cannot be properly constructed or initialized.
- Since there is no way to recover from failed object construction, an exception should be thrown in such cases.
- Since C++ constructors do not have a return type, it is not possible to use return codes.
- Therefore, the best practice is for constructors to throw an exception to signal failure.
- The throw statement can be used to throw a C++ exception and exit the constructor code.

Throwing Exceptions from C++ constructors - Example

```
#include <iostream>
using namespace std;
class Rectangle
{
  private:
    int length;
    int breadth;
  public:
    Rectangle(int l, int b)
    {
      if (l < 0 || b < 0)
      {
        throw 1;
      }
      else
      {
        length = l;
        breadth = b;
      }
    }
    void Display()
    {
      cout << "Length: " << length << " Breadth: " << breadth;
    }
};
```

```
int main()
{
  try
  {
    Rectangle r1(10, -5);
    r1.Display();
  }
  catch (int num)
  {
    cout << "Rectangle Object Creation Failed";
  }
}
```

Output:



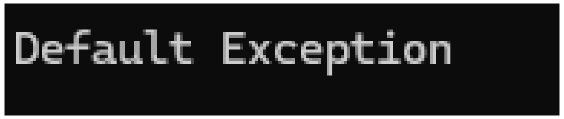
```
Rectangle Object Creation Failed
```

Exception Handling – Cont'd

- Implicit type conversion doesn't happen for primitive types.

```
#include <iostream>
using namespace std;
int main()
{
   try
   {
      throw 'a';
   }
   catch (int x)
   {
      cout << "Caught " << x;
   }
   catch (...)
   {
      cout << "Default Exception\n";
   }
   return 0;
}
```

Output:



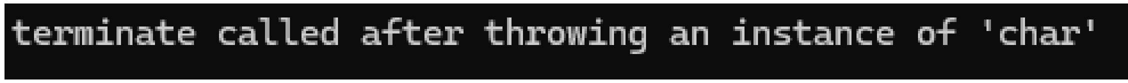
```
Default Exception
```

Exception Handling – Cont’d

- If an exception is thrown and not caught anywhere, the program terminates abnormally.

```
#include <iostream>
using namespace std;
int main()
{
   try
   {
      throw 'a';
   }
   catch (int x)
   {
      cout << "Exception Caught ";
   }
   return 0;
}
```

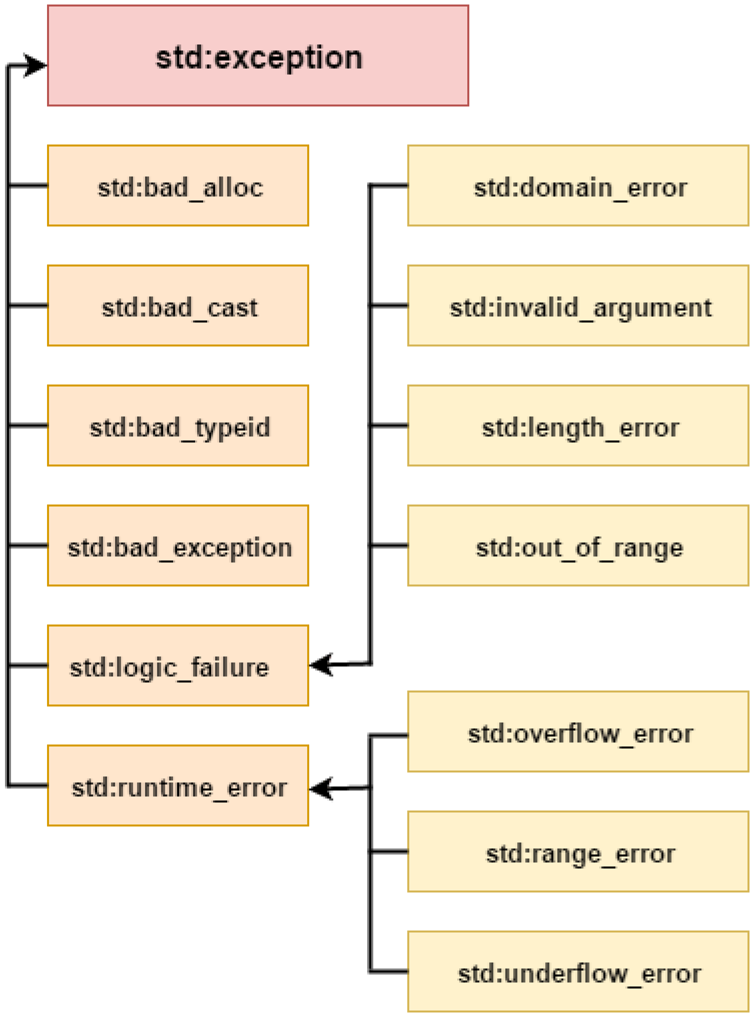
Output:



```
terminate called after throwing an instance of 'char'
```

Exception Handling – Cont'd

C++ Standard Exception



- In C++ standard exceptions are defined in <exception> class that we can use inside our programs.

Exception Handling – Cont’d

C++ Standard Exceptions

- **std::exception** - Parent class of all the standard C++ exceptions.
- **logic_error** - Exception happens in the internal logical of a program.
  - **domain_error** - Exception due to use of invalid domain.
  - **invalid argument** - Exception due to invalid argument.
  - **out_of_range** - Exception due to out of range i.e. size requirement exceeds allocation.
  - **length_error** - Exception due to length error.

Exception Handling – Cont’d

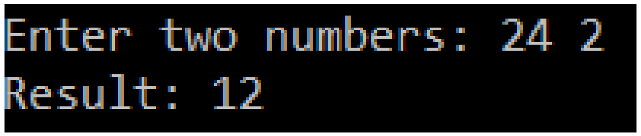
C++ Standard Exceptions

- **runtime_error** - Exception happens during runtime.
  - **range_error** - Exception due to range errors in internal computations.
  - **overflow_error** - Exception due to arithmetic overflow errors.
  - **underflow_error** - Exception due to arithmetic underflow errors
- **bad_alloc** - Exception happens when memory allocation with new() fails.
- **bad_cast** - Exception happens when dynamic cast fails.
- **bad_exception** - Exception is specially designed to be listed in the dynamic-exception-specifier.
- **bad_typeid** - Exception thrown by typeid.

Standard Exception Example 1

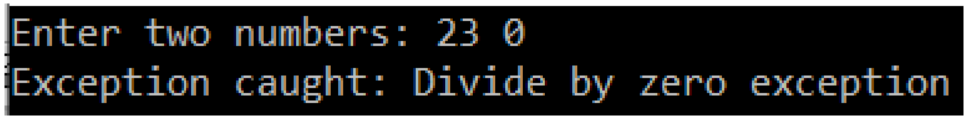
```cpp
#include <iostream>
using namespace std;
int main()
{
    try
    {
        int num1, num2;
        cout << "Enter two numbers: ";
        cin >> num1 >> num2;
        if (num2 == 0)
        {
            throw runtime_error("Divide by zero exception");
        }
        int result = num1 / num2;
        cout << "Result: " << result <<endl;
    }
    catch (const exception& e)
    {
        cout << "Exception caught: " << e.what() << std::endl;
    }
    return 0;
}
```

Output 1:



```
Enter two numbers: 24 2
Result: 12
```

Output 2:

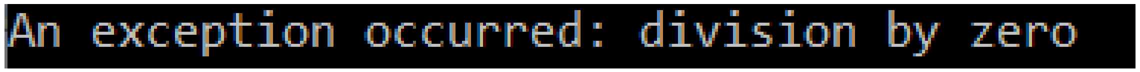


```
Enter two numbers: 23 0
Exception caught: Divide by zero exception
```

Standard Exception Example 2

```cpp
#include<iostream>
#include <stdexcept>
using namespace std;
int divide(int a, int b)
{
    if (b == 0)
    {
        throw invalid_argument("division by zero");
    }
    return a / b;
}
int main()
{
    try
    {
        int result = divide(1, 0);
        cout << result << endl;
    }
    catch (const invalid_argument& e)
    {
        cout << "An exception occurred: " << e.what() << endl;
    }
    return 0;
}
```

Output:



```
An exception occurred: division by zero
```

Exception Handling – Cont’d
re-throwing an Exception

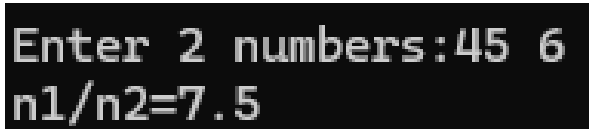
- Re-throwing an exception in C++ involves catching an exception within a try block and **instead of dealing with it locally, throwing it again to be caught by an outer catch block.**
- By doing this, we preserve the type and details of the exception ensuring that it can be handled at the appropriate level within our program.
- This approach becomes particularly valuable when managing exceptions at multiple levels or when additional actions need to be performed before resolving the exception.

re-throwing an Exception - Example

```cpp
#include <iostream>
using namespace std;
void division(int n1,int n2)
{
   try
   {
      if(n2==0)
         throw n2;
      else
         cout<<"n1/n2="<<(float)n1/n2;
   }
   catch(int)
   {
      cout<<"\nCaught an exception as first throwing";
      throw;
   }
}
```

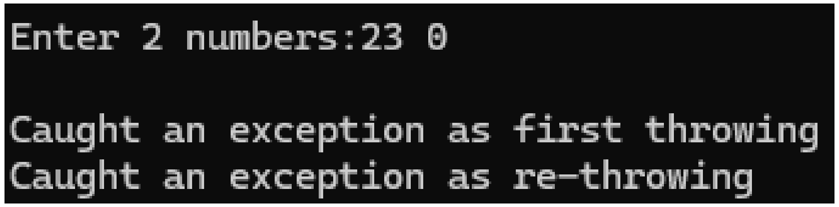
```cpp
int main()
{
   int a,b;
   cout<<"\nEnter 2 numbers:";
   cin>>a>>b;
   try
   {
      division(a,b);
   }
   catch(int)
   {
      cout<<"\nCaught an exception as re-throwing";
   }
   return 0;
}
```

Output 1:



```
Enter 2 numbers:45 6
n1/n2=7.5
```

Output 2:



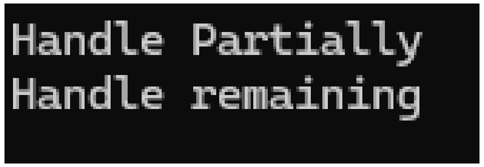
```
Enter 2 numbers:23 0

Caught an exception as first throwing
Caught an exception as re-throwing
```

Exception Handling – Cont’d

- In C++, **try/catch blocks can be nested**.
- Also, an exception can be re-thrown using “throw; “.

```
#include <iostream>
using namespace std;
int main()
{
    // nesting of try/catch
   try {
            try
            {
               throw 20;
            }
            catch (int n)
            {
               cout << "Handle Partially\n";
               throw; // Re-throwing an exception
            }
      }
   catch (int n)
   {
      cout << "Handle remaining\n ";
   }
   return 0;
}
```

Output:



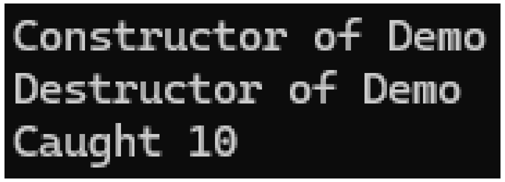
```
Handle Partially
Handle remaining
```

Exception Handling – Cont'd

- When an exception is thrown, all objects created inside the enclosing try block are destroyed before the control is transferred to the catch block.

```
#include <iostream>
using namespace std;
class Demo
{
public:
      Demo()
      {
         cout << "Constructor of Demo " << endl;
      }
      ~Demo()
      {
         cout << "Destructor of Demo " << endl;
      }
};
int main()
{
   try
   {
      Demo obj;
       throw 10;
   }
   catch (int i)
   {
        cout << "Caught " << i << endl;
   }
}
```

Output:



```
Constructor of Demo
Destructor of Demo
Caught 10
```

Exception Handling – Cont’d
User-Defined Exceptions

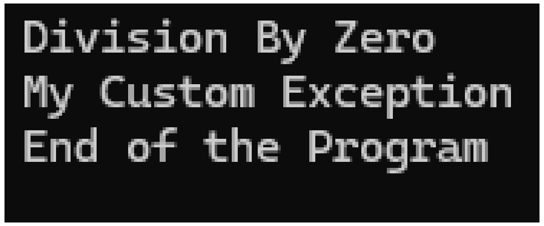
- The C++ **std::exception class allows us to define objects that can be thrown as exceptions**.
- This class has been defined in the <exception> header.
- The class provides us with a virtual member function named what.
- This function returns a null-terminated character sequence of type char *.
- We can overwrite it in derived classes to have an exception description.

User-Defined Exceptions - Example

```cpp
#include<iostream>
using namespace std;
#include <exception>
class MyException:public exception
{
   public:
      char *what()
      {
         return "My Custom Exception";
      }
};
int Division(int a, int b)
{
   if (b == 0)
      throw MyException ();
   return a / b;
}
```

```cpp
int main()
{
   int x = 10, y = 0, z;
   try
   {
      z = Division (x, y);
      cout << z << endl;
   }
   catch (MyException ME)
   {
      cout << "Division By Zero" << endl;
      cout << ME.what () << endl;;
   }
   cout << "End of the Program" << endl;
}
```

Output:

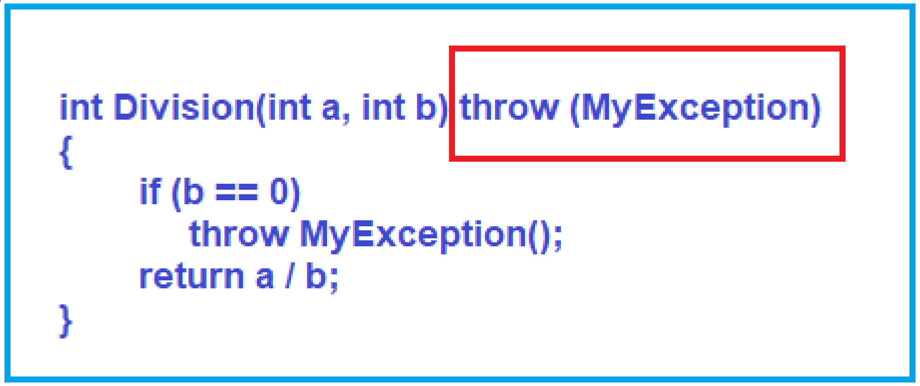


```
Division By Zero
My Custom Exception
End of the Program
```

Exception Handling – Cont'd

- **How to make the function throws something in C++?**
  - when a function is throwing, we can declare that this function throws something.

For example,

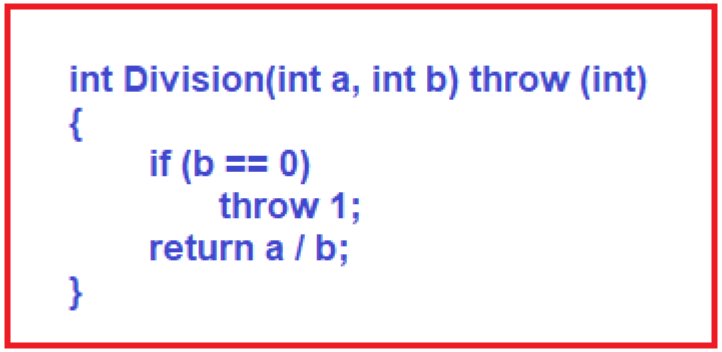


```
int Division(int a, int b) throw (MyException)
{
        if (b == 0)
            throw MyException();
        return a / b;
}
```

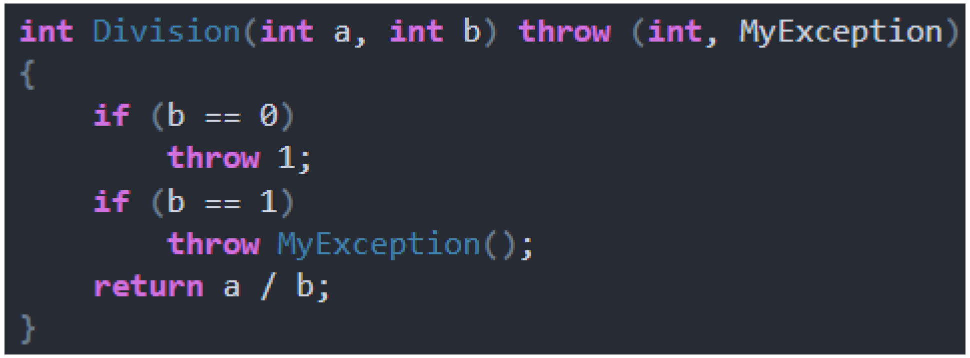
- This Division function declares that it throws some exception i.e. MyException.
- This is optional in C++.
- Whether we want to write or not is up to us.

Exception Handling – Cont’d

- So, whatever the type of value we are throwing, we can mention that in the brackets
- And if there are more values then we can mention them with commas



```
int Division(int a, int b) throw (int)
{
        if (b == 0)
              throw 1;
        return a / b;
}
```



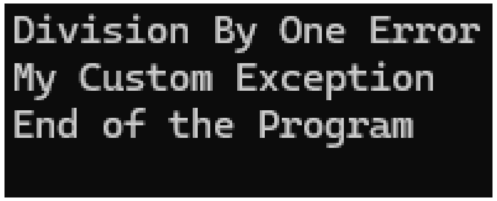
```
int Division(int a, int b) throw (int, MyException)
{
    if (b == 0)
        throw 1;
    if (b == 1)
        throw MyException();
    return a / b;
}
```

function throws something - Example

```cpp
#include<iostream>
using namespace std;
#include <exception>
class MyException:public exception
{
   public:
   char * what()
   {
      return "My Custom Exception";
   }
};
int Division(int a, int b) throw (int, MyException)
{
   if (b == 0)
      throw 1;
   if (b == 1)
      throw MyException();
   return a / b;
}
```

```cpp
int main()
{
   int x = 10, y = 1, z;
   try
   {
      z = Division (x, y);
      cout << z << endl;
   }
   catch (int x)
   {
      cout << "Division By Zero Error" << endl;
   }
   catch (MyException ME)
   {
      cout << "Division By One Error" << endl;
      cout << ME.what () << endl;
   }
   cout << "End of the Program" << endl;
}
```

Output:



```
Division By One Error
My Custom Exception
End of the Program
```