Exception Handling

Errors that occur at program runtime can seriously interrupt the normal flow of a program. Some common causes of errors are

- Division by 0, or values that are too large or small for a type
- No memory available for dynamic allocation.
- Error on file access, for example , file not found.
- Attempt to access an invalid address in main memory
- Invalid user input

Traditional Error Handling

- Traditional structured programming languages use normal syntax to handle errors:
    - errors in function calls are indicated by special return values.
    - global error variables or flags are set when errors occur, and then checked again later.
- If a function uses its return value to indicate errors, the return value must be examined whenever the function is called, even if no error has occurred.

```
if( func()> 0 )
  // Return value positive => o.k.
else
  // Treat errors
```

- Error variables and flags must also be checked after *every* corresponding action.
- Need to continually check for errors while a program is executing, if not, the consequences may be fatal.

Support for Error Handling in C

- C Language does not provide any specific feature for error handling. So, developers use normal programming features to handle errors.
- C standard Library provides a collection of headers that can be used for handling errors in different contexts.
- Language Feature
  - Return value and parameters
  - Local goto
- Standard Library Support
  - Global variables(<errno.h>)
  - Abnormal termination (<stdlib.h>)
  - Condiitional Termination (<assert.h>)
  - Non-Local goto (<setjmp.h>)
  - Signals (<signal.h>)

Error Handling in C using errno.h

```
#include<errno.h>
#include<math.h>
#include<stdio.h>
int main(){
    double x,y,result;
    scanf("%lf%lf",&x,&y);
    errno=0;
    result=pow(x,y);
    if(errno == EDOM)  // if base is –ve and expo is not integer
        printf("Domain error on x/y pair\n");
    else{
        if(errno == ERANGE) // if range of base is more than the double max
            printf("range error in result\n");
        else
            printf("x to the y=%d\n",(int) result);
    }
    return 0;
}
```

Support for Error Handling in C++

- C++ introduces a new approach to error handling, known as Exception Handling.
- Exception handling separates the detection and handling of exceptional flow from the normal flow of the program.
- The basic idea is that errors occurring in one particular part of the program are reported to another part of the program, known as the *calling environment*. The calling environment performs central error handling.
- An application program no longer needs to continually check for errors, because in case of an error, control is automatically transferred to the calling environment.
- When reporting an error, specific information on the error cause can be added. This information is evaluated by the error-handling routines in the calling environment.

Fundamentals of Exception Handling in C++

- Exceptions are conditions that arise infrequently and unexpectedly at run-time that might crash the entire system or application.

- Types of Exceptions
  - ❑ Asynchronous
    - ✓ Exceptions that come Unexpectedly
    - ✓ Example- an Interrupt in a program
    - ✓ Takes control away from the Executing Thread context to a context that is different from that which caused the Exception
  - ❑ Synchronous
    - ✓ Planned Exceptions
    - ✓ Handled in an organized manner
    - ✓ Example- Exception implemented using throw statement.

- Features provided by C++ for Exception handling

  - ❖ try block - which is used to guard the code that might raise an exception

  - ❖ throw expression – generate an exception object and throw the object to handler

  - ❖ catch block – it is an exception handler, which handles the exception (i.e., resolves) or re-throw

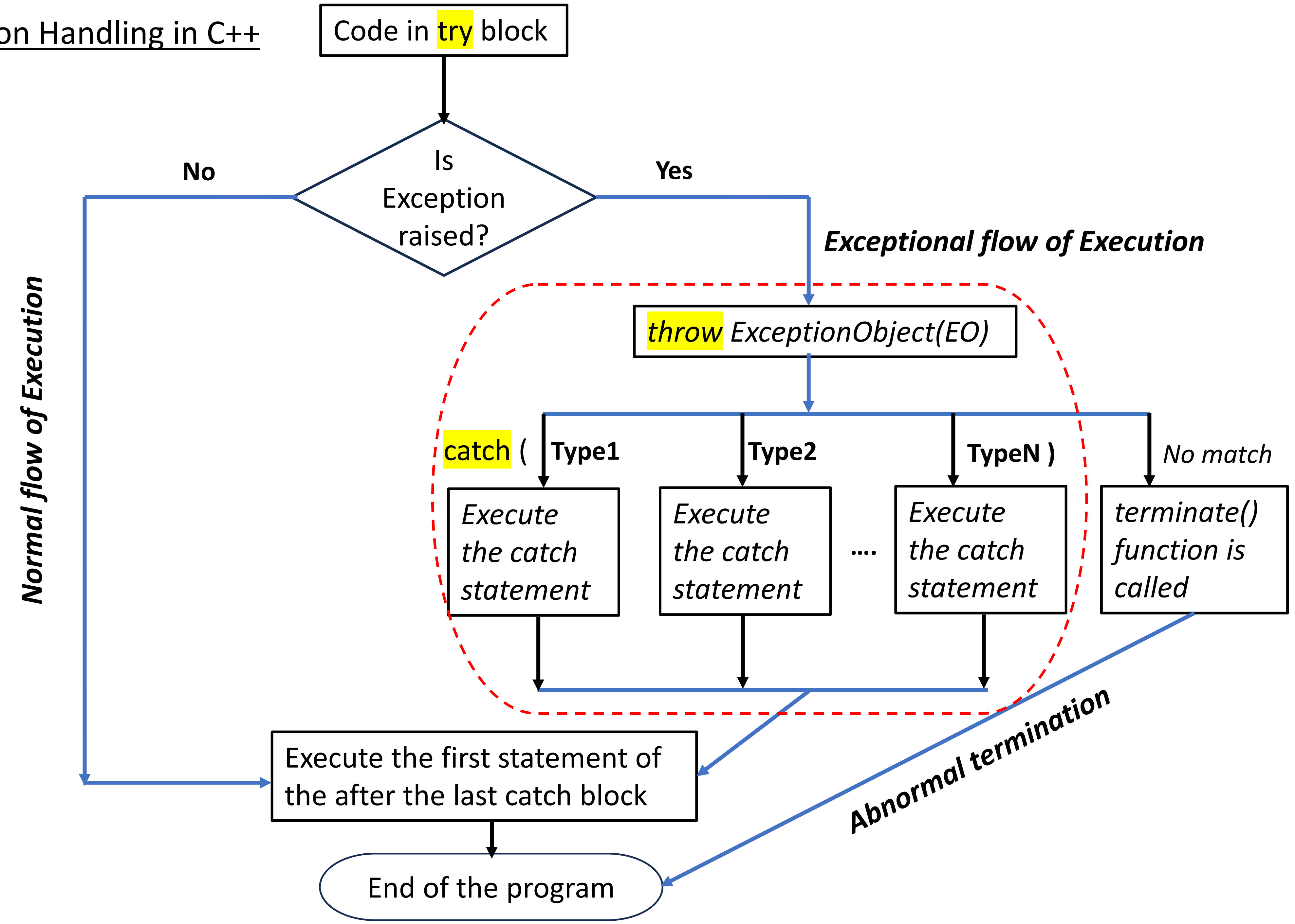
Guidelines for Exception Handling

A *try block* contains the program code in which errors can occur and exceptions can be thrown. Normally, a `try` block will consist of a group of functions that can produce similar errors.

Each *catch block* defines an exception handler, where the *exception declaration*, which is enclosed in parentheses, defines the type of exceptions the handler can catch. The `catch` blocks immediately follow the `try` block. A minimum of one `catch` block is required.

```
try {
    statement 1;
    statement 2;
    throw excj;
    ….
}
catch( Type1 exc1){ // Type1 exceptions are handled here.}
catch( Type2 exc2){ // Type2 exceptions are handled here.}
…
catch(Typen excn){ // Typen exceptions are handled here.}
catch( ... ){
   // All other exceptions are handled here.
}
```

Flow chart of Exception Handling in C++

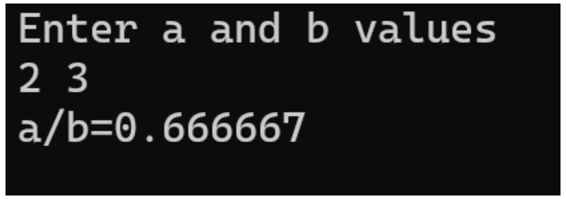

Code in try block
Is Exception raised?
No
Yes
Exceptional flow of Execution
Normal flow of Execution
throw ExceptionObject(EO)
catch (Type1
Type2
TypeN)
No match
Execute the catch statement
Execute the catch statement
....
Execute the catch statement
terminate() function is called
Execute the first statement of the after the last catch block
Abnormal termination
End of the program

Example: Divide-by-Zero Exception

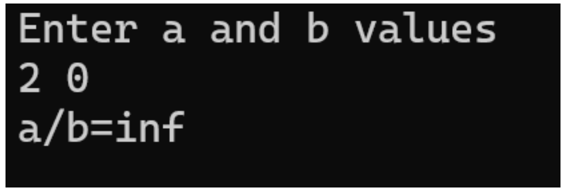
without Exception handling

```
#include<iostream>
using namespace std;
int main(){
    double a,b,result;
    cout<<"Enter a and b values"<<endl;
    cin>>a>>b;
    result=a/b;
    cout<<"a/b="<<result<<endl;
    return 0;
}
```

Output:



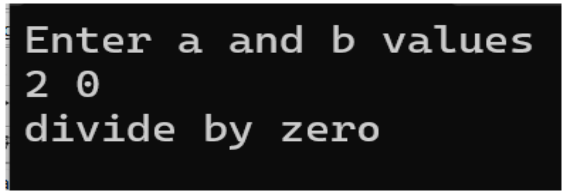
```
Enter a and b values
2 3
a/b=0.666667
```



```
Enter a and b values
2 0
a/b=inf
```

with Exception handling

```
#include<iostream>
using namespace std;
int main(){
    double a,b,result;
    cout<<"Enter a and b values"<<endl;
    cin>>a>>b;
    try{
      if(b==0)
         throw (string)"divide by zero";
      result=a/b;
      cout<<"a/b="<<result<<endl;
    }
catch(string& s){
     cout<<s<<endl;
    }
    return 0;
}
```

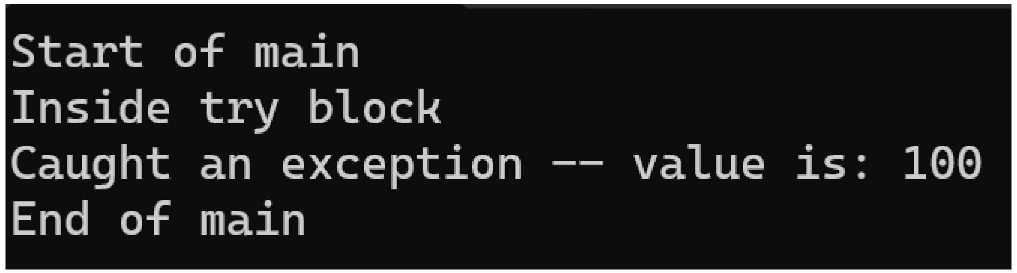


```
Enter a and b values
2 0
divide by zero
```

Example 1

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Start of main"<<endl;
    try {                  // start a try block
        cout << "Inside try block"<<endl;
        throw 100;  // throw an error
        cout << "This will not execute"<<endl;
    }
    catch (int i) {  // catch an error
        cout << "Caught an exception -- value is:  "<<i<<endl;
    }
    cout << "End of main"<<endl;
    return 0;
}
```

Output:



```
Start of main
Inside try block
Caught an exception -- value is: 100
End of main
```

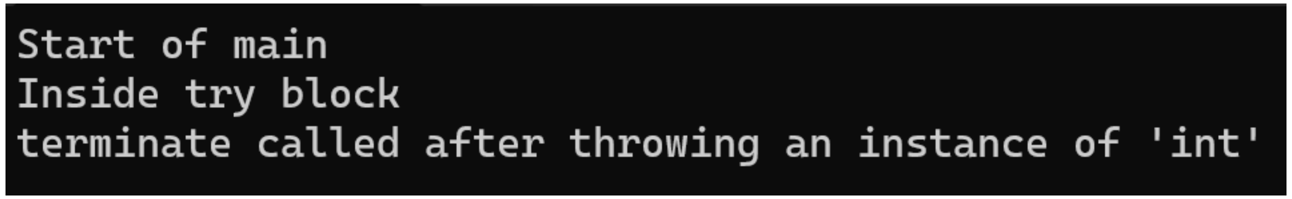
Example 2

*If the catch block can not match the type of argument ,then abnormal termination is happened by calling **terminate()** function.*

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Start of main"<<endl;
    try {     // start a try block
        cout << "Inside try block"<<endl;
        throw 100; // throw an error
        cout << "This will not execute"<<endl;
    }
    catch (double i) {  // won't work for an int exception
        cout << "Caught an exception -- value is: "<<i<<endl;
    }
    cout << "End"<<endl;
    return 0;
}
```

Catch doesn't match the exception thrown

Output:



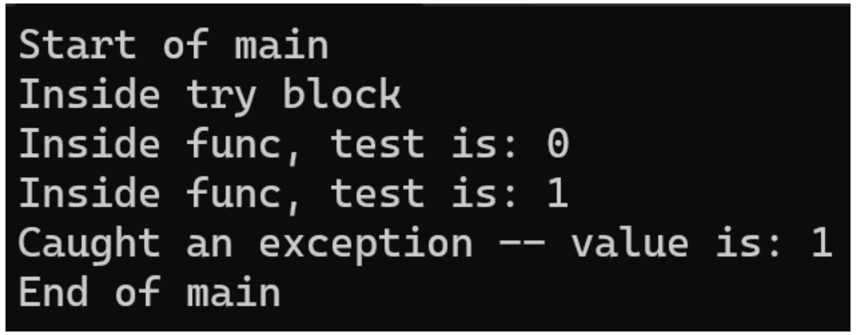
```
Start of main
Inside try block
terminate called after throwing an instance of 'int'
```

Example 3

An exception can be thrown from outside the ***try*** *block as long as it is thrown by a function that is called from within* ***try*** *block.*

```
void func(int test)
{
   cout << "Inside func, test is: " << test << "\n";
   if(test)
     throw test;
}
int main()
{
   cout << "Start of main"<<endl;
   try { // start a try block
      cout << "Inside try block\n";
      func(0);
      func(1);
      func(2); // it is not executed
   }
   catch (int i) { // catch an error
      cout << "Caught an exception -- value is: "<<i<<endl;
   }
   cout << "End of main"<<endl;
   return 0;
}
```

Output:



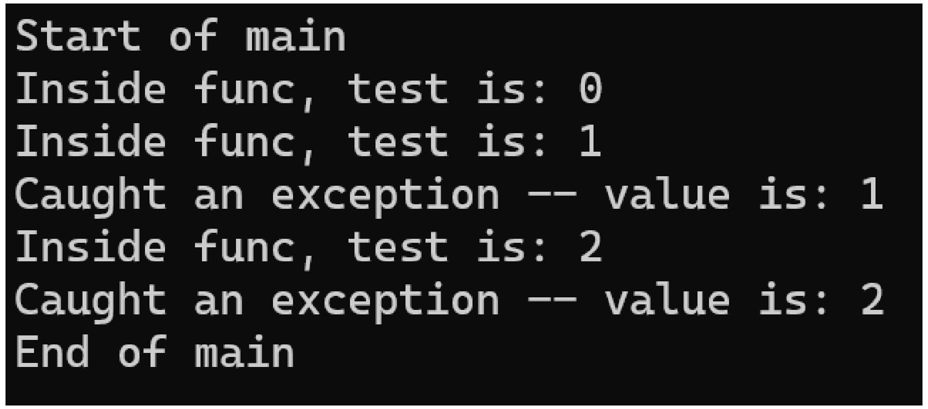
```
Start of main
Inside try block
Inside func, test is: 0
Inside func, test is: 1
Caught an exception -- value is: 1
End of main
```

Example 4

A **try** block can be localized to a function , then for each time the function is entered, the exception handling relative to that ***function is reset***.

```
// Localize a try/catch to a function.
void func(int test)
{
   try{
         cout << "Inside func, test is: " << test << "\n";
         if(test) throw test;
   }
   catch (int i) {
        cout << "Caught an exception -- value is: "<<i<<endl;
   }
}
int main()
{
   cout << "Start of main"<<endl;
   func(0);
   func(1);
   func(2);
   cout << "End of main"<<endl;
   return 0;
}
```

Output:



```
Start of main
Inside func, test is: 0
Inside func, test is: 1
Caught an exception -- value is: 1
Inside func, test is: 2
Caught an exception -- value is: 2
End of main
```

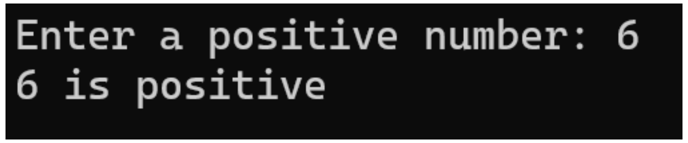
Catching Class Types

- An exception can be of any type, including class types that you create.
- In real-world programs, most exceptions will be class types rather than built-in types.
- Define an exception class and create an object at throw expression

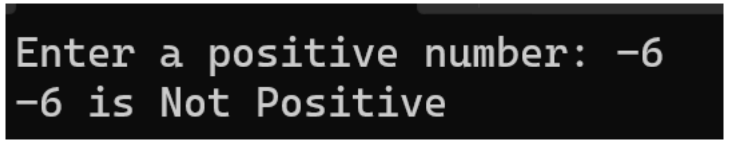
```
class MyException {
   public:
      string emsg;
      int value;
      MyException() { emsg="";value=0; }
      MyException(string s1, int e) {
         emsg=s1;
         value = e;
      }
};
```

```
int main()
{
        int i;
        try {
           cout << "Enter a positive number: ";
           cin >> i;
           if(i<0)
              throw MyException(" is Not Positive", i);
           cout<<i<<" is positive"<<endl;
        }
        catch (MyException e) {  // catch an error
             cout << e.value<< e.emsg << endl;
        }
        return 0;
}
```

Output:



```
Enter a positive number: 6
6 is positive
```



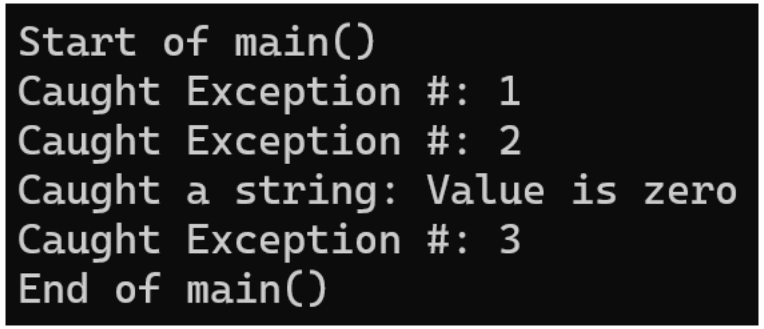
```
Enter a positive number: -6
-6 is Not Positive
```

Multiple catch statements

Example 6

```
void func(int test)
{
   try{
      if(test) throw test;
      else throw "Value is zero";
   }
   catch(int i) {
      cout << "Caught Exception #: " << i <<endl;
   }
   catch(const char *str) {
      cout << "Caught a string: "<<str<<endl;
   }
};
int main()
{
   cout << "Start of main()"<<endl;
   func(1);
   func(2);
   func(0);
   func(3);
   cout << "End of main()"<<endl;
   return 0;
}
```

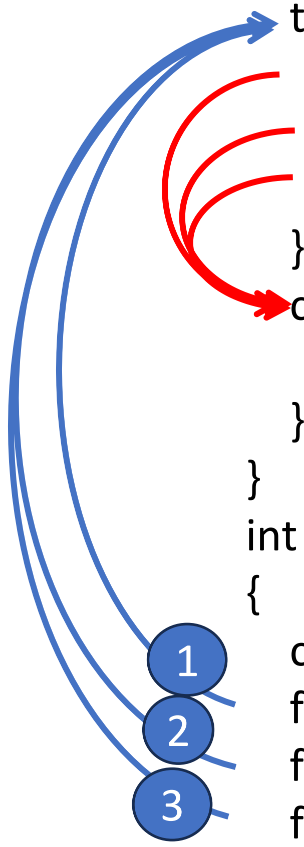
Output:



```
Start of main()
Caught Exception #: 1
Caught Exception #: 2
Caught a string: Value is zero
Caught Exception #: 3
End of main()
```

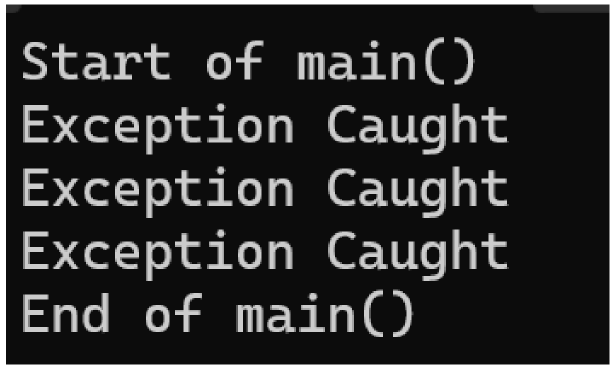
Catching all Exception using single catch() statement

Example 7



```
void func(int test)
{
   try{
      if(test==0) throw test; // throw int
      if(test==1) throw 'a'; // throw char
      if(test==2) throw 123.23; // throw double
   }
   catch(...) { // catch all exceptions
      cout << "Exception Caught "<<endl;
   }
}
int main()
{
   cout << "Start of main()"<<endl;
   func(0);
   func(1);
   func(2);
   cout << "End of main()"<<endl;
   return 0;
}
```

Output:



```
Start of main()
Exception Caught
Exception Caught
Exception Caught
End of main()
```

Restricting Exceptions or Specifying exceptions

- You can restrict or specify the type of exceptions that a function can throw outside of itself.
- To accomplish these restrictions, you must add a **throw** clause to a function definition.
- The general form of this is shown here:

```
ret-type func-name(arg-list) throw(type-list)
{
        // ...
}
```

```
/ This function can only throw ints, chars, and doubles.
void func(int test) throw(int, char, double)
{
   if(test==0) throw test; // throw int
   if(test==1) throw 'a'; // throw char
   if(test==2) throw 123.23; // throw double
}
```

Example 8

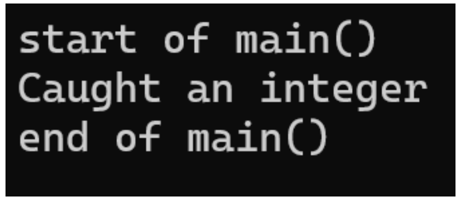
```
// This function can only throw ints, chars, and doubles.
void func(int test) throw(int, char, double)
{
   if(test==0) throw test; // throw int
   if(test==1) throw 'a'; // throw char
   if(test==2) throw 123.23; // throw double
}
```

```
int main()
{
   cout << "start of main()"<<endl;
   try{
      func(0);
   }
   catch(int i) {
      cout << "Caught an integer"<<endl;
   }
   catch(char c) {
      cout << "Caught char"<<endl;
   }
   catch(double d) {
      cout << "Caught double"<<endl;
   }
   cout << "end of main()"<<endl;
   return 0;
}
```

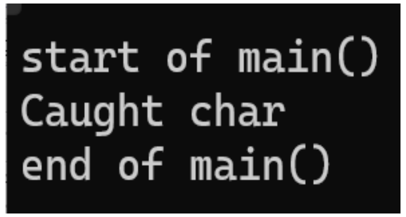
```
try{
   func(0);
}
```

```
try{
   func(1);
}
```

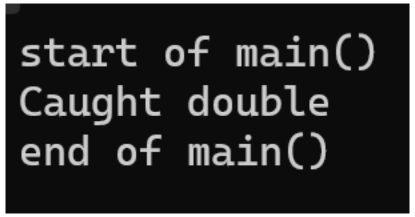
```
try{
   func(2);
}
```



```
start of main()
Caught an integer
end of main()
```



```
start of main()
Caught char
end of main()
```

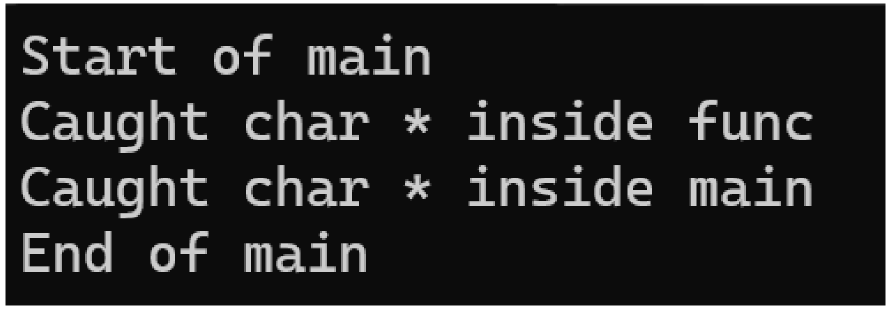


```
start of main()
Caught double
end of main()
```

Rethrowing an Exception

- Rethrowing an Exception can be done by calling throw with no exception.
- Rethrow can be done from only catch block , which can be handled by outer try/catch sequence.

```
#include <iostream>
using namespace std;
void func()
{
   try {
      throw "hello"; // throw a char *
   }
   catch(const char *) { // catch a char *
      cout << "Caught char * inside func\n";
      throw ; // rethrow char * out of function
   }
}
int main()
{
   cout << "Start of main“<<endl;
   try{
      func();
   }
   catch(const char *) {
      cout << "Caught char * inside main\n";
   }
   cout << "End of main”<<endl";
   return 0;
}
```



```
Start of main
Caught char * inside func
Caught char * inside main
End of main
```

Exception stages

- Error Incidence
  - Synchronous (S/W) logical error
  - Asynchronous (H/W) Interrupt(s/w Interrupt)
- Create Object and Raise Exception
  - Create an Exception object which can be of pre-defined data type or user-defined classes.
- Detect Exception
  - Polling – software tests
  - Notification – Control stack adjustments
- Handle Exception
  - Ignore : do not catch
  - Act: catch,handle,and re-throw
  - Own: catch and handle
- Recover from Exception
  - Continue Execution: If handled inside the program
  - Abort Execution : If handled outside the program

Error Handling in C using errno.h

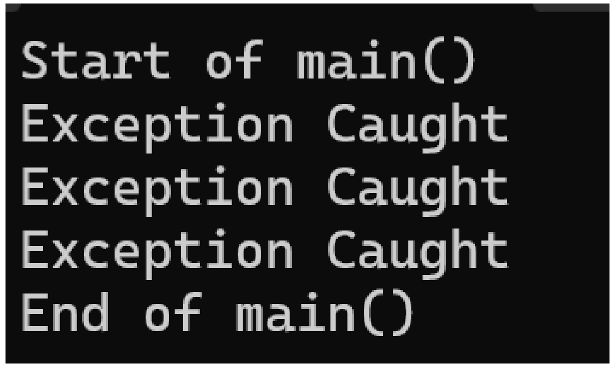
```
#include<errno.h>
#include<math.h>
#include<stdio.h>
int main(){
    double x,y,result;
    scanf("%lf%lf",&x,&y);
    errno=0;

    result=pow(x,y);

    if(errno == EDOM)  // if base is –ve and expo is not integer
        printf("Domain error on x/y pair\n");
    else{
        if(errno == ERANGE)
            printf("range error in result\n");
        else
            printf("x to the y=%d\n",(int) result);
    }
    return 0;
}
```

Catching all Exception using single catch() statement

Example 7

```
void func(int test)
{
  try{
    if(test==0) throw test; // throw int
    if(test==1) throw 'a'; // throw char
    if(test==2) throw 123.23; // throw double
  }
  catch(...) { // catch all exceptions
    cout << "Exception Caught "<<endl;
  }
}
int main()
{
  cout << "Start of main()"<<endl;
  func(0);
  func(1);
  func(2);
  cout << "End of main()"<<endl;
  return 0;
}
```

Output:



```
Start of main()
Exception Caught
Exception Caught
Exception Caught
End of main()
```