Exception Handling

CS(217) Object Oriented Programming

Exception Handling **Type of Errors**

Compile time Errors

- Compile time errors are syntactic errors which occurs during writing of the program.
- Common examples are missing semicolon, missing comma, etc.
- They occurs due to *poor understanding of language*.

Logical Errors

- They occur due to *improper understanding of the program logic*.
- Logical errors cause the unexpected behavior and output of program.

Run time Errors or Exceptions

- They *occurs accidentally* which may result in abnormal termination of the program.
- Common examples are division by zero, opening file to read which does not exist, insufficient memory, violating array bounds, etc.

Exception Handling

- Exception handling is the process to handle the exception *if generated* by the program at runtime.
  - The aim is to write code, which passes exception to a routine.
  - This routine can handle the exception and can take suitable action.
- Exception Handling Steps are:

  Step 1 : Writing try block.

  Step 2 : Throwing an exception.

  Step 3 : Catching and handling the exception thrown.

- C++ provides exception handling mechanism
  - To trap different exceptions in programs.
  - To make programs running smoothly after catching the exception.

Exception Handling **Step 1 : Writing try block.**

- The piece of code in which exception can occur should be written in a try block.
- All variables created in try block are local to that block.
  - They are out of scope when try block ends.
- Exception can occur at any statement in try block.
  - The code following that statement is ignored by system.

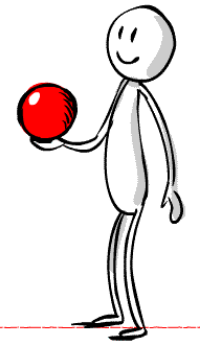
```
try
{
    Statement 1;
    Statement 2;
    Statement 3;
}
```

Exception Handling **Step 2 : Throwing an exception.**

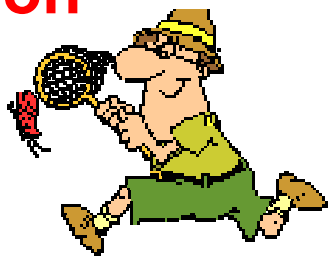
An exception is an object so we can say that an exception object is thrown.

- Whenever an exception is generated in the try block, it is thrown.
- `throw` is reserve word in C++.
- For throwing an exception `throw exception/variable/value;`
- For re-throwing an exception `throw;`

```
try
{
    Statement 1;
    Statement 2;
    Statement 3;
    throw 5; //Could be any data type
}
```



Exception Handling **Step 3 : Catching the exception**



- A try block
  - Must have **at least one matching** catch block.
  - Can have more than one catch blocks for catching different types of exceptions.
- A catch block must have a **try block prior written** which will throw an exception.
- A catch block should have only **one argument**.
- An exception thrown by try block is caught and handled by the catch block.
  - If exception thrown **matches with the argument** in catch block,
    - Then exception will be caught successfully by the catch block.
    - After successful execution of the catch block any code written after the catch block will be executed.
  - If argument **does not match** with the exception thrown,
    - Catch block cannot handle it and this may results in abnormal program termination.

Exception Handling **Step 3 : Catch the exception**

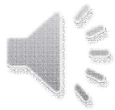
- An exception thrown by try block is caught and handled by the catch block.

```
try{
    Statement 1;
    Statement 2;
    throw exception; //Could be any variable of any data type
} //No code should be written between try and catch blocks
catch (dataType1 identifier){
//Catch takes only single argument, which should match to thrown object data
type
  statements for handling the exception;
}
catch (dataType2 identifier) {
  statements for handling the exception;
}
//Code after catch block
```

Exception Handling **Example**

```
void main(){
      cout << "Start“ << endl;
      try { // start a try block
              cout << "Inside try block\n";
              throw 100; // throw an error
              cout << "This will not execute";
      }
      catch (int i) { // catch an error
              cout << "Caught an exception -- value is: “ << i << endl;
      }
      cout << "End” << endl;
}
```

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

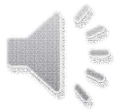



Exception Handling **Example**

```
// Catch argument does not match with thrown value abnormal behavior.
void main(){
      cout << "Start“ << endl;
      try {
            cout << "Inside try block\n";
            throw 100; // throw an error
            cout << "This will not execute";
      }
      catch (double i) { // catch an error
            cout << "Caught an exception -- value is: “ << i << endl;
      }
      cout << "End” << endl;
}
```


Exception Handling **Example**

```
void main(){ // System will find and execute catch with matching argument.
      cout << "Start“ << endl;
      try {
             cout << "Inside try block\n";
             throw 100; // throw an error
             cout << "This will not execute";
      }
      catch (double i) { // catch an error
             cout << "Caught an exception -- value is: “ << i << endl;
      }
      catch (int i) { // catch an error
             cout << "Caught an exception -- value is: “ << i << endl;
      }
      cout << "End” << endl;
}
```

Exception Handling **Example Division by zero**

```
void main(){
      float x, y;
      cout << "Enter two numbers“ <<endl;
      cin >> x >> y;
      try { // start a try block
              if (y != 0)
                      cout << "Div = " << x / y << endl;
              else
                      throw y;
        }
      catch (float y) { // catch an error
              cout << "Caught Division by " << y << endl;
      }
      cout << "Out of try catch block " << endl;
}
```

```
Enter two numbers
55
23
Div = 2.3913
Out of try catch block
```

```
Enter two numbers
55
0
Caught Division by 0
Out of try catch block
```

Exception Handling **Functions**

- An exception can also occur inside a function.
- A function can either handle the exception by adding local try catch blocks.
- Or the function can simply throw the exception.
  - The caller will be responsible for catching and handling the thrown exception.
- If multiple functions calls are made and a function throws the exception
  - Then there must be at least one caller, which should catch and handle that exception.
  - If all functions simply throw the exception and no one handle the exception, then program will be terminated by system abnormally.
  - **Stack unwinding:** System will search of matching catch block in all functions and execute if find one otherwise may abnormally terminate the program.

Exception Handling **Functions**

- A function can handle the exception by adding local try catch blocks.

```
float Divide(float x, float y){
      try {
         if (y != 0)
             return x / y;
          else
             throw y;
    }
      catch (float y) { // catch an error
         cout << " Inside Divide" << endl;
         cout << "Caught Division by "
          << y << endl;
      }
}
```

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl;
      cin >> x >> y;
      cout << "Div = " << Divide(x, y);
}
```

```
Enter two numbers
55
8
Div = 6.875
```

```
Enter two numbers
55
0
 Inside Divide
Caught Division by 0
Div = -nan(ind)
```

Exception Handling **Functions**

- The function can simply throw the exception.

```
float Divide(float x, float y){
      if (y != 0)
            return x / y;
          else
            throw y;
}
```

```
Enter two numbers
88
0
Inside Caller
Caught Division by 0
```

The caller will be responsible for catching and handling the thrown exception.

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl;
      cin >> x >> y;
      try {
       cout << "Div = " << Divide(x, y);
      }
      catch (float y) { // catch an error
         cout << "Inside Caller" << endl;
         cout << "Caught Division by "
          << y << endl;
      }
}
```

Exception Handling **Functions Stack Unwinding**

- If multiple functions calls are made and a function throws the exception

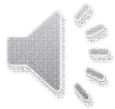
```
float Divide3(float x, float y){
      if (y != 0)
            return x / y;
          else
            throw y;
}

float Divide2(float x, float y){
      return Divide3(x, y);
}

float Divide(float x, float y){
      return Divide2(x, y);
}
```

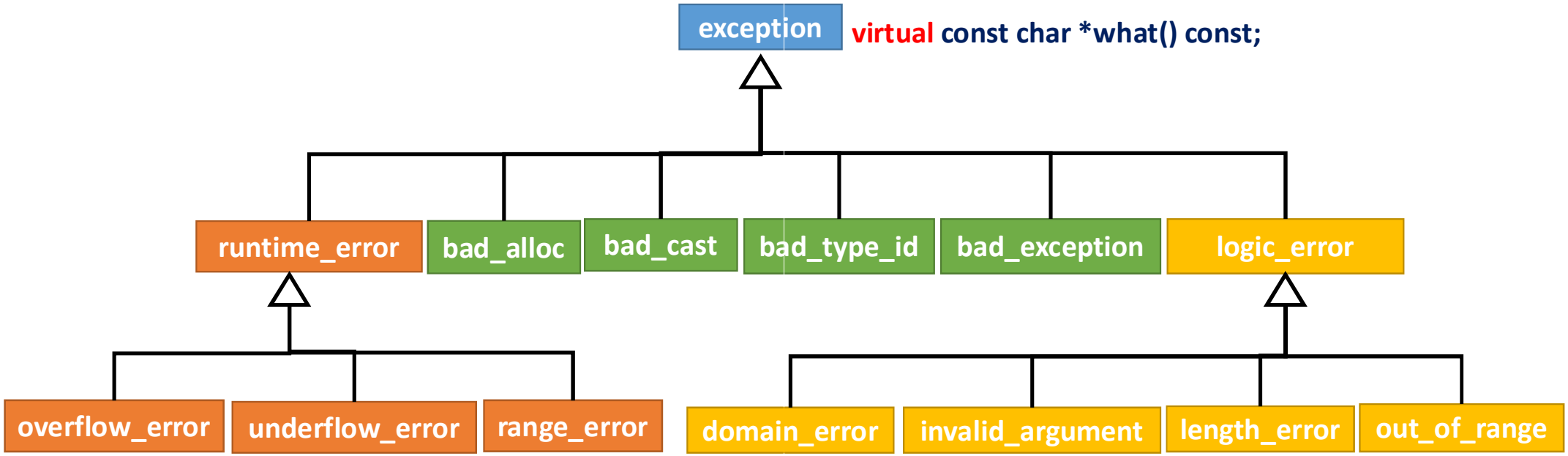
Then there must be at least one caller, which should catch and handle that exception.

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl;
      cin >> x >> y;
      try {
       cout << "Div = " << Divide(x, y);
      }
      catch (float y) { // catch an error
         cout << "Inside Caller" << endl;
         cout << "Caught Division by "
          << y << endl;
      }
}
```



Exception Handling System defined exception classes

`exception`, is a base class which contains virtual function **what** that derived classes can override. If a catch handler catches a reference of base-class type, it can also catch a reference to all derived classes objects, which allows for **polymorphic processing** of related errors.



Exception Handling **System defined exception classes**

- **`bad_alloc`** is thrown by **`new`** when memory is not allocated properly.
- **`bad_cast`** is thrown by **`dynamic_cast`**
- **`bad_typeid`** is thrown by **`typeid`**
- **`logic_error`** is the base class of other exception classes that indicate errors in program logic.
  - **`invalid_argument`** when an attempt is made to pass an invalid value to function.
  - **`length_error`** indicates that a length larger than the maximum size allowed for the object being manipulated was used for that object.
  - **`out_of_range`** indicates that a value, such as a subscript into an array, exceeded its allowed range of values.
- **`runtime_error`** is the base class of other exception classes that indicate execution-time errors.
  - **`overflow_error`** the result of an arithmetic operation is larger than the largest number that can be stored in a given numeric type.
  - **`underflow_error`** the result of an arithmetic operation is smaller than the smallest number that can be stored in a given numeric type.

Exception Handling **Example exception**

```
void main(){
      float x, y;
      cout << "Enter two numbers" <<endl;
      cin >> x >> y;
      try {
            if (y == 0)
                  throw exception ("Division by zero!"); // throw exception object
            if (y < 0)
                  throw exception ("Negative Number!"); // throw exception object
            else
                  cout << "Div = " << x / y << endl;
        }
      catch (exception e) { // catch exception object
            cout << e.what() <<endl;
      }
      cout << "Out of try catch block " << endl;
}
```

```
Enter two numbers
88
0
Division by zero!
Out of try catch block
```

```
Enter two numbers
997
-1
Negative Number!
Out of try catch block
```

Exception Handling **Example bad_alloc**

```
void main(){
      int * arr[5];
       try {
            for (int i = 0; i < 5; i++)
                  arr[i] = new int[100000000];
            cout << "done";
      }
      catch (bad_alloc b){
            cout << b.what() <<endl;
      }
      cout << "Out of try catch block " << endl;
}
```

```
bad allocation
Out of try catch block
```

Exception Handling **Example bad_alloc**

```
void main(){// Add general base exception objects after derived ones
      int * arr[5];
       try {
            for (int i = 0; i < 5; i++)
                  arr[i] = new int[100000000];
            cout << "done";
      }
      catch (exception e) {
            cout << e.what() <<endl;
            cout << "base class dominates" <<endl;
      }
      catch (bad_alloc b){
            cout << b.what() <<endl;
      }
      cout << "Out of try catch block " << endl;
}
```

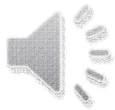
```
bad allocation
base class dominates
Out of try catch block
```

Exception Handling **Exception class inheritance**

- The exception class can be inherited to handle different type of exceptions .
- Override what function according to the class requirements.

```
class arrayIndexoutofBound : public exception{
	public:
	// call the Constructor of base class exception
	arrayIndexoutofBound( const char * msg) :exception(msg){}

	// override the what function.
	const char * what(){
		cout << "Array index out of Bound!" << endl;
		return exception::what();
	}
};
```



Exception Handling **Exception class inheritance**

```
void main() {
	int arr[5] = { 1, 2, 3, 4, 5 };
	 try {
		int i = 0;
		cin >> i;
		if (i >= 5 || i < 0)
			throw arrayIndexoutofBound("Index out of bound");

		arr[i] = 100;
		cout << arr[i];
	}

	catch (arrayIndexoutofBound a) {
		cout << a.what() << endl;
	}
	return;
}
```

```
9
Array index out of Bound
Index out of bound
```

Exception Handling **Special catch block**

- A catch block can take no arguments but three dots and it can catch all type of exceptions.

```
void main(){
        int x;
        cout << "Enter a number" <<endl;
        cin >> x ;
        try { // start a try block
                if (x == 0)
                        throw x;
                else if  (x == 1)
                        throw runtime_error(" runtime ");
                else if  (x == 2)
                        throw logic_error(" logic ");
          }
        catch (...) { // Generic catch with three dots
                cout << "Exception occurred !" << endl;
                cout << "What type I dont know!" << endl;
          }
}
```

```
Enter a number
0
Exception occurred !
What type I dont know!
```

```
Enter a number
1
Exception occurred !
What type I dont know!
```

```
Enter a number
2
Exception occurred !
What type I dont know!
```