Data Structures and Object Oriented Programming

Lecture 27

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Return Error Code

- The error handling code increases the complexity of the code
  - Error handling code is mixed with program logic
  - The code becomes less readable
  - Difficult to modify

Example

```
int main()
{
    if (!function1())
    {
        if (!function2())
        {
            if (!function3())
            {
                ...
            }
            else cout << “Error Z has occurred”;
        }
        else cout << “Error Y has occurred”;
    }
    else cout << “Error X has occurred”;

    return 0;
}
```

Exception Handling

- C++ language provides built-in Exception Handling
  - Is a much elegant solution
  - Enables separation of main logic and error handling code
  - Release local resources automatically

Exception Handling Process

- Programmer writes the code that is suspected to cause an exception in **try block**
- Code section that encounters an error **throws** an object that is used to represent exception
- **Catch blocks** follow try block to catch the object thrown

Exception Handling Process

try

- Put statements and function calls that may generate exceptions in a **try** block
- Each try block is associated with a sequence of handlers that follow immediately
- **try** blocks can be nested

```
try
{
    try
    {
       ...
    }
}
```

throw

- Code section that encounters an error **throws** an object that is used to represent exception
- Any object (built-in or user-defined) can be thrown

```
throw 1.2;

MyClass obj;
throw obj

int x=10;
throw X


throw MyClass();
```

catch

- **Catch** block is the handler
- Must immediately follow the try block
- **Catch** handlers are differentiated on the basis of argument type

```
try
{
   ...
}
catch(int x)
{
  ...
}
catch(MyClass x)
{
  ...
}
```

Examples

```
void GetNumbers(int& a, int& b)
{
    cout << "Enter two integers"<<endl;
    cin >> a >> b;
}

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


void OutputQuotient(int a, int b, int quo)
{
    cout << "Quotient is " << quo << endl;
}
```

```
int main() {
    int quot;
    int a, b;
    for (int i = 0; i < 10; i++)
    {
        try
        {
            GetNumbers(a, b);
            quot=Quotient(a, b);
            OutputQuotient(a, b, quot);
        }
        catch(int ex)
        {
            i--;
            cout << “Denominator Zero" << endl;
        }
    }
    return 0;
}
```

Examples

```
class DividbyZero
{

};
void GetNumbers(int& a, int& b)
{
    cout << "Enter two integers"<<endl;
    cin >> a >> b;
}

int Quotient(int a, int b)
{
    if (b == 0) {
        throw DividbyZero();
    }
    return a / b;
}

void OutputQuotient(int a, int b, int quo)
{
    cout << "Quotient is " << quo << endl;
}
```

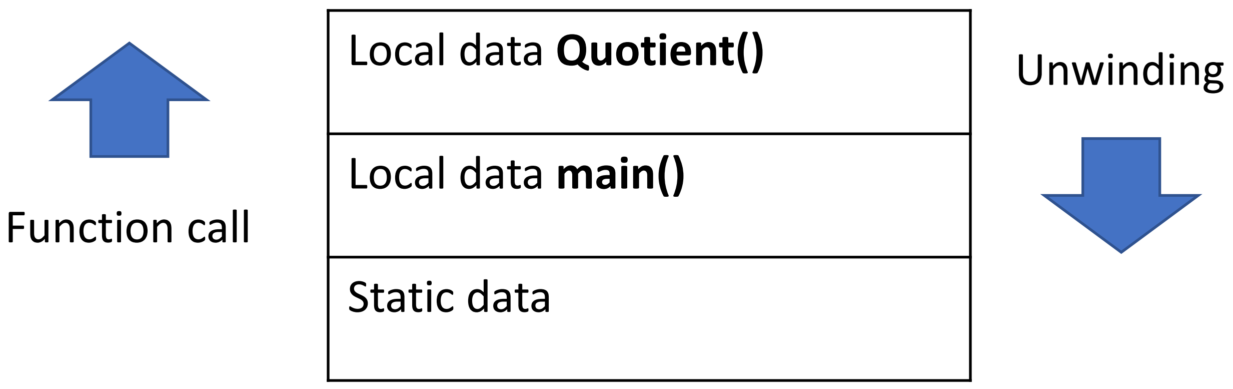
```
int main() {
    int quot;
    int a, b;
    for (int i = 0; i < 10; i++)
    {
        try
        {
            GetNumbers(a, b);
            quot=Quotient(a, b);
            OutputQuotient(a, b, quot);
        }
        catch(DividbyZero)
        {
            i--;
            cout << “Denominator Zero" << endl;
        }
    }
    return 0;
}
```

Syntax – Try and Catch

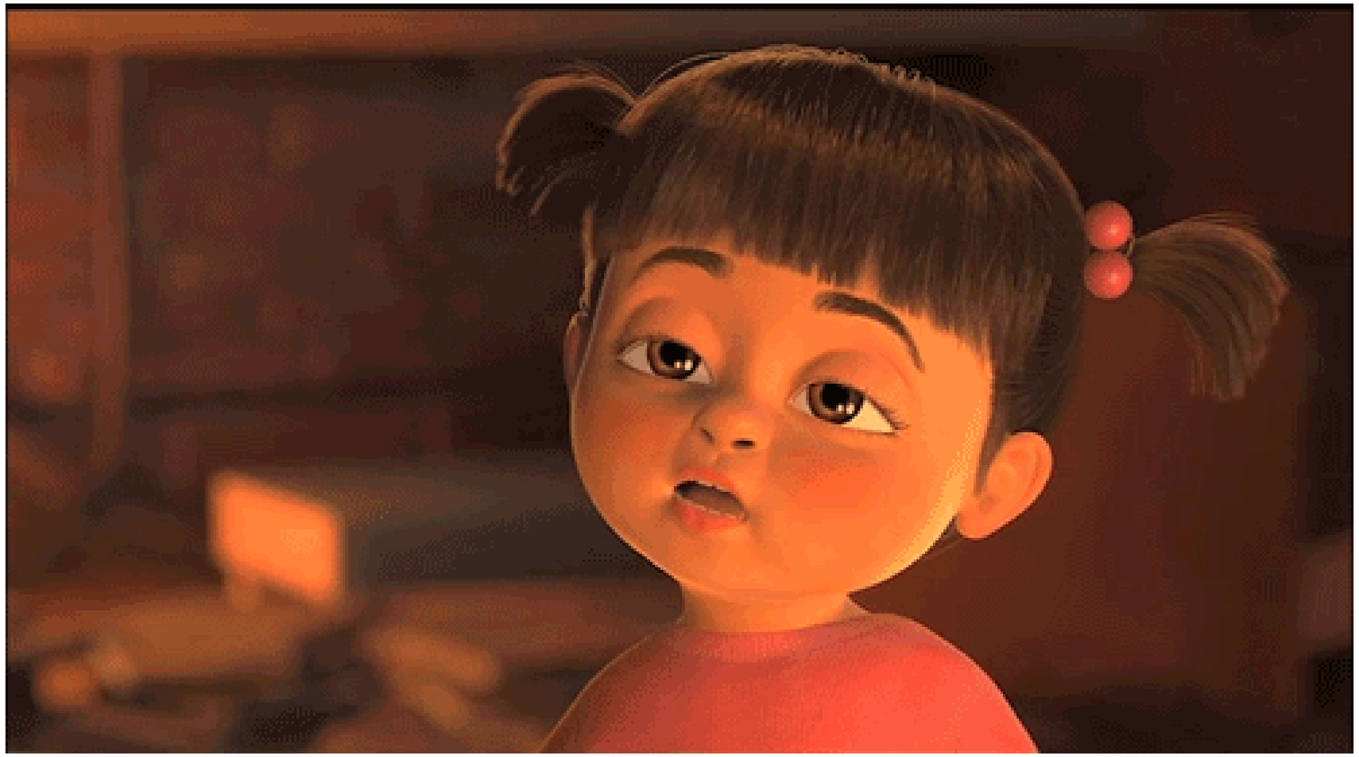
```
int main() {
    int quot;
    int a, b;
    for (int i = 0; i < 10; i++)
    {
        try
        {
            GetNumbers(a, b);
            quot=Quotient(a, b);
            OutputQuotient(a, b, quot);
        }
        catch(DividbyZero)
        {
            i--;
            cout << “Denominator Zero" << endl;
        }
        catch(SomeOtherError)
        {
        }

    }
    return 0;
}
```

Stack Unwinding

- Passing an exception while searching for a handler can cause abnormal exit from a function while in middle of executing it (i.e., without any return value)
  - The stack frame corresponding to the exited function's scope is **removed** – **this is called stack unwinding**
  - So the lifetime of local objects in the exited functions ends



Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over