Exception Handling II

DM2233 ADVANCED DATA STRUCTURES & ALGORITHMS

Module Schedule

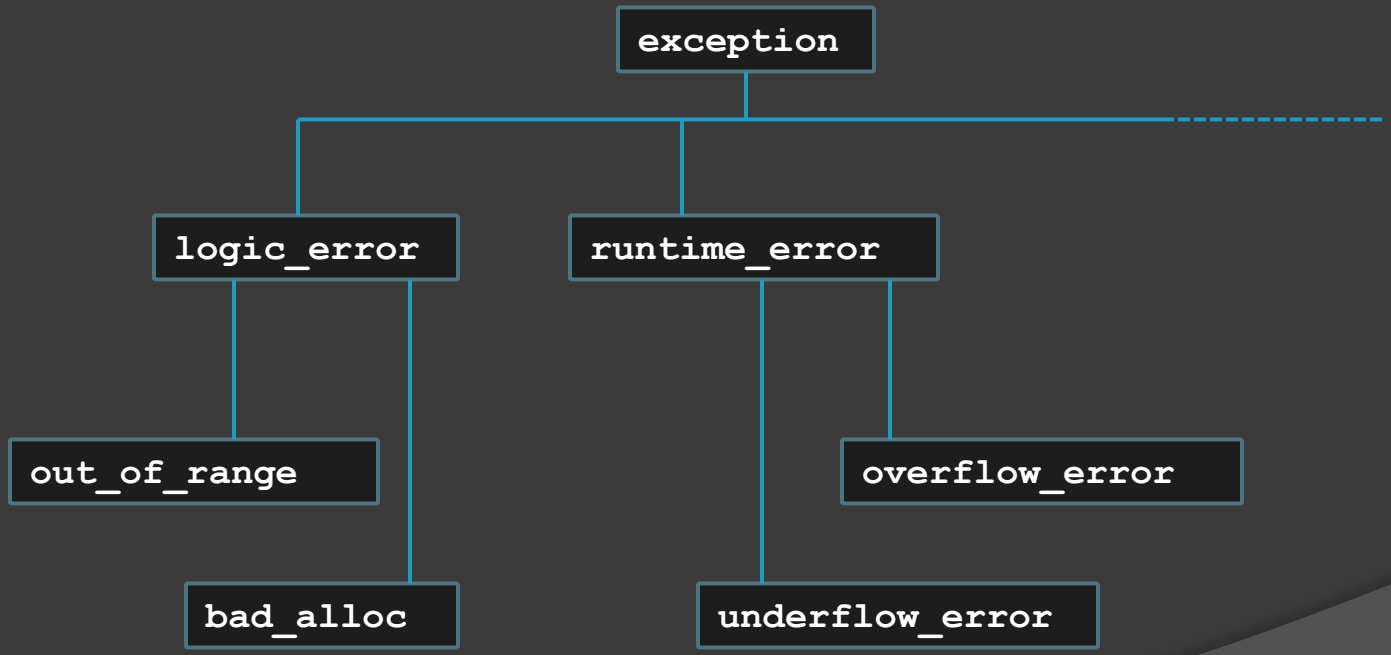
Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Preprocessing / Assignment 1	Vesak Day (Mon)
Week 8 and 9: Mid-Sem Break		
10	Sorting and Searching I	
11	Sorting and Searching II	
12	Sorting and Searching III	
13	Binary Tree I	Hari Raya Puasa (Fri)
14	Lab Test	
15	Binary Tree II	
16	Binary Tree III	SG50 Day (Fri)
17	Standard Template Library / Assignment 2	National Day (Mon)

Objective

- Exception Classes
- Creating your own Exception Classes
- Rethrowing an Exception

Exception Classes

- C++ provides support to handle exceptions via a hierarchy of classes



Exception Classes

- Example 1

```
void main (void) {
    string sentence, str1, str2, str3;

    try {
        sentence = "Testing string exceptions!";
        cout << "sentence = " << sentence << endl;
        cout << " length = " << sentence.length() << endl;

        str1 = sentence.substr (8, 20);
        cout << "str1 = " << str1 << endl;

        str2 = sentence.substr (28, 10);
        cout << "str2 = " << str2 << endl;

    } catch (out_of_range re) {
        cout << "out of range: " << re.what() << endl;
    }
}
```

Exception Classes

- Example 2

```
void main (void) {
    int * list [100];

    try {
        for (int i = 0; i < 100; i ++) {
            list[i] = new int [50000000];
            cout << "created list[" << i << "]" << endl;
        }
    } catch (bad_alloc be) {
        cout << "caught: " << be.what() << endl;
    }
}
```

```
created list[0]
created list[1]
created list[2]
created list[3]
created list[4]
created list[5]
caught: bad allocation
```

Creating your own Exception Class

- Exceptions are likely to occur
- Although C++ provides numerous exception classes to deal with common situations, it may not cover the exceptions you need
- To solve that, C++ enables the programmer to write his own exception classes

Creating your own Exception Class

- Example 1

```
class myExcp {};

void main (void) {
    int dividend, divisor, quotient;

    try {
        cout << "Enter dividend: ";
        cin >> dividend;

        cout << "Enter divisor: ";
        cin >> divisor;

        if (divisor == 0) throw myExcp();
        quotient = dividend / divisor;

        cout << "Quotient = " << quotient << endl;

    } catch (myExcp) {
        cout << "Exception!!";
    }
}
```

Creating your own Exception Class

- Example 2

```
class myExcp {
  private:
    string msg;

  public:
    myExcp (void) {
      msg = "Divide by zero";
    }

    myExcp (string str) {
      msg = str;
    }

    string what (void) {
      return msg;
    }
}
```

```
output
Divide by zero
```

```
void main (void) {
  int dividend, divisor, quotient;

  try {
    cout << "Enter dividend: ";
    cin >> dividend;

    cout << "Enter divisor: ";
    cin >> divisor;

    if (divisor == 0)
      throw myExcp();
    quotient = dividend / divisor;

    cout << "Quotient = "
         << quotient << endl;

  } catch (myExcp mE) {
    cout << mE.what();
  }
}
```

Creating your own Exception Class

- Example 3

```
class myExcp {
  private:
    string msg;

  public:
    myExcp (void) {
      msg = "Divide by zero";
    }

    myExcp (string str) {
      msg = str;
    }

    string what (void) {
      return msg;
    }
}
```

```
output
another err
```

```
void main (void) {
  int dividend, divisor, quotient;

  try {
    cout << "Enter dividend: ";
    cin >> dividend;

    cout << "Enter divisor: ";
    cin >> divisor;

    if (divisor == 0)
      throw myExcp("another err");
    quotient = dividend / divisor;

    cout << "Quotient = "
         << quotient << endl;

  } catch (myExcp mE) {
    cout << mE.what();
  }
}
```

Rethrowing an Exception

```
class myExcp {...}

void divide (void) {
    try {
        ...
        if (divisor == 0) throw myExcp ("Divide by 0");
        ...
    } catch (myExcp mE) {
        throw;
    }
}

void main (void) {
    try {
        divide ();
    } catch (myExcp mE) {
        cout << "main: " << mE.what();
    }
}
```

Rethrowing an Exception

```
class myExcp {...}

void divide (void) {
    try {
        ...
        if (divisor == 0) throw myExcp ("0");
        ...
    } catch (myExcp mE) {
        throw string ("Divide by " + mE.what());
    }
}

void main (void) {
    try {
        divide ();
    } catch (string msg) {
        cout << "main: " << msg;
    }
}
```

Summary

- We had just discussed about,
  - Handling Exceptions
  - Try / Catch
  - Throwing Exceptions
  - Restricting Exceptions