Exception Handling I

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

- Handling Exceptions
- Try / Catch
- Throwing Exceptions
- Restricting Exceptions

Handling Exceptions

- What's wrong with the following?

  (Assuming User will always enter the correct data type, integer.)

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

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

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

  quotient = dividend / divisor;
  cout << "Quotient = " << quotient << endl;

}
```

Handling Exceptions

- Solution 1

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

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

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

  if (divisor != 0) {
    quotient = dividend / divisor;
    cout << "Quotient = " << quotient << endl;
  } else {
    cout << "Cannot divide by zero";
  }

}
```

Handling Exceptions

- Solution 2

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

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

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

  assert (divisor != 0);
  quotient = dividend / divisor;
  cout << "Quotient = " << quotient << endl;

}
```

Handling Exceptions

- The syntax of assert:

```
assert (<expression>);
```

- <expression> is any logical expression
- If <expression> evaluates to true, the next statement executes
- Otherwise, program terminates indicating where the error occurs
- Good for debugging, not release
  - Remove after debugging
  - Compile with #define NDEBUG

Handling Exceptions

- Solution 3

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

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

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

  try {
    if (divisor == 0) throw 0;
    quotient = dividend / divisor;
    cout << "Quotient = " << quotient << endl;

  } catch (int x) {
    cout << "Divide by " << x << endl;
  }

}
```

Try / Catch

- Statements that may generate exceptions are placed in a try block
- The try block is followed by one or more catch blocks
- A catch block specifies the type of exception it can catch and contains an exception handler

Try / Catch

- General syntax

```
try {
   ...     // statements

} catch (<type1> <id>) {
   ...     // handle <type1> exception

} catch (<type2> <id>) {
   ...     // handle <type2> exception

   .
   .       // more catch blocks
   .

} catch (...) {
   ...     // catch any type of exception

}
```

Try / Catch

- If no exception is thrown in a try block, all associated catch blocks are ignored; execution resumes after the last catch block
- If an exception is thrown, the remaining statements in the try block are ignored; catch blocks are searched (in order) for the appropriate handler
- Usually the last catch block is a catch-all (…)
- The order of catch blocks is significant

Try / Catch

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

```
catch (int) {
   ...
}
```

- x is a catch block parameter; a catch block can have at most one catch block parameter
- If there is no catch block parameter, the exception handling code have no access to the value thrown

Try / Catch

- For an exception to occur in a try block and be caught, it must be thrown

```
int num = 5;
string str = "Hello";

throw 4;
throw num;
throw str;
```

Throwing Exceptions

- Example

```
try {
  if (divisor == 0)
    throw 0;
  else if (divisor < 0)
    throw string ("Negative divisor");

  quotient = dividend / divisor;
  cout << "Quotient = " << quotient << endl;

} catch (int x) {
  cout << "Cannot divide by " << x << endl;

} catch (string x) {
  cout << x << endl;

}
```

Restricting Exceptions

- Throwing Exceptions from Functions

```
void myFunction(int test)
{
  if(test==0)
     throw test;                  // throw int
  if(test==1)
     throw 'a';                   // throw char
}
```

```
void main()
{
  try{
    myFunction(0);
  }
  catch(int i) {
    cout << "Caught an integer" << endl;
  }
  catch(char c) {
    cout << "Caught char" << endl;
  }
}
```

Throwing Exceptions

This is deprecated in newer version of VC++. FYI only.

- When declaring a function, we can restrict the exception types it might directly or indirectly throw

```
void test (void);                          // all exceptions allowed

void test (void) throw ();                 // no exceptions allowed

void test (void) throw (int);              // int exceptions allowed

void test (void) throw (int, string); // int and string exceptions allowed
```

Restricting Exceptions

This is deprecated in newer version of VC++. FYI only.

- Example

```
void myFunction(int test) throw(int, char, double)
{
  if(test==0)
     throw test;                    // throw int
  if(test==1)
     throw 'a';                     // throw char
}
```

```
void main()
{
  try{
    myFunction(0);
  }
  catch(int i) {
    cout << "Caught an integer\n";
  }
  catch(char c) {
    cout << "Caught char\n";
  }
}
```

Summary

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