Exception Handling

Prof. Jinkyu Jeong (jinkyu@skku.edu)
TA -- Minwoo Ahn (minwoo.ahn@csl.skku.edu)
TA -- Donghyun Kim (donghyun.kim@csl.skku.edu)

Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
What is Exception?

• An exception is a problem that arises at “runtime”
• e.g., divide by zero

```c
int main(void) {
    int num1, num2;
    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;
    
    cout << "Quotient: " << num1/num2 << endl;
    cout << "Remainder: " << num1 % num2 << endl;
    
    return 0;
}
```

What if num2 is zero?

Input (2 numbers): 3 0
Floating point exception
Exception Handling by “if”

- Divide by zero can solved by conditional check

```c
int main(void) {
    int num1, num2;
    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;
    if(num2 == 0) {
        cout << "num2 is zero" << endl;
    }
    else {
        cout << "Quotient:" << num1/num2 << endl;
        cout << "Remainder:" << num1 % num2 << endl;
    }
    return 0;
}
```

Then... what is the problem?
Readability

• Developer cannot recognize difference of
  – Logical functionality, and
  – Exception

```c
int main(void) {
    int num1, num2;
    cout << “Input (2 numbers): “;
    cin >> num1 >> num2;

    if(num2 == 0) {
        cout << “num2 is zero” << endl;
    } else if(num2 > num1) {
        cout << “Quotient: 0” << endl;
        cout << “Remainder: “ << num1 << endl;
    } else {
        cout << “Division: “ << num1/num2 << endl;
        cout << “Remainder: “ << num1 % num2 << endl;
    }
    return 0;
}
```
Exception Handling in C++

• try-throw-catch
  – try: detect an exception
  – throw: raise an exception
  – catch: handling exception

```cpp
try {
    ...  
    if(exception occurs)
        throw exception;
    ...
}

catch(type exception) {
    ...  
    //handling of exception  
    ...
}
```

Format of try-throw-catch

(…) indicates all types of exception object
```cpp
int main(void) {
    int num1, num2;
    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;

    try {
        if(num2 == 0)
            throw num2;
        cout << "Quotient: " << num1 / num2 << endl;
        cout << "Remainder: " << num1 % num2 << endl;
    } catch(int expn) {
        cout << "num2 is zero" << endl;
    }
    cout << "end of main" << endl;
    return 0;
}
```

(case1)
Input (2 numbers): 9 2
Quotient: 4
Remainder: 1
end of main

(case2)
Input (2 numbers): 7 0
num2 is zero
end of main
Try Block

• Instructions are executed in order inside try block
• Inside try block,
  – If exception raised, jump to catch block (remaining parts are ignored)
  – If there’s no raised exception, jump to end of catch block

```c
int main(void) {
    int num1, num2;

    cout << "Input (2 numbers): ";
    cin >> num1 >> num2;
    try {
        if(num2 == 0)
            throw num2;
    } catch(int expn) { … }
    cout << "Quotient: " << num1 / num2 << endl;
    cout << "Remainder: " << num1 % num2 << endl;
    return 0;
}
```

Should be moved to try block
Stack Unwinding

```c
void Divide(int num1, int num2) {
    if(num2 == 0)
        throw num2;
    cout << "Quotient: " << num1/num2 << endl;
    cout << "Remainder: " << num1%num2 << endl;
}

int main(void) {
    ... 
    try {
        Divide(num1, num2);
        cout << "Division Finished" << endl;
    } catch(int expn) {
        ...
    }
    return 0;
}
```

There is no `catch` block inside function

Propagate data of exception (similar to return)
Nested Function Call

int main(void) {
    try {
        Func1();
    } catch(int expn) {
        cout << "Exception #: " << expn << endl;
    }
    return 0;
}

void Func1(void) {
    ...
    Func2();
    ...
}

void Func2(void) {
    ...
    Func3();
    ...
}

void Func3(void) {
    ...
    throw -1;
    ...
}
Stack Unwinding (cont’d)

- What if there is no \textit{catch} block inside function?
  - Try to find \textit{catch} block of caller

![Call stack diagram]

- Exception propagation diagram
Finding Catch Block

- Follow callers (toward main) until appropriate catch block is found
- If type of exception objects mismatch, it’s not appropriate
```c
int main(void) {
    ...
    try {
        cout << str1 << " + " << str2 << " = " << StoI(str1)+StoI(str2) << endl;
    }
    catch(char ch) {
        cout << "Character " << ch << "cannot be input" << endl;
    }
    catch(int expn) {
        if(expn == 0)
            cout << "input cannot start with zero" << endl;
        else
            cout << "Not allowed input" << endl;
    }
    ...
}
```
Explicit Exception Object Type

```c
1: int ThrowFunc(int num) throw (int, char) {
2:    ...
3:    if(...) 
4:        throw 3;
5:    ...
6:    if(...) 
7:        throw 'a';
8:    ...
9:}
```

```c
10:int main() {
11:    ...
12: try {
13:        ...
14:        ThrowFunc(20);
15:        ...
16:    }
17: catch(int expn) {...}
18: catch(char expn) {...}
19:}
```

“throw 3” caught by line 17
“throw ‘a’” caught by line 18
Exception Class

- Class can be exception object of *throw* block

```cpp
try {
    Account.Deposit(2000);
    Account.Deposit(-300);
} catch (DepositException &expn) {
    expn.ShowExceptionReason();
}
```

```cpp
void ShowExceptionReason::DepositException(int deposit) {
    cout << "Cannot deposit" << deposit << endl;
}
```

**OUTPUT**
Cannot deposit -300
try {
    ...
    throw A;
    throw B;
    throw C;
    ...
}
catch(A &expn) {
    cout << "Exception caught by A" << endl;
}
catch(B &expn) {
    cout << "Exception caught by B" << endl;
}
catch(C &expn) {
    cout << "Exception caught by C" << endl;
}

OUTPUT
Exception caught by A
Exception caught by A
Exception caught by A

Thrown of class B and C also caught by A
try {
    ... 
    throw A;
    throw B;
    throw C;
    ...
} 
catch(C &expn) {
    cout << "Exception caught by C" << endl;
}
catch(B &expn) {
    cout << "Exception caught by B" << endl;
}
catch(A &expn) {
    cout << "Exception caught by A" << endl;
}

OUTPUT
Exception caught by A
Exception caught by B
Exception caught by C
Pre-defined Exception: bad_alloc

• Exception caused by new operator

```cpp
int main() {
    int num=0;

    try {
        while(1) {
            num++;
            cout << num << " times allocated" << endl;
            new int[10000][10000];
        }
    } catch { 
        cout << bad.what() << endl;
        cout << "Cannot allocate more!" << endl;
    }
    return 0;
}
```

OUTPUT
1 times allocated
2 times allocated
3 times allocated
bad allocation
Cannot allocate more!
[Lab - Practice #1]

• Exception handling of “space management”

• Exceptions to be handled
  – Date out of range
    • Date of ~09/12/30
  – Non-exist space type
  – Non-exist operation
  – Non-exist member type
  – Member name with numbers
  – Negative time

• You should print in output.dat with
  – Return code: -1
  – Description: Exception name