Exceptions

August 8, 2020

1 Errors & Exceptions

http://www.cplusplus.com/doc/tutorial/exceptions/ http://openbookproject.net/thinkcs/python/english3e/excep

1.1 Table of Contents

- **Section 1.2**
- **Section 1.3**
- **Section ??**

1.2 Errors

- errors are also called bugs in computer programs
- as long as humans will write computer programs, there will always be some bugs in programs
- the process of finding and getting rid of bugs is called debugging and is an integral part of programming
- three types of errors:
  1. **Syntax Errors**
     - errors in the grammar dictated by the programming language
     - programs will not run if there is a syntax error
     - compiler usually provides feedback on syntax errors
     - annoying in the beginning but one will get better as they become more proficient and experienced with the language
     - e.g., forgetting a colon or } or] where they need to be
  - **Semantic Errors**
    - semantic errors are errors in programs that produce wrong answers
    - program will run successfully without generating any error messages
    - program will not do the right thing (does what you tell it to do!)
    - essentially, the problem is that the program you wrote is not the program you wanted to write
      - the meaning of the program (its semantics) is wrong
    - testing your program with lots of input samples will help you reveal semantic errors
    - identifying semantic errors can be tricky and requires you to work backward (trace-back) by looking at the output and figure out what it is doing
  - **Exceptions**
    - run-time errors that may manifest at certain circumstances

1.2.1 Exercise

Which of the following is a semantic error? 1. Attempting to divide by 0 - Forgetting a semicolon at the end of a statement where one is required - Forgetting to divide by 100 when printing a percentage amount

1.3 Exceptions

- exceptions are runtime errors that may occur in exceptional circumstances
- unlike syntax error and logical/semantic errors, exceptions may or may not manifest while the program is being executed
- when exception is thrown, program will halt if the exception is not handled!
  - can be catastrophic if the OS of Satellites, Mars Rovers, Airplanes halts (crashes)
- C++ provides a mechanism to react to exceptional circumstances (runtime errors) in programs by transferring control to special functions called **handlers**
- to catch exceptions, a portion of code is placed under exception inspection block called a **try-block**
- when the exceptional circumstance arises within that block, an exception is thrown that transfers the control to the exception handler called **catch-block**
- 3 options to handle exceptions
  1. if possible try to recover or correct error
  - if not possible to recover, log the error and continue
  - if option 1 or 2 is not possible, halt the program
- if no exception is thrown, the code continues normally and all handlers (catch-blocks) are ignored
- syntax:

```
try {
    // try code that may throw exceptions
}
catch(type1 name1) {
    // first handler that throws exception of type1
}
catch(type2 name2) {
    // second handler that throws exception of type2
}
... // other handlers
    // more generic handlers should be written towards the end
catch(...) { // ellipsis
    // catch all exceptions if previous handler couldn't catch
}
```

- various exception types
  - https://en.cppreference.com/w/cpp/error/exception

[2]:
```
#include <iostream>
#include <string>
#include <vector>
#include <exception>
```

```
#include <cstdio>

using namespace std;
```

[2]:
```
try {
    cout << "executed..." << endl;
    throw 20; //throw an integer exception
}
catch (int ex) {
    cout << "Exception occured. #: " << ex << endl;
}
```

executed…
Exception occured. #: 20

1.4 Example 1 - string.at() and string.substr() may throw out_of_range exception

[3]:
```
// at method of string class throws std::out_of_range if pos >= size().
string name = "John Doe";
cout << name.at(0) << endl;
```

J

[4]:
```
cout << name.at(20) << endl;
cout << "continue with the rest..." << endl; // never executes
```

Error:

[5]:
```
// handle the exception
try {
    cout << name.at(20) << endl;
}
catch(const out_of_range& e) {
    cout << "exception occured: " << e.what() << endl;
}
cout << "continue with the rest..." << endl; // will continue from here
```

exception occured: basic_string
continue with the rest…

[5]: @0x104a8c010

[6]:
```
// substr() method throws out_of_range exception
cout << name.substr(5) << endl;
cout << "continue with other code..." << endl;
```

```
Doe
continue with other code…
```

[7]:
```
try {
    cout << name.substr(20) << endl;
}
catch(int e1) {
    cout << "Interger exception occured: exception #: " << e1 << endl;
}
cout << "continue with other code..." << endl;
```

```
        Error:
```

[11]:
```
// substr() method throws out_of_range exception
try {
    cout << name.substr(20) << endl;
}
catch(int e1) {
    cout << "Interger exception occured: exception #: " << e1 << endl;
}
catch(const out_of_range & ofr) {
    cout << "exception: out_of_range: " << ofr.what() << endl;
    // can catch out_of_range exception to know exactly what exception was␣
 ↪thrown
}
catch(...) {
    cout << "some exception flew by..." << endl;
}
cout << "continue with other code..." << endl;
```

```
exception: out_of_range: basic_string
continue with other code…
```

1.5 Example 2 - bad_alloc exception

https://en.cppreference.com/w/cpp/memory/new/bad_alloc

[1]:
```
#include <iostream>
using namespace std;
```

[2]:
```
unsigned long long int n;
int *nums;
```

[4]:
```
cout << "Enter how many integers would you like to store?";
cin >> n;
nums = new int[n]; // try 10000000000000
```

```
cout << "successfully allocated memory to store " << n << " integers\n";
```

```
Enter how many integers would you like to store?100000000000000

        Standard Exception: std::bad_alloc
```

[5]:
```
n = 0;
cout << "Enter how many integers would you like to store?";
cin >> n;
try {
    nums = new int[n]; // try 10000000000000
    cout << "successfully allocated memory to store " << n << " integers\n";
}
catch(const bad_alloc & e) {
    cout << "Exception occured: " << e.what() << endl;
}
cout << "continue..." << endl;
```

```
Enter how many integers would you like to store?10000000000000000
Exception occured: std::bad_alloc
continue…
```

[6]:
```
vector<int> v(10);
try {
    // try resizing vector to a large value
    v.resize(1000000000000000000);
}
catch(const bad_alloc & e) {
    cout << "Exception occured: " << e.what() << endl;
}
cout << "size of nums = " << v.size() << endl;
```

```
Exception occured: std::bad_alloc
size of nums = 10
```

1.6 Example 3 - input validation

[1]:
```
int decimalNum;
bool valid;
```

[5]:
```
// User input with validation
valid = false;
do
{
    cout << "Enter a number in decimal: ";
    try
```

```
    {
        cin >> decimalNum;
        if (cin.fail())
        {
            throw "Invalid input. Tray again!";
        }
        valid = true;
    }
    catch (const char *error)
    {
        cin.clear(); // clear the cin failure state
        cin.ignore(100, '\n'); // ignore next 100 characters or up to \n char
        printf("%s\n", error);
    }
} while (!valid);

cout << "Thank you for entering a number: " << decimalNum << endl;
```

```
Enter a number in decimal: adsf324
Enter a number in decimal: adf
Enter a number in decimal: sdaf
Enter a number in decimal: 234
Thank you for entering a number: 234
```

1.7 Division by zero

- in C++ division by zero is not an exception!
- typically program crashes without throwing an exception

[6]:
```
cout << 3/0 << endl;
```

```
input_line_13:2:11: warning: division by zero is
undefined [-Wdivision-by-zero]
 cout << 3/0 << endl;
          ^~
```

21

1.8 Create your own exception class

[9]:
```
// DivisionByZero class inherits from base exception class
class DivisionByZero {
  private:
    string description;
  public:
    DivisionByZero(const string& des="Zero Division Error") {
```

```
        this->description = des;
    }
    string what() const {
        return this->description;
    }
};
```

[7]:
```
int divisor, dividend;
int quotient;
```

[11]:
```
cout << "Enter divisor and dividend separated by space: ";
cin >> divisor >> dividend;
//assert(divisor != 0); // can do this
try {
    if (divisor == 0)
        throw DivisionByZero();
    quotient = dividend/divisor;
    cout << dividend << " / " << divisor << " = " << quotient << endl;
}
catch(const DivisionByZero& e) {
    cout << "Exception: " << e.what() << endl;
}
cout << "continue...";
```

```
Enter divisor and dividend separated by space: 0 4
Exception: Zero Division Error
continue…
```

[ ]: