Programming Languages: Part 3

Robert M. Dondero, Ph.D.
Princeton University
Objectives

- You will learn/review:
  - In C, Java, and Python...
  - Classes, objects, object references
  - Command-line arguments
  - Files
  - Regular expressions
"Fraction" Modules

- Illustrate:
  - Object-oriented features...
  - Classes
  - Object references
  - Objects
"Fraction" in C

- See `fraction.h`
  - Definition of opaque pointer type
    - "Fraction" is an alias for "struct Fraction"!
  - Declarations of functions
See `fraction.c`

- Structure type definition
- Function definitions; some "static"
- `Fraction_print()` function
- Parameter validation via `assert()`
- Functions do not create result objects
- Conditional def of `main()` for unit testing
"Fraction" in C

- Generalizing
  - Any function should be either:
    - **Declared** in module interface and defined non-static
    - **Not declared** in module interface and defined static
  - Requirement to explicitly free objects dramatically affects program design
  - Parameter validation...
C Parameter Validation

- "Private" (static) function
  - Validate parameters via assert
  - Enabled by default
  - Can disable at preprocess-time

- `$ gcc -Wall -ansi -pedantic -D NDEBUG fraction.c readnum.c -o fraction`
"Public" (non-static) function

- Validate parameters via:
  - Assert?
    - Can disable
    - Could crash program!
  - "If" statement and return value?
    - What if return value has other use?
    - What if caller doesn't check?

- No good choices
"Fraction" in Java

- See Fraction.java
  - Constructors
  - Method overloading (of constructors)
  - Constructor chaining
  - Private vs. public methods
  - Overriding of methods inherited from Object
    - toString(), equals(), hashCode()
  - toString() instead of print()
  - Object creation via "new" operator
Validation of public method parameters via "if" and "throw"

Validation of private method parameter via "assert"

Definition of main() method for unit testing

Generalizing

Parameter validation...
Java Parameter Validation

- Public method
  - Validate parameters with "if" and "throw"
  - Cannot (and should not) be disabled
Java Parameter Validation

- Private method
  - Validate parameters with "assert"
  - Disabled by default
    - Appropriate for released code???
  - Can enable at run-time
    - $ java -ea Fraction
    - Appropriate when testing
"Fraction" in Python

- See *frac.py*
  - Inheritance from "object" class
  - Constructor: `__init__()`
    - Creation of fields within constructor
  - Default arguments
  - *Lack of* private vs. public members
    - Use of underscore to *suggest* private
"Fraction" in Python

- Overriding of methods inherited from "object"
  - `__str__()`, `__eq__()`, `__ne__()`, `__hash__()`
- Operator overloading
  - `__eq__()` method defines `==` operator
  - `__ne__()` method defines `!=` operator
- Validation of method parameters via "if" and "raise"
- Validation of "private" method parameter via "assert"
"Fraction" in Python

- Definition of main() method for unit testing
- Object creation by "calling the class"
- Named actual parameters
- Use of overloaded operators
- Generalizing
  - Parameter validation...
"Public" method

- Validate parameters with "if" and "raise"
- Cannot (and should not) be disabled
- Often must validate type (unlike Java)
Python Parameter Validation

- "Private" method
  - Validate parameters with "assert"
  - Enabled by default
    - Appropriate when testing
  - Can disable at run-time
    - $ python -O fraction.py
      - Appropriate for released code???
Artificial

Illustrate:

- Data of primitive types are passed by value
- Objects are passed by reference
  - More precisely, object references are passed by value
"Calls" in C

- See `calls1.c`
  - What does it print?
  - Data of primitive types are passed by value

- See `calls2.c`
  - What does it print?
  - "Object" (really, structure) references are passed by value
"Calls" in Java

- See `Calls1.java`
  - What does it print?
  - Data of primitive types are passed by value
- See `Calls2.java`
  - What does it print?
  - Object references are passed by value
"Calls" in Python

- See calls1.py
  - What does it print?
  - Data of primitive types are passed by value
- See calls2.py
  - What does it print?
  - Object references are passed by value
"DivMod" Programs

- Illustrate:
  - How a function can "return" multiple values
"DivMod" in C

- See testdivmod1bad.c
  - Fails!
  - int is passed by value
- See testdivmod2.c
  - Works
  - Uses call-by-reference via pointers
"DivMod" in Java

- See TestDivMod1Bad.java
  - Fails!
  - int is passed by value

- See TestDivMod2Bad.java
  - Wrapper classes
  - Fails!
  - Integer class is immutable!!!
"DivMod" in Java

- See TestDivMod3.java, MutableInt.java
  - Works, but bulky
- See TestDivMod4.java, IntPair.java
  - Works, but bulky
- Or could return an array of two ints
  - As in Assignment 1
  - But what if the types of the two items differ?
"DivMod" in Python

- See `testdivmod.py`
  - Works!
  - Function returns a "tuple"
  - Caller uses unpacking assignment
"DivMod" in Python

- Generalizing
  - Examples of tuples:
    - (1, 'hi', 2.3, False, None)
    - (1,)
  - Non-example of a tuple
    - (1)
"Immutable" Programs

- Artificial
- Illustrate:
  - Strings
  - The danger of strings in C
  - The immutability of strings in Java and Python
"Immutable" in C

- See `immutable.c`
  - What does it do?
  - A common C memory management error
"Immutable" in Java

- See Immutable.java
  - What does it print?
  - Strings are objects
  - Strings are immutable

- Generalizing
  - Alternative: StringBuffer class
"Immutable" in Python

- See `immutable.py`
  - What does it print?
  - Strings are objects
  - Strings are immutable

- Generalizing
  - Alternatives:
    - `UserString.MutableString` (educational purposes only)
    - Create list of strings, then `".".join(list)`
"Formatter" Programs

- The job:
  - Read from stdin or file
    - File name specified as command-line arg
  - Write to stdout
    - No more than 60 chars per line
    - Don't split words

- Dr. Kernighan uses to format e-mail
- Keep in mind for Assignment 2
"Formatter" Programs

- Illustrate:
  - String manipulation
  - Files
  - Command-line arguments
  - Regular expressions
"Formatter" in C

- **See** formatter.c
  - Command-line arguments
    - No args => read from stdin
    - One arg => read from that file
    - Otherwise => error
  - File I/O
  - Wrappers around standard functions
  - *Absence of* regular expressions
    - strtok() instead
"Formatter" in Java

- See `Formatter1.java`
  - Command-line arguments
    - No args => read from stdin
    - One arg => read from that file
    - Otherwise => error
  - File I/O
    - Minimal now; more later
Regular expressions

- string.split()
- Beware: generates empty strings for leading and trailing separators
"Formatter" in Java

- See Formatter2.java
  - Regular expressions
    - Pattern.compile()
    - pattern.matcher()
    - matcher.group()
  - More verbose, but cleaner
"Formatter" in Python

- See `formatter1.py`
  - Command-line arguments
    - "argv" is a global variable
    - No args => read from stdin
    - One arg => read from that file
    - Otherwise => error
  - File I/O
  - Raw strings
  - Strings as objects
    - "len(seq)" calls "seq.__len__()"
"Formatter" in Python

- Regular expressions
  - `re.compile()`
  - `re.split()`
  - Beware: generates empty strings for leading and trailing separators
"Formatter" in Python

- See `formatter2.py`
  - Regular expressions
    - `re.compile()`
    - `re.findall()`
  - Cleaner
- Generalizing
  - Regular expressions...
Regular Expressions

- Used widely
  - Java (string manipulation)
  - Python (string manipulation)
  - Unix grep command (file searching)
  - Bash shell (filename wildcards)
  - SQL "like" clauses (querying databases)
    - See upcoming Databases lectures
  - ...

### Regular Expression Examples

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>thing</td>
<td>thing anywhere in string</td>
</tr>
<tr>
<td>^thing</td>
<td>thing at beginning of string</td>
</tr>
<tr>
<td>thing$</td>
<td>thing at end of string</td>
</tr>
<tr>
<td>^thing$</td>
<td>string that contains only thing</td>
</tr>
<tr>
<td>^</td>
<td>any string, even empty</td>
</tr>
<tr>
<td>^$</td>
<td>empty string</td>
</tr>
<tr>
<td>.</td>
<td>non-empty, i.e. the first char in string</td>
</tr>
<tr>
<td>thing.$</td>
<td>thing plus any char at end of string</td>
</tr>
<tr>
<td>thing.$</td>
<td>thing. at end of string</td>
</tr>
<tr>
<td>\thing\</td>
<td>\thing\ anywhere in string</td>
</tr>
<tr>
<td>[tT]hing</td>
<td>thing or Thing anywhere in string</td>
</tr>
<tr>
<td>thing[0-9]</td>
<td>thing followed by one digit</td>
</tr>
<tr>
<td>thing[^0-9]</td>
<td>thing followed by a non-digit</td>
</tr>
<tr>
<td>thing[0-9][^0-9]</td>
<td>thing followed by digit, then non-digit</td>
</tr>
<tr>
<td>thing1.*thing2</td>
<td>thing1 then any (or no) text then thing2</td>
</tr>
<tr>
<td>^thing1.*thing2$</td>
<td>thing1 at beginning and thing2 at end</td>
</tr>
</tbody>
</table>

Thanks to Dr. Brian Kernighan
Regular Expression Examples

What do these match?

- a.*e.*i.*o.*u
  - Try with grep command and /usr/share/dict/words file

- ^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeiou]*u[^aeiou]*$
  - Try with grep command and /usr/share/dict/words file

Thanks to Dr. Brian Kernighan
Regular Exprs in Java/Python

- Implementations vary
  - See *Mastering Regular Expressions* (Jeffrey Friedl) book
    - ~500 pages!
  - Common to Java and Python (and grep)…
### Regular Expressions in Java/Python

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>the character X, except for metacharacters</td>
</tr>
<tr>
<td>\X</td>
<td>the character X, where X is a metacharacter</td>
</tr>
<tr>
<td>.</td>
<td>any character except \n</td>
</tr>
<tr>
<td>^</td>
<td>start of string</td>
</tr>
<tr>
<td>$</td>
<td>end of string</td>
</tr>
<tr>
<td>XY</td>
<td>X followed by Y</td>
</tr>
<tr>
<td>X*</td>
<td>zero or more cases of X (X*? is the same, but non-greedy)</td>
</tr>
<tr>
<td>X+</td>
<td>one or more cases of X (X+? is the same, but non-greedy)</td>
</tr>
<tr>
<td>X?</td>
<td>zero or one case of X (X?? is the same, but non-greedy)</td>
</tr>
<tr>
<td>[...]</td>
<td>any one of ...</td>
</tr>
<tr>
<td>[^...]</td>
<td>any character other than ...</td>
</tr>
<tr>
<td>[X-Y]</td>
<td>any character in the range X through Y</td>
</tr>
<tr>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>(...)</td>
<td>..., and indicates a group</td>
</tr>
</tbody>
</table>

Precedence: * + ? higher than concatenation, which is higher than |
### RE Matches

<table>
<thead>
<tr>
<th>RE</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>\t</td>
<td>tab</td>
</tr>
<tr>
<td>\v</td>
<td>vertical tab</td>
</tr>
<tr>
<td>\n</td>
<td>newline</td>
</tr>
<tr>
<td>\r</td>
<td>return</td>
</tr>
<tr>
<td>\f</td>
<td>form feed</td>
</tr>
<tr>
<td>\a</td>
<td>alert</td>
</tr>
<tr>
<td>\e</td>
<td>escape</td>
</tr>
<tr>
<td>\</td>
<td>backslash</td>
</tr>
<tr>
<td>\A</td>
<td>empty string at start of given string</td>
</tr>
<tr>
<td>\b</td>
<td>empty string, but only at start or end of a word</td>
</tr>
<tr>
<td>\B</td>
<td>empty string, but not at start or end of a word</td>
</tr>
<tr>
<td>\d</td>
<td>a digit</td>
</tr>
<tr>
<td>\D</td>
<td>a non-digit</td>
</tr>
<tr>
<td>\s</td>
<td>a white space character, that is, [\t\n\r\f\v]</td>
</tr>
<tr>
<td>\S</td>
<td>a non-white space character</td>
</tr>
<tr>
<td>\w</td>
<td>an alphanumeric character, that is, [a-zA-Z0-9_]</td>
</tr>
<tr>
<td>\W</td>
<td>a non-alphanumeric character</td>
</tr>
<tr>
<td>\Z</td>
<td>the empty string at the end of the given string</td>
</tr>
</tbody>
</table>
More Reg Exp Examples

- What kinds of strings do these regular expressions match?
  - /*.*?\*/ (use with DOTALL)
    - Why the question mark?
    - Why DOTALL?

- Editorial: Regular expressions are write-only!!!
Some theory:

- Regular expressions have the same power as deterministic finite state automata (DFAs)
- A regular expression defines a regular language
- A DFA also defines a regular language
Regular Exprs and DFAs

ab*c

a
b

c

a[bc]+d

a
b,c
b,c

a
b,c

d

red circle
Summary

- We have covered:
  - In C, Java, and Python...
  - Classes, objects, object references
  - Command-line arguments
  - Files
  - Regular expressions