

Programming Language Basics

Lecture Outline & Notes

Overview

1. History & Background
2. Basic Program structure
 - a. How an operating system runs a program
 - i. Machine code
 - ii. OS-specific commands to setup memory
 - iii. Main entry point
 - b. Sequential (linear) control flow
3. Syntax Rules
 - a. Individual commands (lines) and line terminators
 - b. Blocks (curly brackets)
 - c. Reserved words & Special symbols
 - d. Organization
 - i. Code separation and modularity
 - ii. Standard libraries
 - e. Comments
4. Setup and getting started: Hello World!
5. Variables

Problem solving involves manipulating data; variables are a means by which we hold data

 - a. Literals
 - i. Numeric
 - ii. Character
 - iii. Strings
 1. Escape sequences (\t, \n, etc.)
 - b. Variables
 - i. Primitive Data Types
 1. Byte representations & limitations
 - a. Integer: sign, magnitude
 - b. Floating point: mantissa, exponent, sign
 - c. Character: ASCII set
 - d. Boolean types (true or false)
 - ii. User defined types (more depth later)
 - iii. Declaration & Scope
 1. Globally scoped variables: bad practice

- Pollutes the name space (if multiple libraries or files declare the same global variable: conflict)
- Not predictable: if everything can access a variable, anything can change it; test coverage now extends to everything!

iv. Identifiers

1. Naming rules
2. Naming conventions
 - a. Camel casing
 - b. CAPS_UNDER_SCORES for constants

v. Assignments

1. Assignment operator
2. Memory & storage

6. Operators & Expressions

- a. Standard arithmetic operators: +, -, *, /
- b. Integer remainder division
- c. Unary operators (negations: -, !), increment/decrement (pre and postfix)
- d. Logical operators (&&, ||, more later)
- e. Mixed types
 - i. Casting rules
 - ii. Integer division & truncation
- f. Type casting
- g. Precedence rules: similar to arithmetic precedence rules

7. Standard Input & Output

- a. Print-formatted standard
 - i. Standard placeholders: %c, %Nd, %N.Mf/%lf
- b. Interactive vs command line argument input

8. Comments

- a. Purpose
 - General documentation
 - Comments provide the *what* and the *why*: a high-level description of what the program/function/block does and why (its purpose or why/how it should be used)
 - “Self-documenting” code is well-written code that communicates *how* it is done (the details that are not appropriate for comments)
 - Best strategy: write comments *first* (this function does blah), *then* write the actual code (allows you to also design a strategy/algorithm!)

9. Coding Styles

10. Compiling & Executing

- a. Hello World program
- b. Debugging
 - i. Syntax errors

- ii. Runtime errors
- iii. Logic Errors

Overview – C

1. History & Background

a. History

- i. Developed by Dennis Ritchie while at AT&T Bell Labs 1969 - 1972
- ii. “C” because features were derived from “B” (BCPL-Basic Combined Programming Language)
- iii. Closely tied to Unix
- iv. 1978: K&R (Kernighan) C
- v. 1979 – 1983: OOP C++ developed by Bjarne Stoustrup
- vi. 1989: ANSI C (C89)
- vii. 1999: C99
- viii. 2011: C11
- ix. Today:
 1. Many other (interpreted) languages are written in C (Perl, PHP, Python, Matlab)
 2. Influenced numerous other languages (C-style languages)
 3. Extremely popular, drives a plurality of open and closed source projects

b. Language Basics

- i. Used extensively in “systems programming”: OS Kernels, embedded systems,
- ii. Portable, stable, & efficient (close to the OS)
- iii. Imperative (or structured, procedural) style language: program’s state is changed through a series of sequential statements and function executions.

2. Basic structure

a. Preprocessor directives

- i. Global constants:
`#define PI 3.14159`
Essentially a macro (compiler cut and pastes)
- ii. Inclusion of standard libraries:
`#include<stdlib.h>`
- iii. Conditional compilation and compiler directives `#define`, `#if`, `#endif`, `#ifdef`

b. Main entry point

- i. The main function is always the starting point for any compiled program:
`int main(int argc, char **argv)`

c. Sequential (linear) control flow: from the entry point, commands are executed in a sequential manner unless interrupted by conditionals, loops, or function calls

3. Syntax Rules

- a. All command lines terminated with a semi-colon
- b. Blocks
 - i. Delimited by curly brackets
 - ii. Blocks can contain sub-blocks
 - iii. Best practice/style: use proper indentation
- c. Reserved words & Special symbols

- i. Reserved words: double, int, if, else, void, return, etc.
 - ii. Standard identifiers (should **not** be redefined): printf, scanf
- d. Organization
 - i. Code separation and modularity
 - ii. Standard libraries (use #include<...>
 - 1. stdlib.h
 - 2. stdio.h
 - 3. math.h
 - a. Math library is not a standard library and needs to be explicitly included at compile time (-lm flag)

4. Variables

a. Literals

i. Numeric

1. Base-10:
int a = 1234;
2. Base-2 (binary)
int b = 0b10011010010;
3. Base-16 (hex)
int c = 0x4D2;
4. Floating point numbers: -10.43
5. Scientific notation:
3.14;
3.14e0;
3.14E0;
314E-2;
.314E1;

ii. Characters:

1. A single ASCII character delimited by single quotes
2. The ASCII text table: 'A' = 65, '0' = 30, etc.
3. Characters and numbers are equivalent, so:
char myFirstInit = 67;
is valid

iii. Strings

1. Delimited by double quotes
2. Escape sequences & special characters (\t, \n, etc.)

b. Variables

i. Primitive Data Types

1. int
 - a. 32* bit signed, 2s-complement integer (negative numbers are complemented)
 - b. ANSI Standard: minimum of 16 bits (-32768 – 32767)
 - c. Standard 32 bit range: - 2147483648 ~ +2147483647

2. float: 32 bit floating point number: 7 decimal digits of precision
 3. double: 64 bit floating point number: 19 decimal digits of precision
 - a. Mantissa, exponent sign (more later)
 - b. Cannot fully represent real numbers
 - c. Potential problems with loss of precision in floating point operations (more later)
 4. char: a single byte containing a value equal to the character's ASCII value
 5. Boolean type: none! Use an integer (0 = false, *anything* else = true)
- ii. Declaration & Scope
1. All variables must be declared before they can be used
 2. Declaration involves: specifying its type and name (identifier)
 3. Assignment operator: =
 - a. *Not* an algebraic operator
 - b. Not an equality test
 - c. Place the value of the expression on the Right hand side into the variable on the left hand side:
`a = 10;`
`b = (10 + 3);`
 - d. Default values: undefined!
 - e. Memory & storage discussion
 4. Syntactic sugar:
 - a. Optional declaration/assignment:
`int a = 10;`
`double pi = 3.14;`
 - b. Multiple variable declaration:
`int a, b, c;`
 - c. Multiple declaration, assignment:
`int a = 10, b, c = 20;`
 5. Scope: variable is only valid in the block that it was declared in; outside the block it goes out of scope and is lost
- iii. Identifiers
1. Naming rules
 - a. Must begin with [a-zA-Z] (avoid `_`: indicates a "private" variable by convention)
 - b. May contain [a-zA-Z0-9_]
 2. Naming conventions
 - a. Old C convention: lower_case_underscore
 - b. Modern convention: lowerCamelCasing
 - c. Avoid: Hungarian notation (building the type into the name)
 - d. CAPS_UNDER_SCORES for macros/constants

5. Operators & Expressions

- a. Standard arithmetic operators: +, -, *, /
 - b. Integer remainder division: %
 - c. Unary operators (negations: -, !), increment/decrement (pre and postfix)
 - d. Logical operators (&&, ||, more later)
 - e. Mixed types
 - i. Casting rules
 - ii. Integer division & truncation
 - f. Type casting
 - g. Precedence rules: similar to arithmetic precedence rules
6. Standard Input & Output
- a. Print-formatted standard
 - i. printf (in the stdio.h library)
 - ii. Usage: printf("format", var arg list);
 - b. Interactive input: scanf("format", &var, &arg, &list);
 - i. Crucial difference: must use ampersands!
 - ii. Crucial difference: for reaching doubles, use %lf
 - iii. Examples
 - c. Command line input: main(int argc, char *argv)
 - i. argc: number of arguments (arg count) *including* the executable
 - ii. arguments delimited by whitespace, may be encapsulated with double quotes
 - iii. arguments available as strings (more later): argv[0], argv[1], etc.
 - iv. conversion functions: atoi, atof
7. Comments
- a. Syntax
 - i. Single line (//this is a comment)
 - ii. Multi line (/* this is a comment that may span multiple lines */)
 - iii. Cannot nest multiline comments
 - b. Usage
 - i. Code documentation (use sparingly—comment programs, functions, copyright, etc.)
 - ii. Tips: <http://www.devtopics.com/13-tips-to-comment-your-code/>
8. Coding Styles
9. Compiling & Executing
- a. Hello World program
 - b. Debugging
 - i. Syntax errors
 - ii. Runtime errors
 - iii. Logic Errors
10. Exercises

Overview – Java

1. Java History & Overview

a. History

- i. Developed by James Gosling, Sun Microsystems, 1995
- ii. Five Principles:
 1. Simple, Object-oriented, familiar
 2. Robust and secure
 3. Architecture-neutral and portable
 4. High performance
 5. Interpreted, threaded, and dynamic
- iii. Versions
 1. 1.0 – 1996
 2. 1.1 – 1997 introduced JDBC, inner classes, reflection
 3. 1.2 – 1998 Collections framework
 4. 1.3 – 2000 JNDI, HotSpot JVM
 5. 1.4 – 2002 Library improvements
 6. 1.5 – 2004 Generics introduced, enhanced-for loop (foreach loop), concurrency utilities
 7. SE6 – 2006 JVM improvements (synch, compiler, garbage collection), Update 26 (June 7, 2011)
 8. 1.7 – July 2011
- iv. 2009/10: Oracle purchases Sun in order to sue Google
- v. Summer 2012: Oracle loses

b. Key Aspects

- i. “C-style syntax”: semicolons, bracket blocks, identifiers
- ii. Object Oriented (everything is a class, except primitives)
- iii. No memory management (built-in garbage collection)
- iv. Portable across systems: Write once, run anywhere
- v. Java Virtual Machine (performance hit, but benefits outweigh, not much of an issue anymore)

2. Basic structure

- a. Everything is a class or must be contained in a class:

```
public class MyClass { ...
```
- b. The source file must have the same name as the class: `MyClass.java`
- c. Classes are organized in a package hierarchy (essentially and literally a directory structure):

```
package un1.cse.cse155h;
```
- d. Libraries may be imported using the import key word:

```
import java.lang.Math;
import java.util.Arrays;
```
- e. Any class may be executable if it has a main method:

```
public static void main(String args[]) { ...
```


3. Syntax Rules

- a. Very similar

4. Variables

- a. Literals: mostly the same (binary supported in Java 1.7+)
- b. Variables

i. Primitive Data Types

- byte (8 bit signed 2's complement)
- short (16 bit signed 2s complement)
- int (32 bit signed 2s comp)
- long (64 bit signed 2s comp)
- float (32 bit floating point number)
- double (64 bit floating point number)
- boolean (true boolean type: true or false)
- char (16 bit Unicode character!)
- Primitive wrappers: Byte, Integer, Double, Character, etc. provide basic functionality:

- a. `int a = Integer.parseInt("12345")`

- b. Necessary for container/collections classes (more later)

- Strings:
`String s = "Hello World";`
`s = s + "!"; //built-in concatenation!`

ii. Object types: String, Integer, Scanner, etc.

1. Full objects (not primitive types)
2. User defined
3. Variables are actually *references* to the object's location in memory
4. null keywords

iii. Same syntax and rules for declaration, assignment, scope

1. Difference: Everything *does* have a defined default (0, 0.0, false, null)

5. Operators & Expressions

- a. Same operators, precedence rules, truncation, etc.

b. Key differences:

- i. Boolean type exist, so you cannot negate an integer
- ii. Explicit down-casting required (cannot assign a double to an integer)
- iii. Auto boxing/unboxing of object versions of primitives:

```
Integer i = 10;
```

```
int b = i + 5;
```

6. Standard Input & Output

- a. Standard output: System.out

- i. Can only handle strings or string casts; auto casting with string concatenation operator: +

- ii. Examples:

```
System.out.print("Hello \n");  
System.out.println("World!");  
System.out.println("a = "+a); //type mixing
```
- iii. Formatted Print supported:
 - 1.

```
System.out.printf("a = %d", a);  
String s = String.format("%40s", message);  
System.out.println(s);
```

b. Command Line Arguments

- i. `main(String args[])` – an array of strings (more later)
- ii. `args.length` gives number of arguments *not* including the class name!!!
- iii. Similar conversion tools:

```
int a = Integer.parseInt(args[0]);  
double d = Double.parseDouble(args[1]);
```

c. Interactive Input

- i. Standard input: `System.in`
- ii. Make use of the `Scanner` class:

```
Scanner s = new Scanner(System.in);  
System.out.println("Enter an integer: ");  
int a = s.nextInt(); // "blocks" (waits) for the user to enter the input
```
- iii. Careful: exceptions thrown for bad input!

7. Comments

- a. Same Rules
- b. Nice Eclipse feature: javadocs `/**`-enter (formats in nice HTML, full documentation can be generated and distributed)

8. Coding Styles

9. Compiling & Executing

- a. Hello World program
- b. Debugging
 - i. Syntax errors
 - ii. Runtime errors
 - iii. Logic Errors

11. Exercises

- Convert a C program to Java
- Argument demo
- Distance converter