Introduction to Programming: Logic, Variables and Objects

HORT 530

Lecture 7

Instructor: Kranthi Varala

What is a program?

- A set of instructions to the computer that perform a specified task in a specified manner.

- The task of writing a functional, error-free and cohesive set of instructions is called programming.

- Two major components of Programming:
  - Logic – What are my set of instructions?
  - Syntax – How do I convey them to the CPU?

Logic

- Programming logic is an exercise in how to break down a complex real-world problem into a set of mathematical notations.
- Independent of programming language used.
- Divide a complex task into a series of simpler tasks.
- Divide each task into a set of simple, sequential instructions.
- Arrange simple tasks in the most efficient order to accomplish complex task.

Syntax

- Programming syntax is a predetermined set of rules in which the instructions need to be provided.
- Usually unique to each programming language.
- Once the logic is established, convert the instructions into the syntax prescribed.
- Constraints imposed by the syntax may require adjusting the logic.

Programming languages

- Simply put, a language understood by the computer.
- Computers are not truly intelligent (Yet!). So, the instructions need to be extremely detailed and precise.
- At the lowest hardware level, a computer operates on bits (1 and 0), called binary code.
- Modern programming languages are closer to how humans think/talk.

Low level vs. High level languages

- Programming languages range from low level (close to binary) to high level (close to human language).

- Order of languages:
  High-level --> Assembly --> Machine/Binary

- Low-level languages are extremely hard to learn and lack portability, but make optimal use of hardware.

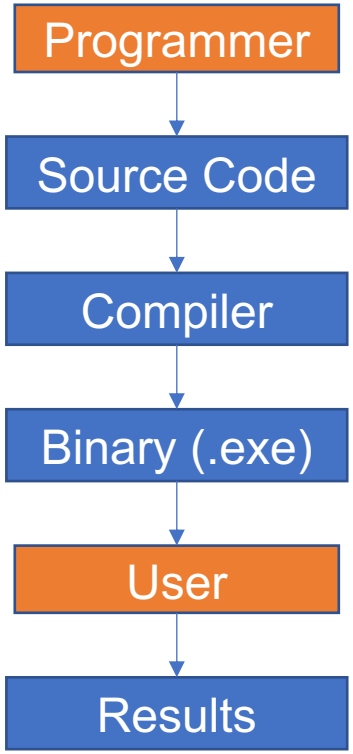
- High-level languages are easier to learn and generalize, but are poorer at using hardware resources.

High-level languages

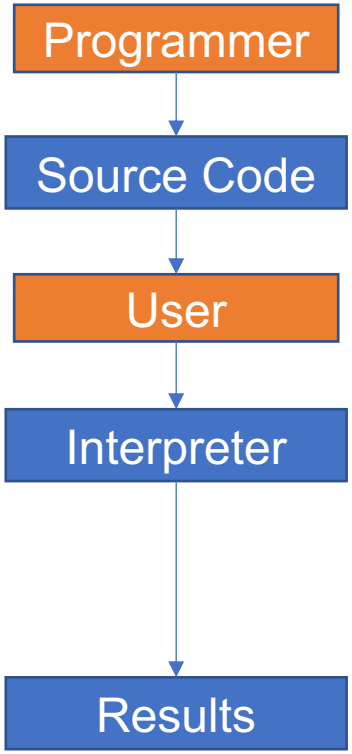
- Common high-level languages are Java, C, C++, Python, Perl etc.
- Code written in a high-level language is interpreted and converted into low-level language by an intermediate layer that is either a compiler or and interpreter.
- The compiler and interpreter differ in the timing of generation of machine code.

Compiled vs Interpreted

- Compiled languages
  E.g., C, C++, Fortran, Java etc.



- Interpreted languages
  E.g., Python, Perl, Ruby, Javascript etc.



Compiler

- Converts entire source code into machine code at the compilation stage.
- Optimizes the source code to use hardware resources better.
- Compilation process can be lengthy and computationally intensive, depending on the source code.
- Compilers can be told to generate different machine codes for different hardware platforms e.g., Mac vs PC vs Linux.
- Optimized machine code is faster, but not portable.

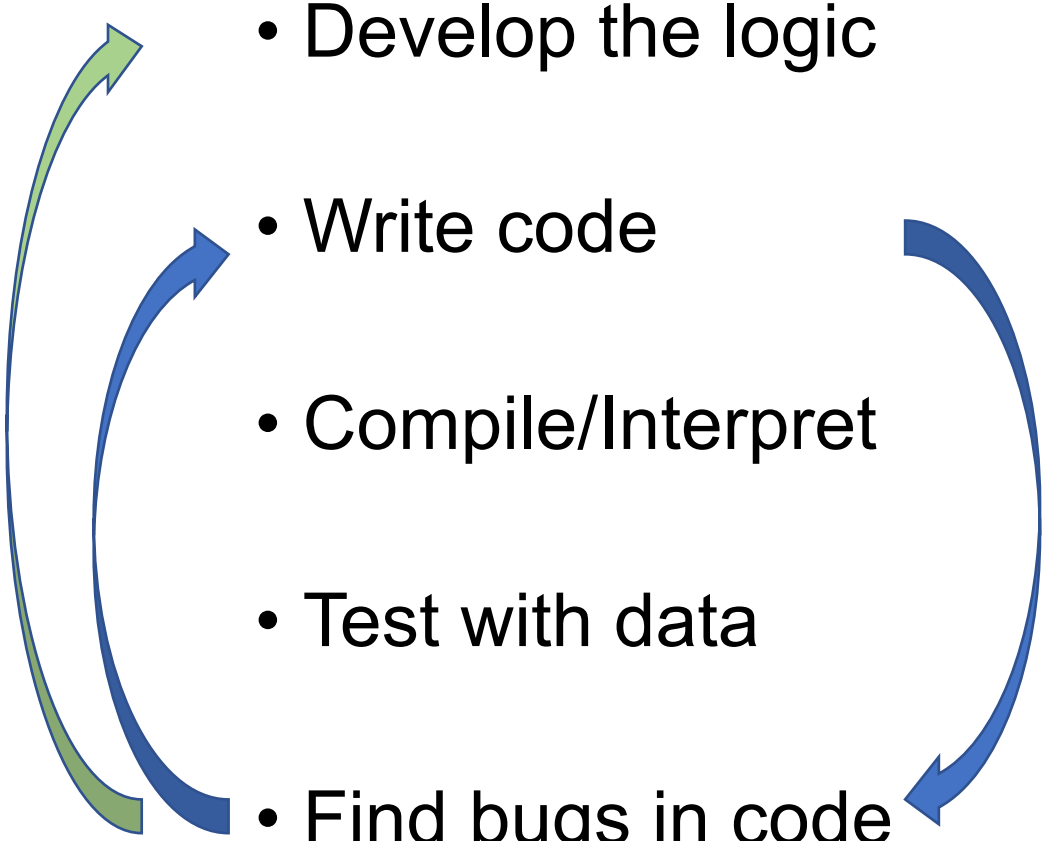
Interpreter

- Interpreter resides on the target machine and accepts source code as input.
- Reads through the code one line at a time and converts instructions in that line to machine code.
- Lacks the optimization step included in compilers and hence program execution is often slower than compiled programs.
- Source code is highly portable, since the interpreter is integrated in the target platform.
- Languages that rely on interpreters are often called “Scripting languages”.

Machine code is the ultimate result

All source code HAS to be eventually converted into machine code

Stages of programming



Pseudocode

- A high level description of the problem and the programmatic solution that helps develop the overall logic and the subdivision of tasks.
- Informal way of writing code that does not worry about the syntax of the programming language.
- Follows a loose set of rules that allow logical grouping of functions.

Pseudocode Example

- Problem: Calculate the average length of all protein sequences in a file.
- PseudoCode (Simplest):
  - Read all Protein sequences
  - Calculate length of each sequence
  - Calculate the mean of all lengths

Pseudocode Example

- Problem: Calculate the average length of all protein sequences.
- PseudoCode (Simple):
  - Open protein sequence file
  - Loop over protein sequences
    - Read each sequence
    - Calculate length of current sequence
    - Add length to sum variable
  - Calculate the mean length by dividing sum/ no. of sequences

Pseudocode Example

- PseudoCode (Detailed):
  - Create mean length variable; set to 0
  - Create No. of sequences variable; set to 0
  - Create sum variable; set to 0
  - Ask user input for name of sequence file
  - Open protein sequence file (if exists)
  - Loop over protein sequences
    - Read each sequence
    - Calculate length of current sequence
    - Add sequence length to sum variable
    - Increment No. of sequences by 1
  - Calculate value of mean as Sum/No. of sequences.
  - Output value of mean

Other examples

- Find all 5 letter words in a file and print only those that start with a vowel.
- Exchange the value of 2 variables (integers) without using a third variable.
- Print the following patterns:

1.
```
*
**
***
****
*****
```

2.
```
    *
   **
  ***
 ****
*****
```

3.
```
    *
   **
  ***
 ****
*****
```

Variables and Objects

- Variables are the basic unit of storage for a program.
- Variables can be created and destroyed.
- At a hardware level, a variable is a reference to a location in memory.
- Programs perform operations on variables and alter or fill in their values.
- Objects are higher level constructs that include one or more variables and the set of operations that work on these variables.
- An object can therefore be considered a more complex variable.

Data types

- All variables are empty when created.
- Since variables have no implicit value, they can store many different forms of values.
- Example values of variable:
  - Var = TRUE #Boolean
  - Var = 523243 #Integer
  - Var = 42.2524 #Float
  - Var = Hi there #String
  - Var2 = Reference to Var #Reference

Implicit (Weak) vs. Explicit (Strong) data types

- Some languages require each variable to be explicitly defined as a single data type, e.g. integer, float, string etc.
- Explicit data typing prevents nonsensical operations such as multiplying two strings.
- Compiled languages, such as C, Java etc., typically use explicit data types, since it allows improved optimization.
- Other languages allow all variables to be any one of the implicit data types.
- With implicit data typing, a variable can change its data type within the same program.
- Scripting languages, such as Perl often use implicit (i.e., weak) data types.

Documentation

- Always document your code with as much detail as possible.
- For each variable, when created, write a comment explaining the intended use of the variable.
- For each loop, write a comment describing what the loop is expected to iterate over.
- For each function, write a comment explaining the expected input and expected output of function.
- For each stage of the program, write a long comment describing the current stage, expected input and outcome of this stage.

Summary

- Programming languages can be either compiled or interpreted.
- Compiled code is faster but non-portable, while interpreted code is very portable.
- Programming has two stages: 1. Establishing the logic and 2. Writing the code
- Pseudocode is a great way to establish logic before getting bogged down by syntax.
- The basic data units in programming are variables and objects.
- Languages may have strong or weak typing for variables.