Purpose of a high-level programming language

- To take a complex task and break it into smaller, more manageable subtasks
- To translate the requirements of a program into a style that can be (eventually) understood and executed by a computer
- To allow a programmer to express computer tasks in a more natural language

Programming languages are just like any other language!

- Syntax ==> “grammar”
- Statements ==> “sentences”
- Algorithm ==> “content”

Algorithm

- “Recipe” or procedure
- Explains how the program is to accomplish the task it's commissioned for
- Often written out “logically”, in English, first, and then translated into code
Types of programming languages

- Compiled
- Interpreted

Compiled language

- Examples: C, C++, Java
- Translated from high-level language to machine code, then machine code is executed
- Compiler: software program that does this translation
  - specific to a particular architecture, since it has to know that architecture's machine code!

Interpreted language

- Examples: Perl, PHP, Javascript, Java
- Programs can be run “as is”, without compiling
- A special program, called an “interpreter”, translates the statements on the fly

Note: Java is both compiled and interpreted!

- compiled into bytecode (intermediate machine code that can run on all machines)
- bytecode is then interpreted
  - Java Virtual Machine: interpreter program
  - JVMs are specific to an architecture
Categories of programming languages

- Procedural (imperative)
- Object-oriented
- Functional
- Logic

Procedural language

- Programs are written as a series of “steps” towards a solution
- Linear and hierarchical
- Examples: C, Fortran, BASIC

Object-oriented language

- Programs are written in terms of interacting objects
- Each object has certain data and actions that it can share with other objects
- Modular
- Examples: Java, C++, the .Net languages (C#, etc.)

Key programming concepts

- Program body, declaration
- Variable assignment
- Conditional statements
- Looping construct
- Input/output statements
Operating system

- “Manager” of a computer
- Set of software programs
- Interacts with other software programs (applications, system programs) and with the computer hardware
- Manages computer resources (memory, processor, I/O)
- Provides an interface to allow a user to interact with a computer

History of operating systems

Single-operator --> batch processing --> timesharing --> “multiprogramming”

Main tasks of an operating system

- Memory management
- Process management
- CPU scheduling

Memory management

- How to store one or more programs (and all associated data) in memory (RAM)
- Paged memory
  - divide RAM into chunks, or “pages” of a set size
  - programs/data loaded into pages wherever there's room
    - can be out of order
  - program is executed by finding the next “page” and reading it
  - “page swap”: transferring a program from secondary storage to main memory
Process management

- Keeps track of which programs are currently executing, how far along they are, the associated data, etc.

CPU scheduling

- Determines which process gets to run at any given time
- Processes that are waiting for I/O or some other resource will not get access to the processor until they enter the ready state
- Some processes can also be preempted (forced to give up the processor) for other reasons

Scheduling algorithms

- First-come, first-served (FCFS)
  - processes gain access to the CPU in the order they arrive
- Round robin (RR)
  - processes are given a small slice of time in which to execute, after which they go back to waiting for their next turn
  - “most fair” form of scheduling