Introduction to Programming and Problem Solving

Parts of a Computer System:

- Hardware: Electronic Devices
- Software: Instructions and Computer Programs

Hardware

- Input : Keyboard, Mouse
- System unit:
  - Random Access Memory (RAM)
  - Central Processing Unit (CPU)
- Output: Monitor, Printer
- Secondary Storage: Disk Drive

Software

- Instructions for the hardware.
  - Actions to be performed
- A set of instructions is called a program.
  - Driving force behind the computer
  - Without a program – What is a computer?
    - Collection of Useless Hardware
- 2 purposes:
  - Tell the computer what to do
  - Tell other people what we want the computer to do.

CPU

- The central processing unit (CPU)
- The “brain” of a computer
- Retrieves instructions from memory and executes them.

Memory (RAM)

- Stores data and program instructions for CPU to execute
  - A program and its data must be brought to memory before they can be executed
- Stores intermediate and final results of processing.
- Volatile: Contents are erased when computer is turned off or reset.
- A memory unit is an ordered sequence of bytes, each holds eight bits. A byte is the minimum storage unit. No two data can share or split the same byte.

Storage Devices

- Hard Drives, CDs/DVDs, Flash Drives, etc.
- Non-Volatile or Permanent Storage
- Programs and data are permanently stored on storage devices and are moved to memory when the computer actually uses them.

Computer Language

- Digital devices have two stable states, which are referred to as zero and one by convention
- The binary number system has two digits, 0 and 1. A single digit (0 or 1) is called a *bit*, short for *bi*nary dig*it*. A byte is made up of 8 bits.
- Binary Language: Data and instructions (numbers, characters, strings, etc.) are encoded as binary numbers - a series of bits (one or more bytes made up of zeros and ones)

Computer Language (cont.)

- Encoding and decoding of data into binary is performed automatically by the system based on the encoding scheme
- Encoding schemes
  - Numeric Data: Encoded as binary numbers
  - Non-Numeric Data: Encoded as binary numbers using representative code
    - ASCII – 1 byte per character
    - Unicode – 2 bytes per character

Binary Number System

- Decimal
  - Base 10, ten digits (0-9)
  - The position (place) values are integral powers of 10: 10^0(ones), 10^1(tens), 10^2(hundreds), 10^3(thousands)…
  - n decimal digits - 10^n unique values
- Binary
  - Base 2, two digits (0-1)
  - The position (place) values are integral powers of 2: 2^0(1), 2^1(2), 2^2(4), 2^3(8), 2^4(16), 2^5(32), 2^6(64)…
  - n binary digits - 2^n unique values

ASCII Table

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	so	d;e	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	“	#	$	%	&	‘
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	del		

Programming Languages

- Computers can not use human languages, and programming in the binary language of computers is a very difficult, tedious process
- Therefore, most programs are written using a programming language and are converted to the binary language used by the computer
- Three major categories of prog languages:
  - Machine Language
  - Assembly Language
  - High level Language

Machine Language

- Natural language of a particular computer
- Primitive instructions built into every computer
- The instructions are in the form of binary code
- Any other types of languages must be translated down to this level

Assembly Languages

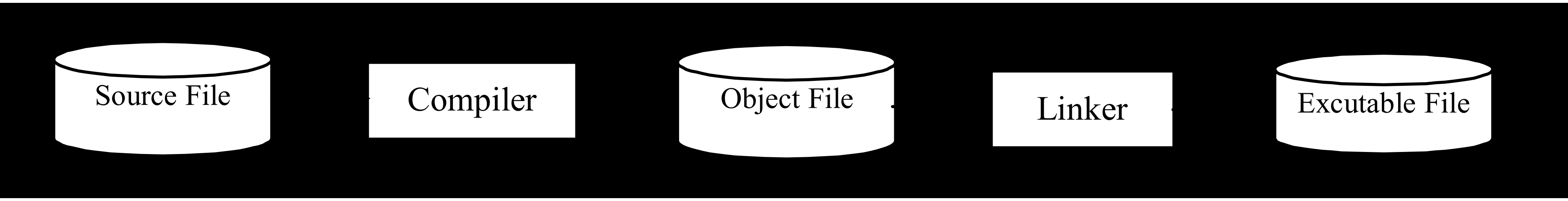
- English-like Abbreviations used for operations (Load R1, R8)
- Assembly languages were developed to make programming easier
- The computer cannot understand assembly language - a program called assembler is used to convert assembly language programs into machine code

High Level Languages

- English-like and easy to learn and program
- Common mathematical notation
  - Total Cost = Price + Tax;
  - area = 5 * 5 * 3.1415;
- Java, C, C++, FORTRAN, VISUAL BASIC, PASCAL

Compiling Source Code

A program written in a high-level language is called a *source program (or source code)*. Since a computer cannot understand a source program. Program called a *compiler* is used to translate the source program into a machine language program called an *object program*. The object program is often then linked with other supporting library code before the object can be executed on the machine.



Compiling Java Source Code

You can port a source program to any machine with appropriate compilers. The source program must be recompiled, however, because the object program can only run on a specific machine. Nowadays computers are networked to work together. Java was designed to run object programs on any platform. With Java, you write the program once, and compile the source program into a special type of object code, known as *bytecode*. The bytecode can then run on any computer with a Java Virtual Machine. Java Virtual Machine is a software that interprets Java bytecode.

Programming

- Programming – the creation of an ordered set of instructions to solve a problem with a computer.
- Only about 100 instructions that the computer understands - Different programs will just use these instructions in different orders and combinations.
- The most valuable part of learning to program is learning how to think about arranging the sequence of instructions to solve the problem or carry out the task

Programming Fundamentals: Putting the Instructions Together

- Sequential Processing
  - A List of Instructions
- Conditional Execution
  - Ifs
- Repetition
  - Looping / Repeating
- Stepwise Refinement / Top-Down Design
  - Breaking Things into Smaller Pieces
- Calling Methods / Functions / Procedures / Subroutines
  - Calling a segment of code located elsewhere
  - Reuse of previously coded code segment

Methods of Programming

- Procedural
  - Defining set of steps to transform inputs into outputs
  - Translating steps into code
  - Constructed as a set of procedures
  - Each procedure is a set of instructions
- Object-Oriented
  - Defining/utilizing objects to represent real-world entities that work together to solve problem
  - Basic O-O Programming Components
    - Class
    - Object/Instance
    - Properties
    - Methods

Object Oriented Programming

- Class
  - Specifies the definition of a particular kind of object
    - Its Characteristics : **Properties** (or Attributes)
    - Its Behaviors: **Methods**
  - Used as blueprint / template to create objects of that type
- Object/Instance
  - A specific instance of a class – an object created using the Class Definition
  - All specific instances of the same class share the same definition
    - Same Properties – can be modified
    - Same Methods – can be modified

Class and Object Example

CLASS (ALL DOGS)

- All Instances of Class Dog Have
  - Properties
    - Name
    - Breed
    - Weight
    - Color
  - Methods
    - Walk
    - Bark
    - Jump

OBJECT (ONE DOG)

- A particular Instance Could Have
  - Properties
    - Name -Spot
    - Breed - Mutt
    - Weight - 10 pounds
    - Color - Black
  - Methods
    - Walk
    - Bark
    - Jump

    (these can also be modified to fit a particular dog)

Problem Solving

- The process of defining a problem, searching for relevant information and resources about the problem, and of discovering, designing, and evaluating the solutions for further opportunities. Includes:
  - Finding an Answer to a Question
  - Figuring out how to Perform a Task
  - Figure out how to Make Things Work
- Not enough to know a particular programming language… Must be able to problem solve…
- Very desirable to be a good Problem Solver in any CIS discipline.

Polya's 4 Steps of Problem Solving

- U – Understand the Problem
- D – Devise a Good Plan to Solve
- I – Implement the Plan
- E – Evaluate the Solution

Example:
Solving Math Word Problem

- Read the Problem: Understand the description of problem or scenario, identifying the knowns and unknowns
- Decide how to go about solving the problem: Determine what steps need to be taken to reach the solution
- Solve the Problem: Write the solution
- Test the Answer: Make sure the answer is correct

Solving Computing Problems

- In general, when we solve a computing problem we are taking some **input**s, **process**ing (performing some actions on) the inputs, and then **output**ting the solution or results.
- This is the classic view of computer programming – computation as calculation
- Polya's steps (UDIE) can be very effective when applied to solving computing problems

Applying Polya's Problem Solving to Programming
Step 1 - Understand the Problem

- What is the Problem to be solved? What is the unknown? What is the condition? What is the data? What is needed to solve the problem? What actions need to take place?
- Identify the **inputs** and **outputs**
- Identify the **processes** needed to produce the **outputs** from the given **inputs**
- Draw a figure. Introduce suitable notation.
- Isolate Principle parts of the problem.

Applying Polya's Problem Solving to Programming
Step 2 - Devise a Plan

- Find connections between the knowns and unknowns.
- Simplify: Break the problem into smaller sub-problems
- Design a solution
- Make a plan or list of actions to implement the solution
  - Algorithm / Flowchart / Psuedocode

Applying Polya's Problem Solving to Programming
Step 2 - Devise a Plan (cont.)

- Algorithm
  - A FINITE set of clear, executable steps that will eventually terminate to produce the desired outcome
  - Logical design used to solve problems – usually a list of actions required to perform task
- Pseudocode
  - Written like program code but more "English Like" and doesn't have to conform to language syntax
- Flowchart
  - Diagram that visually represents the steps to be performed to arrive at solution.

Applying Polya's Problem Solving to Programming

Step 3 - Implement the Plan

- Implement in a Programming Language
- Carry out the plan checking the preliminary results at each step.
- Code A Little Test A lot

Applying Polya's Problem Solving to Programming

Step 4 - Evaluate the Solution

- Run the Code
- Check results repeatedly and thoroughly
  - Use numerous test cases or data sets
  - Use highly varied test case, including expected as well as and unexpected cases
- Look for new solutions
  - Is there a better, easier, or more efficient solution
- Can other problems be solved using these techniques?

Summary

- U - Read the Problem Statement
  - Identify the inputs, outputs, and processes
- D - Decide how to Solve the Problem
  - Create an Algorithm / Flowchart / Psuedocode
- I - Program the Code
  - Implement in Programming Language
- E - Test the Solution
  - Run the Code using numerous, varied test cases

Practice

Using Polya's first 2 steps, understand and devise a solution to the following problem:

- Determine the week's earnings for an employee from the hourly pay rate and hours worked for the week. Report the gross earnings (including overtime earnings) for the week.
- Definitions:
  - Overtime hours = hours over 40
  - Overtime pay rate = 1.5 * reg pay rate

More Practice

Using Polya's first 2 steps, understand and devise a solution to the following problem:

Determine the week's earnings for an employee from the hourly pay rate, total hours for the week and tax rate. Report the number of reg hours, overtime hours, gross reg earnings, gross overtime earnings, tax witheld, and total net earnings for the week.

Definitions:

- Overtime hours = hours over 40
- Overtime pay rate = 1.5 * reg pay rate