Overview of Computers & Programming

Computers

- Computers receive, store, process, and output information.
- Computer can deal with numbers, text, images, graphics, and sound.
- Computers are worthless without programming.
- Programming Languages allow us to write programs that tell the computer what to do and thus provide a way to communicate with computers.
- Programs are then converted to machine language (0 and 1) so the computer can understand it.

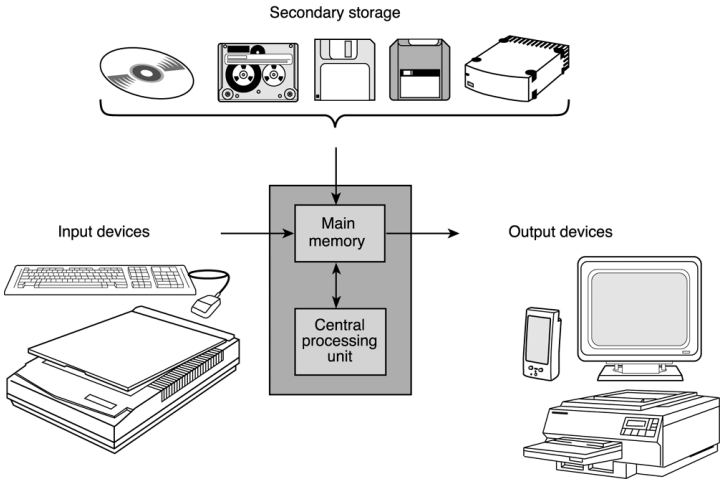
Hardware & Software

- Hardware is the equipment used to perform the necessary computations.
  - i.e. CPU, monitor, keyboard, mouse, printer, speakers etc.
- Software consists of the programs that enable us to solve problems with a computer by providing it with a list of instructions to follow
  - i.e. Word, Internet Explorer, Linux, Windows etc.

Computer Hardware

- Main Memory
  - RAM - Random Access Memory - Memory that can be accessed in any order (as opposed to sequential access memory), volatile.
  - ROM - Read Only Memory - Memory that cannot be written to, no-volatile.
- Secondary Memory - Hard disks, floppy disks, zip disks, CDs and DVDs.
- Central Processing Unit - Coordinates all computer operations and perform arithmetic and logical operations on data.
- Input/Output Devices - Monitor, printer, keyboard, & mouse.
- Computer Networks – Computers that are linked together can communicate with each other. WAN, LAN, MAN, Wireless-LAN.

Components of a Computer



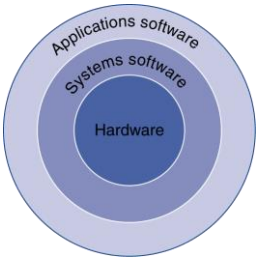
Memory

- Memory Cell (MC) – An individual storage location in memory.
- Address of a MC- the relative position of a memory cell in the main memory.
- Content of a MC – Information stored in the memory cell. e.g Program instructions or data.
  - Every memory cell has content, whether we know it or not.
- Bit – The name comes from binary digit. It is either a 0 or 1.
- Byte - A memory cell is actually a grouping of smaller units called bytes. A byte is made up of 8 bits.
  - This is about the amount of storage required to store a single character, such as the letter H.

Computer Software

- Operating System - controls the interaction between machine and user. Example: Windows, Unix, Dos etc.
  - Communicate with computer user.
  - Manage memory.
  - Collect input/Display output.
  - Read/Write data.
- Application Software - developed to assist a computer user in accomplishing specific tasks. Example: Word, Excel, Internet Explorer.

Below Your Program



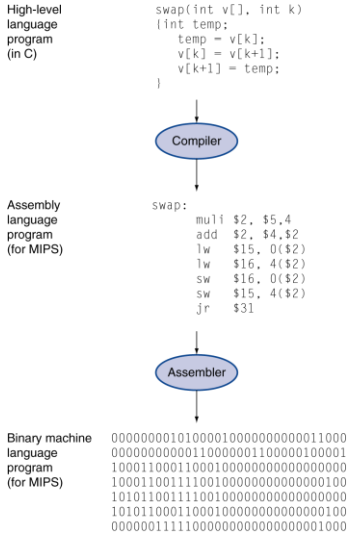
- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Computer Languages

- Machine Language – A collection of binary numbers
  - Not standardized. There is a different machine language for every processor family.
- Assembly Language - mnemonic codes that corresponds to machine language instructions.
  - Low level: Very close to the actual machine language.
- High-level Languages - Combine algebraic expressions and symbols from English
  - High Level **:** Very far away from the actual machine language
  - For example: Fortran, Cobol, C, Prolog, Pascal, C#, Perl, Java.

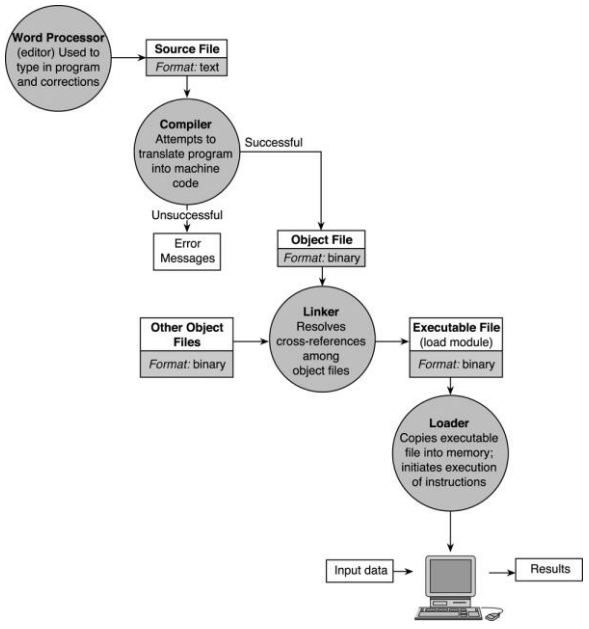
Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



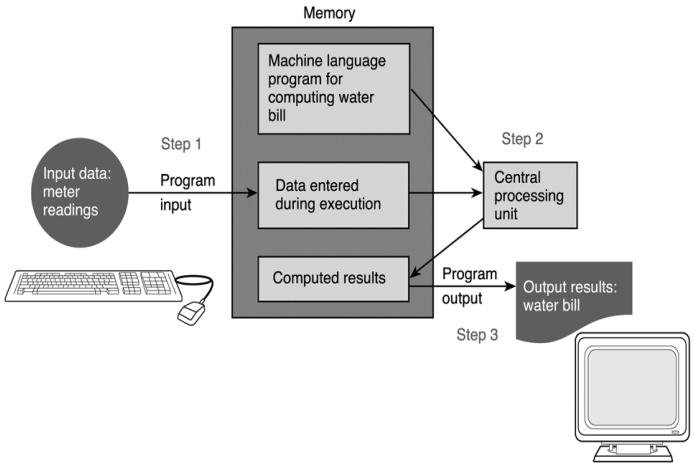
Compiler

- Compilation is the process of translating the source code (high-level) into executable code (machine level).
- Source file - A file containing the program code
  - A Compiler turns the Source File into an Object File
- Object file - a file containing machine language instructions
  - A Linker turns the Object File into an Executable
- Integrated Development Environment (IDE) - a program that combines simple word processing with a compiler, linker, loader, and often other development tools
  - For example, Eclipse or Visual Studio



Editing, Translating, and Running a High-Level Language Program

Flow of Information During Program Execution

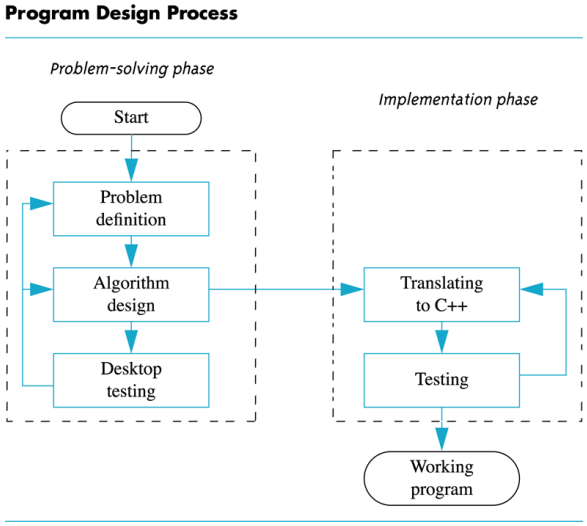


Software Development Method

1. Specify **problem** requirements
2. **Analyze** the problem
3. **Design** the algorithm to solve the problem
4. **Implement** the algorithm
5. **Test** and verify the completed program
6. **Maintain** and update the program

Steps Defined

1. **Problem** - Specifying the problem requirements forces you to understand the problem more clearly.
2. **Analysis** - Analyzing the problem involves identifying the problem's inputs, outputs, and additional requirements.
3. **Design** - Designing the algorithm to solve the problem requires you to develop a list of steps called an **algorithm** that solves the problem and then to verify the steps.
4. **Implementation** - Implementing is writing the algorithm as a program.
5. **Testing** - Testing requires verifying that the program actually works as desired.
6. **Maintenance** - Maintaining involves finding previously undetected errors and keep it up-to-date.



Converting Miles to Kilometers

1. **Problem**: Your boss wants you to convert a list of miles to kilometers. Since you like programming, so you decide to write a program to do the job.
2. **Analysis**
   - We need to get miles as input
   - We need to output kilometers
   - We know 1 mile = 1.609 kilometers
3. **Design**
   1. Get distance in miles
   2. Convert to kilometers
   3. Display kilometers

4. Implementation

```
/* Converts distances from miles to kilometers */

#include <iostream>                    /* cin and couit definitions */
Using namespace std;
#define KMS_PER_MILE 1.609             /* conversion constant */

int main()
{
    double miles,      //distance in miles
           kms;        //equivalent distance in kilometers

    //Get the distance in miles
    cout<<"Enter the distance in miles> ";
    cin>>miles;

    //Convert the distance to kilometers
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometers
    cout<<"That equals "  << kms << " kilometers.\n";

    return 0;
}
```

Miles to Kilometers cont'd

5. Test

- We need to test the previous program to make sure it works. To test we run our program and enter different values and make sure the output is correct.

6. Maintenance

- includes any functionality changes to meet new requirements, as well as performance improvements.

Testing and Debugging

- Bug
  - A mistake in a program
- Debugging
  - Eliminating mistakes in programs

Program Errors

- Syntax errors
  - Violation of the grammar rules of the language
  - Discovered by the compiler
    - Error messages may not always show correct location of errors
- Run-time errors
  - Error conditions detected by the computer at run-time
- Logic errors
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize an error

Pseudo code & Flowchart

- **Pseudo code** - A combination of English phrases and language constructs to describe algorithm steps
- **Flowchart** - A diagram that shows the step-by-step execution of a program.
- **Algorithm** - A list of steps for solving a problem.

Why use pseudo code?

- Pseudo code cannot be compiled nor executed, and there are no real formatting or syntax rules.
- It is simply one step - an important one - in producing the final code.
- The benefit of pseudo code is that it enables the programmer to concentrate on the algorithms without worrying about all the syntactic details of a particular programming language.
- In fact, you can write pseudo code without even knowing what programming language you will use for the final implementation.
- Example:

  Input Miles

  Kilometers = Miles * 1.609

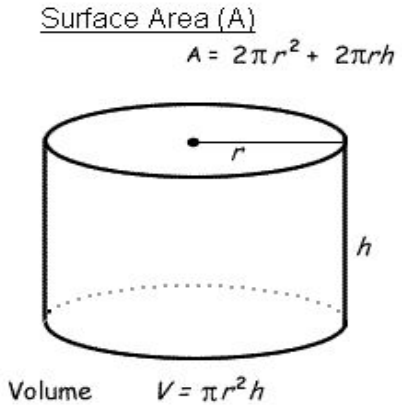
  Output Kilometers

Another Example of Pseudo code

- **Problem**: Calculate your final grade for CMPE110
- **Specify the problem** - Get different grades and then compute the final grade.
- **Analyze the problem** - We need to input grades for exams, labs, quizzes and the percentage each part counts for. Then we need to output the final grade.
- **Design**
  1. Get the grades: quizzes, exams, and labs.
  2. Grade = .30 * 2 regular exams & quizzes + .20 * Final exam + .50 * labs
  3. Output the Grade
- **Implement** – Try to put some imaginary number and calculate the final grade after you learn how to program.

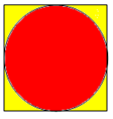
Exercise

- Develop a pseudo code algorithm for an interactive program to find the **surface area (A)** of a cylinder given its volume (**V**), and its height (**h**) as inputs.

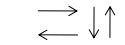


Exercise

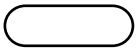
- Develop a pseudo code algorithm for computing the shaded area with colour yellow shown in the diagram.



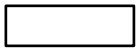
Flowchart Symbols



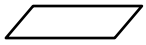
Flowlines Connects blocks and shows the direction of flow.



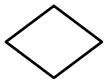
Start/Stop or Begin/End: Shows the start and the end.



Processing. Indicates a processing block such as calculations



I/O: Input to and output from the computer



Decision: Used for comparison operations

On-Page Connector Flowchart sections can be connected by these symbols

Off-Page Connector Flowchart sections on different pages can be connected by these symbols

Flowchart for computing the area of a circle with radius r: