

# ICS103 Programming in C

## Lecture 1: Overview of Computers & Programming

# Outline

- Overview of Computers
  - Hardware
  - Software
- Computer Languages
- Software Development Method
- Pseudo Code and Flowcharts

# 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 tells the computer what to do and thus provides a way to communicate with computers.
- Programs are then converted to machine language (0 and 1) so the computer can understand it.

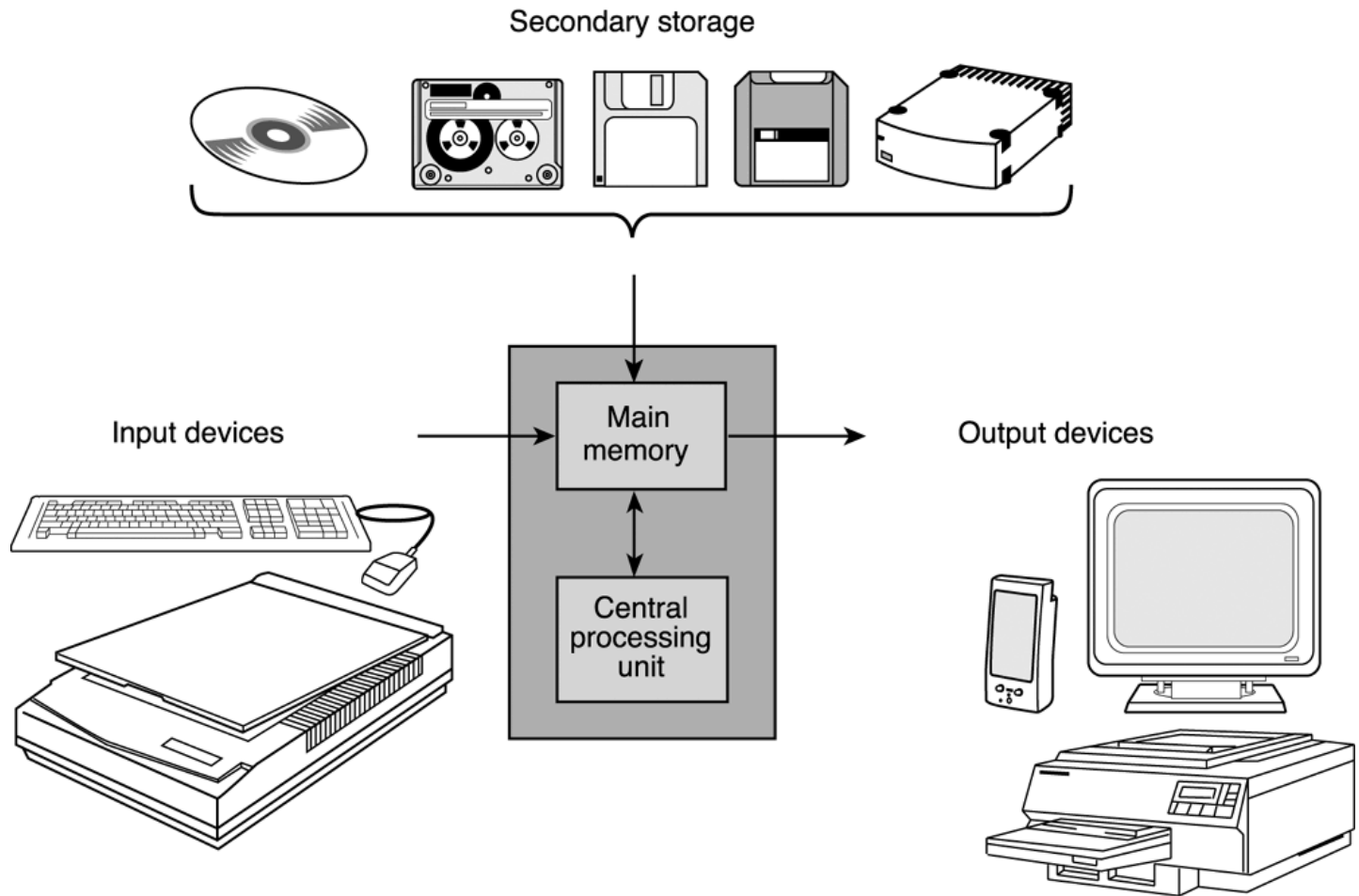
# Hardware & Software

- 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.

# 1000 memory cells in Main memory

Memory

Address	Contents
0	-27.2
1	354
2	0.005
3	-26
4	H
⋮	⋮
998	X
999	75.62



# 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.

# 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.

# Example of Computer Languages

C Source Code:

```
char name[40];  
printf("Please enter your name\n");  
scanf("%s", name);  
printf("Hello %s", name);
```

Assembly Code:

```
push    offset string "Please enter your name\n"  
(41364Ch)  
call    dword ptr [__imp__printf (415194h)]  
add     esp,4  
lea     eax,[name]  
push    eax  
push    offset string "%s" (413648h)  
call    dword ptr [__imp__scanf (41519Ch)]  
add     esp,8  
lea     eax,[name]  
push    eax  
push    offset string "Hello %s" (41363Ch)  
call    dword ptr [__imp__printf (415194h)]  
add     esp,8
```

Machine Code:

```
68 4C 36 41 00 FF 15 94 51 41 00 83 C4 04 8D 45 D8  
50 68 48 36 41 00 FF 15 9C 51 41 00 83 C4 08 8D 45  
D8 50 68 3C 36 41 00 FF 15 94 51 41 00 83 C4 08
```

# 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

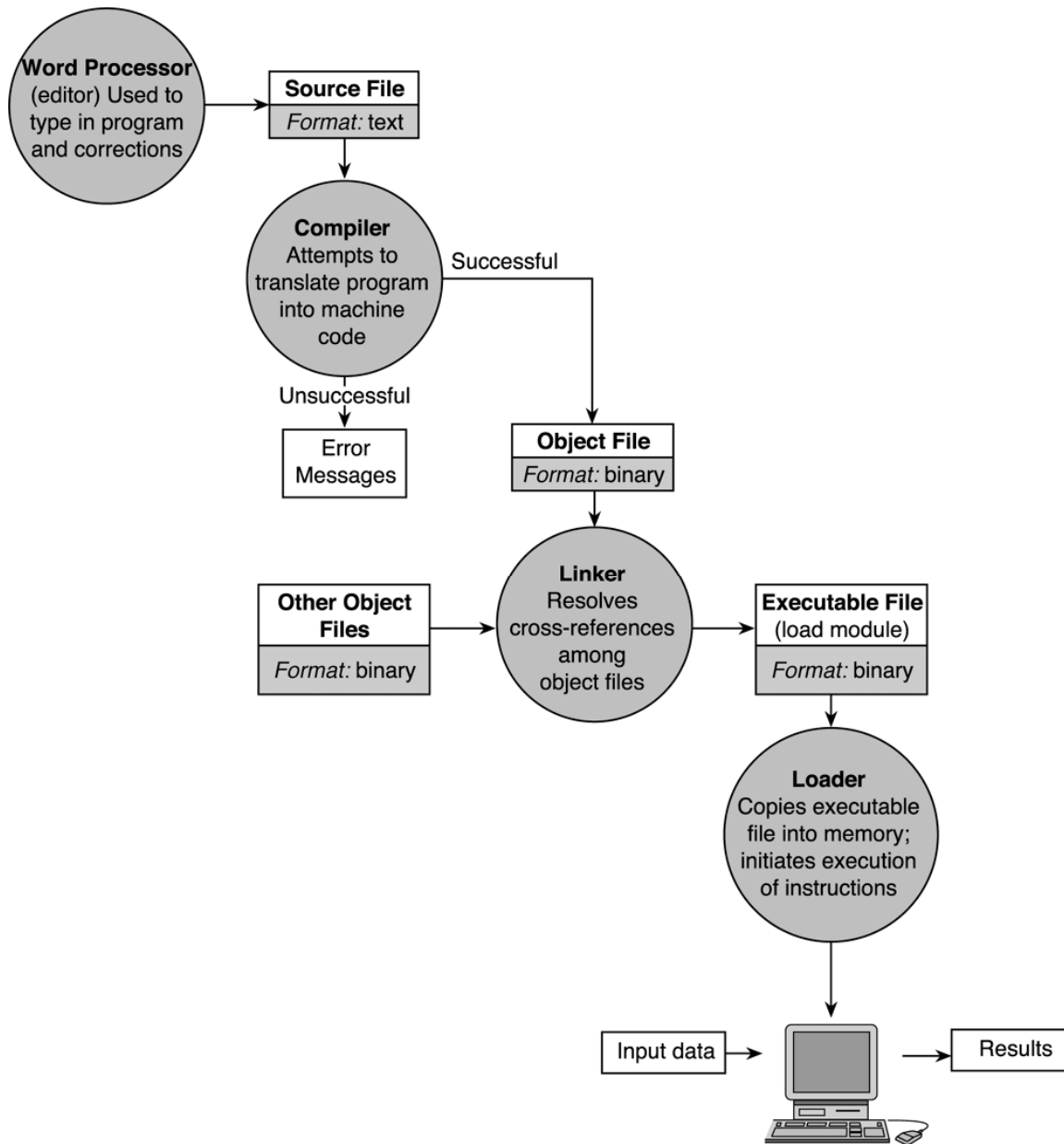
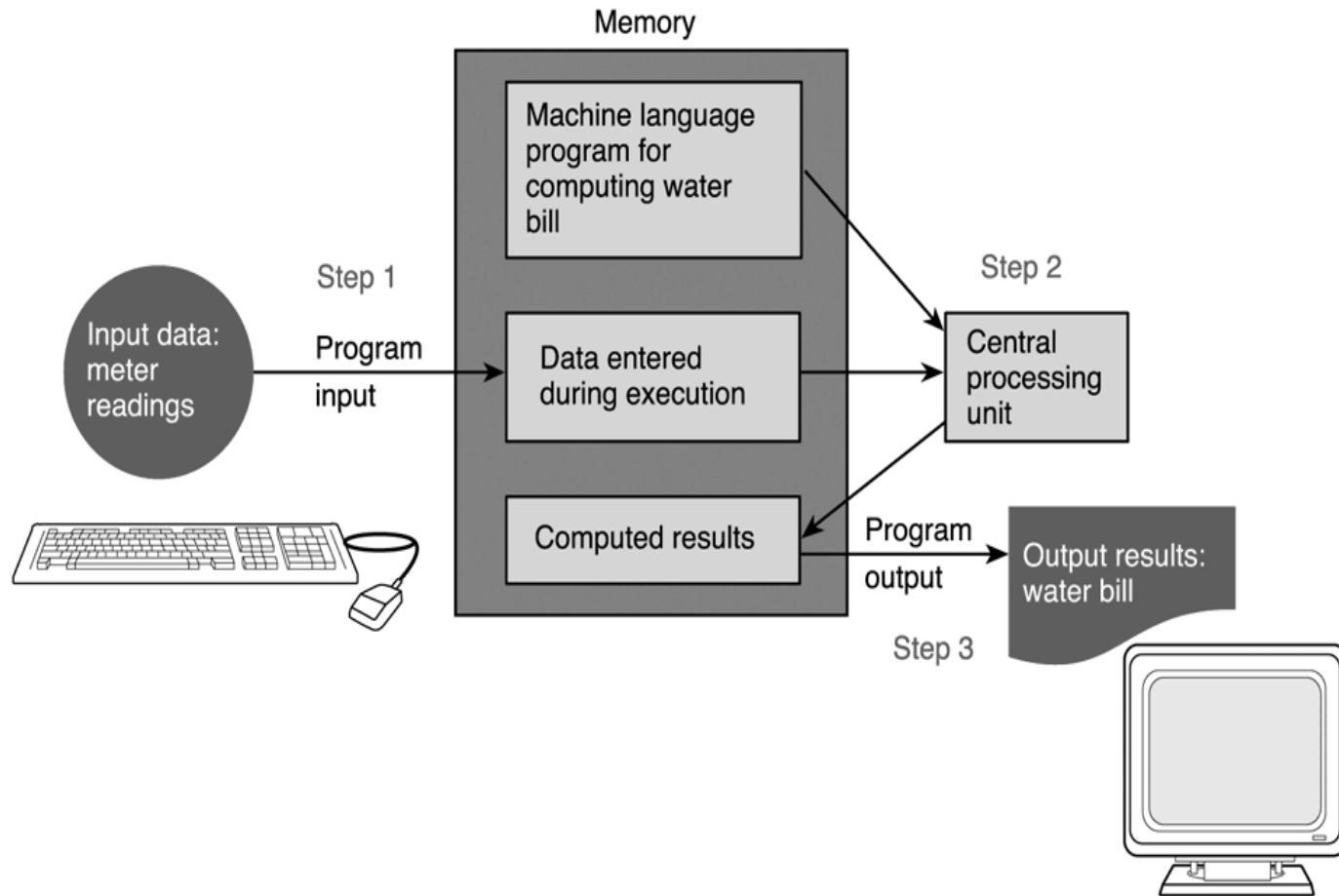


Fig 1.12  
Entering,  
Translating,  
and Running  
a High-Level  
Language  
Program

# 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

```
1. /*
2.  * Converts distance in miles to kilometers.
3.  */
4. #include <stdio.h>           /* printf, scanf definitions */
5. #define KMS_PER_MILE 1.609   /* conversion constant      */
6.
7. int
8. main(void)
9. {
10.     double miles, /* input - distance in miles.      */
11.           kms;    /* output - distance in kilometers */
12.
13.     /* Get the distance in miles. */
14.     printf("Enter the distance in miles> ");
15.     scanf("%lf", &miles);
16.
17.     /* Convert the distance to kilometers. */
18.     kms = KMS_PER_MILE * miles;
19.
20.     /* Display the distance in kilometers. */
21.     printf("That equals %f kilometers.\n", kms);
22.
23.     return (0);
24. }
```

## Sample Run

```
Enter the distance in miles> 10.00
That equals 16.090000 kilometers.
```

# 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

- Next time, your boss gets a contract with NASA, so he wants you to add support for converting to AU's

# 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

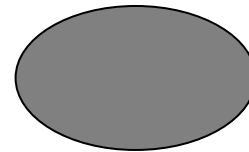
- **Problem:** Calculate your final grade for ICS 103
- **Specify the problem** - Get different grades and then compute the final grade.
- **Analyze the problem** - We need to input grades for exams, labs, quizzes and the percentage each part counts for. Then we need to output the final grade.
- **Design**
  1. Get the grades: quizzes, exams, and labs.
  2.  $\text{Grade} = .30 * 2 \text{ regular exams \& quizzes} + .20 * \text{Final exam} + .50 * \text{labs}$
  3. Output the Grade
- **Implement** – Try to put some imaginary number and calculate the final grade after you learn how to program.

# Flowcharts

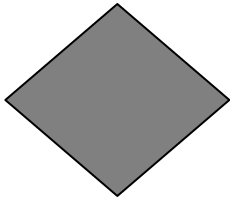
Flowchart uses boxes and arrows to show step by step execution of a program.



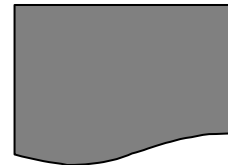
Process



Start or Terminal



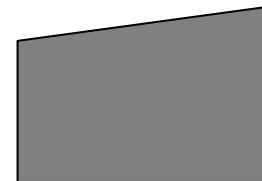
Decision



Document



Display



Manual Input

# Example of a Flowchart

