Syrian Private University

Introduction to Algorithms and Programming

Instructor: Dr. Mouhib Alnoukari

Introduction to Algorithms & C Programming

Course focus, First Program, and C Programming

Focus of the Course

- This is an intro to problem solving and programming class (that uses the C programming language).
- The main focus is on:
  - Problem solving
  - The logic of programming
  - Program design, implementation, and testing
  - The fundamentals of programming

Book: Let Us C - Yashwant Kanetkar

Notes …

- Learning how to program takes a lot of time!
- It also requires a lot of patience.
- You cannot learn how to program by just reading the textbook. You have to spend long hours in front of the computer.
- If you want to learn how to program well you will have to take at least 2-3 other programming classes as well. This class alone is not enough!
- This class is not exclusively about writing code. It also emphasizes Problem Solving.

Hardware & Software

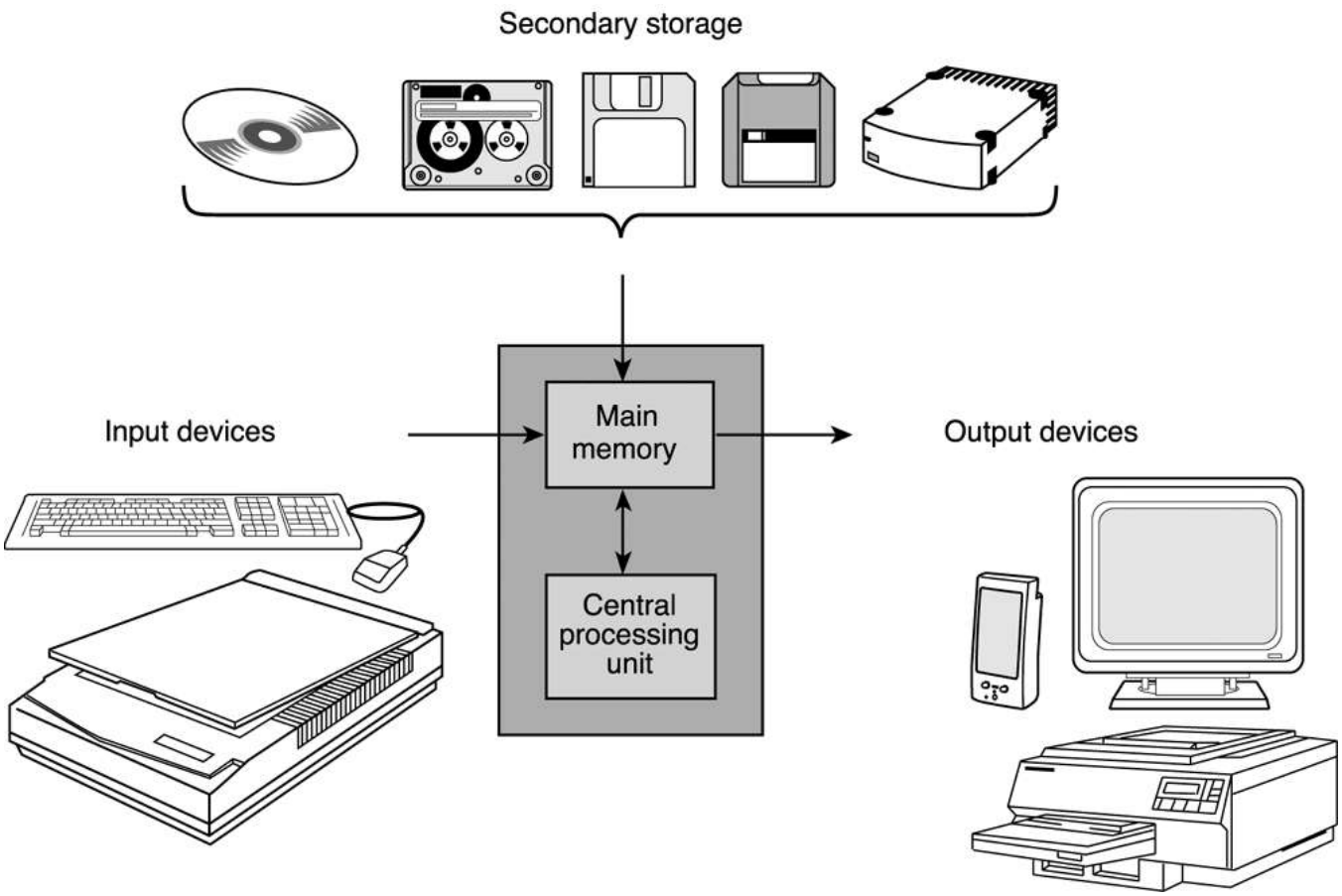
Hardware & Software

- Hardware
  - Physical, tangible parts of a computer.
  - Keyboard, monitor, disks, wires, chips, etc.
- Software
  - Programs and data (intangible).
  - A *program* is a series of instructions.
- A computer requires both hardware and software.
- Each is essentially useless without the other.

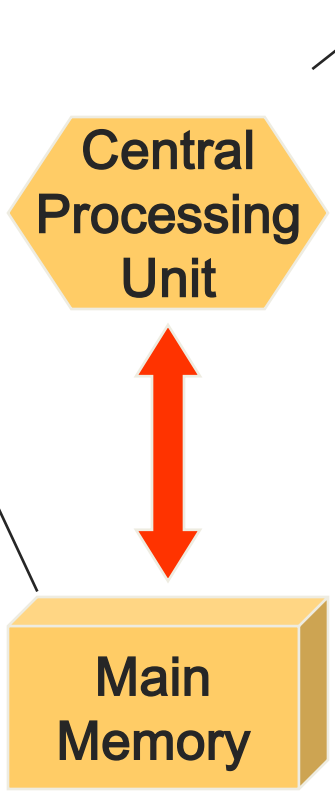
Software Categories

- Operating System
  - controls all machine activities.
  - provides the user interface to the computer.
  - manages resources such as the CPU and memory.
  - Windows XP, Unix, Linux, Mac OS.
- Application program
  - generic term for any other kind of software.
  - word processors, missile control systems, games.
- Most operating systems and application programs have a *graphical user interface* (GUI).

Computer Components



CPU & Main Memory



Chip that executes program commands

Primary storage area for programs and data that are in active use

Synonymous with RAM

RAM is volatile memory

Start-up instructions are burnt into ROM

Memory

Memory

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

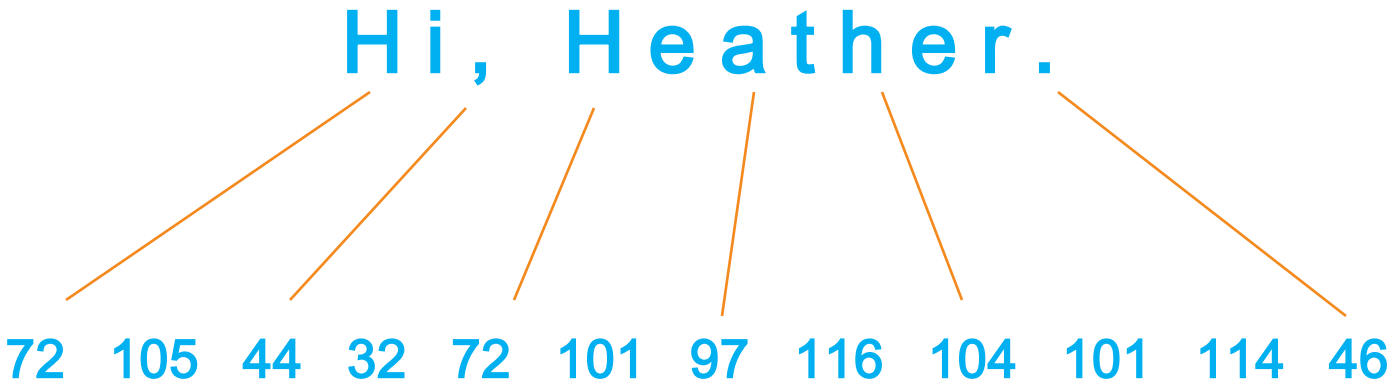
Most modern computers are byte-addressable

Digital Information

- Computers store all information digitally:
  - numbers
  - text
  - graphics and images
  - video
  - audio
  - program instructions
- In some way, all information is *digitized* - broken down into pieces and represented as numbers

Representing Text Digitally

- For example, every character is stored as a number, including spaces, digits, and punctuation
- Corresponding upper and lower case letters are separate characters

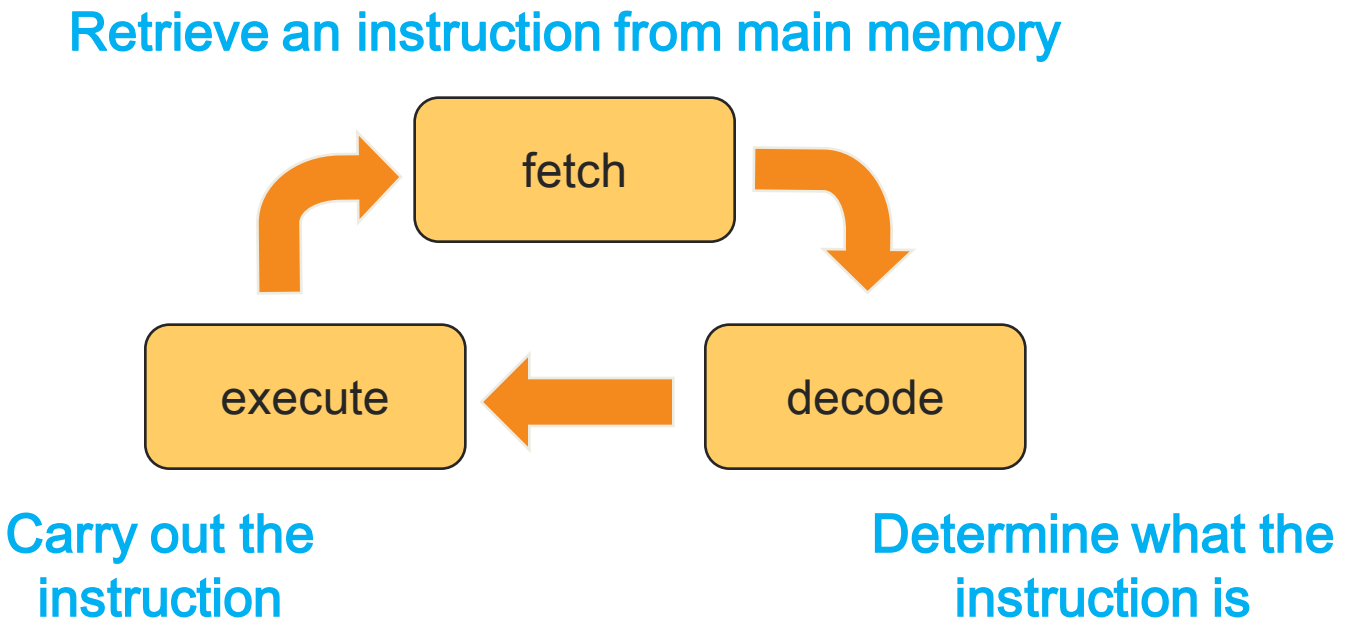


Binary Numbers

- Once information is digitized, it is represented and stored in memory using the *binary number system*
- A single binary digit (0 or 1) is called a *bit*
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations of bits are used to store values

Central Processing Unit

- A CPU is on a chip called a *microprocessor*
- It continuously follows the *fetch-decode-execute cycle:*

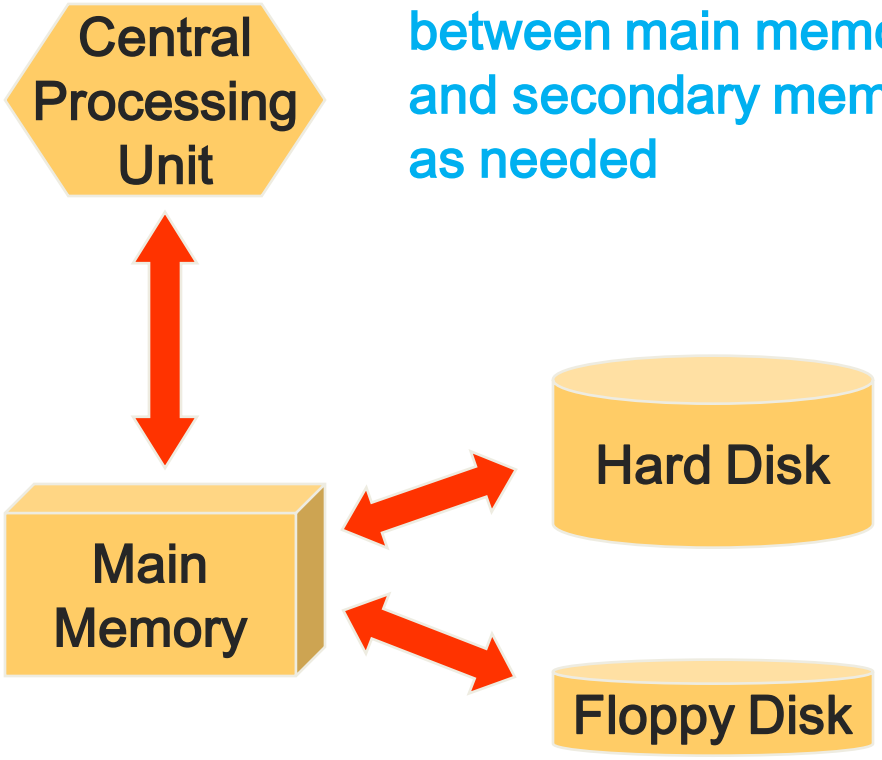


Secondary Memory Devices

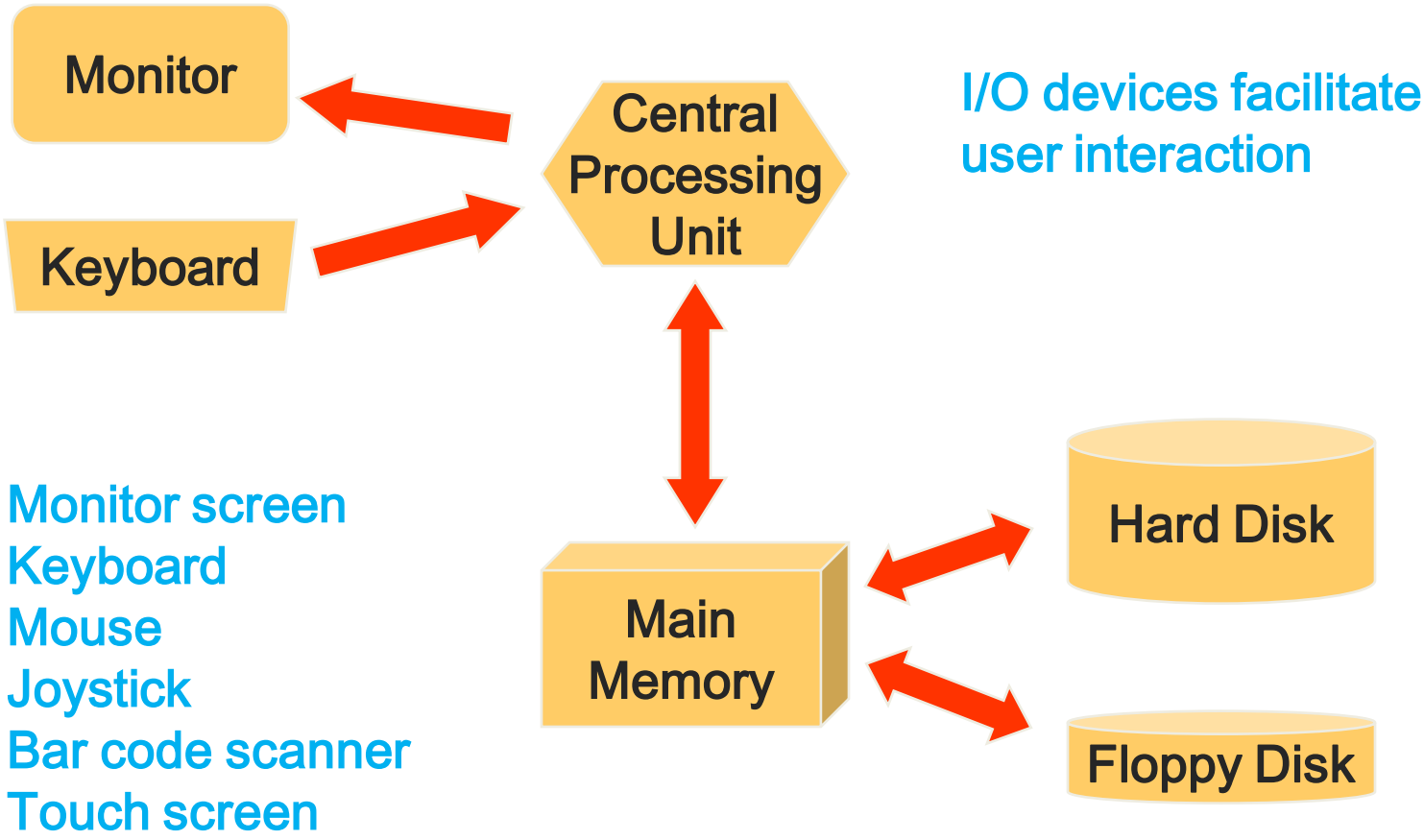
Secondary memory devices provide long-term storage

Information is moved between main memory and secondary memory as needed

- Hard disks
- Floppy disks
- ZIP disks
- Writable CDs
- Writable DVDs
- Tapes



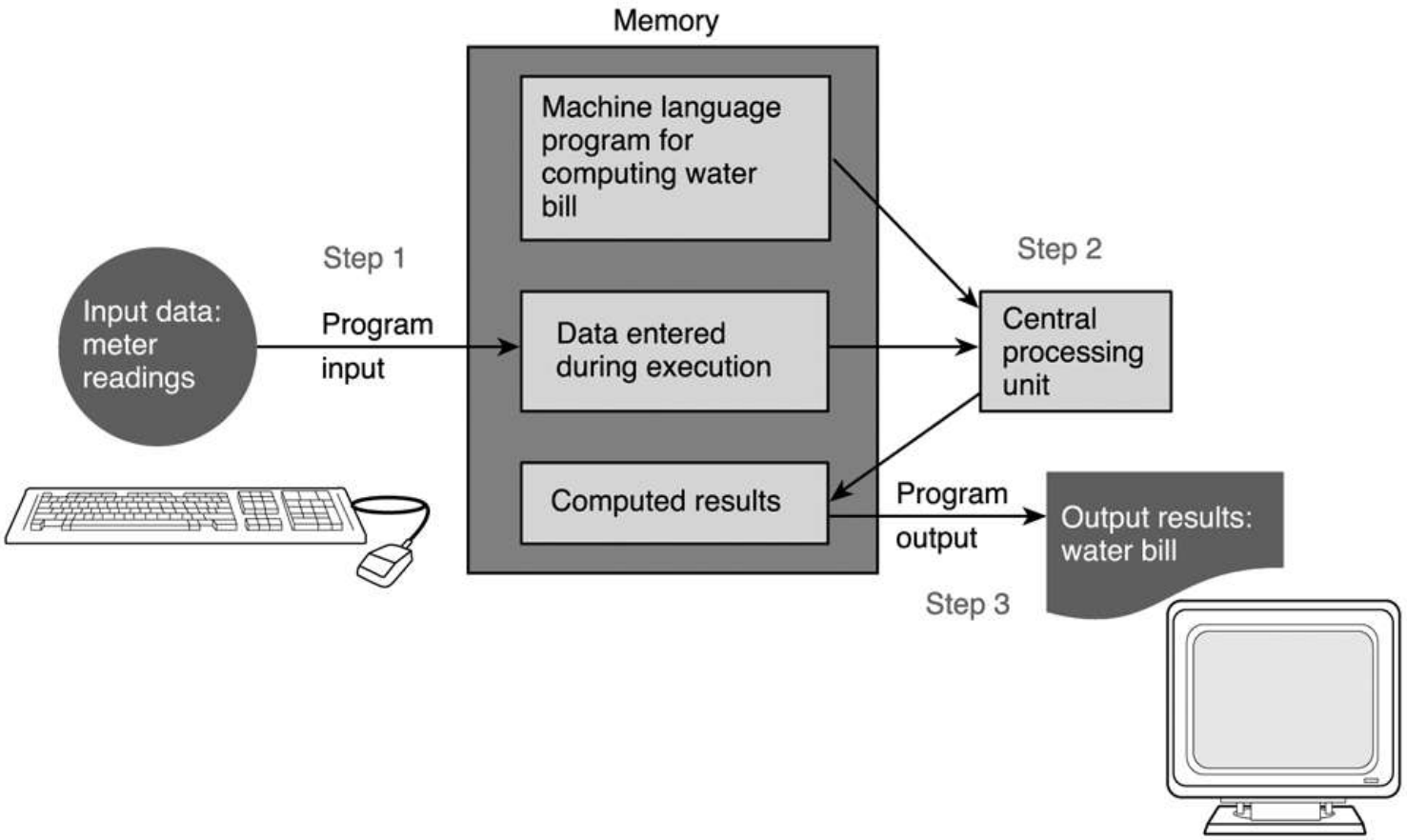
Input / Output Devices



I/O devices facilitate user interaction

Monitor screen
Keyboard
Mouse
Joystick
Bar code scanner
Touch screen

Flow of Information During Program Execution

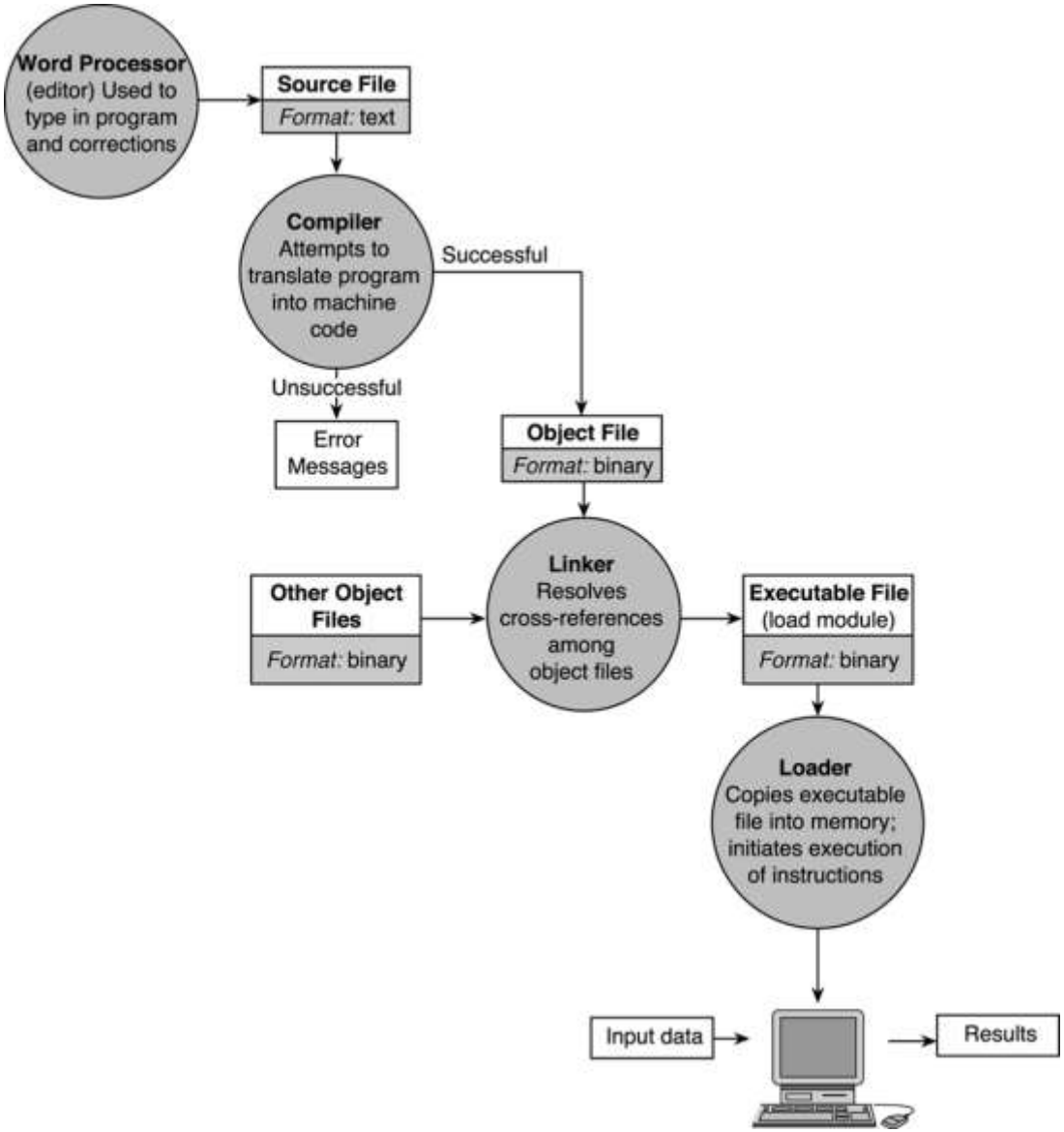


Program Development

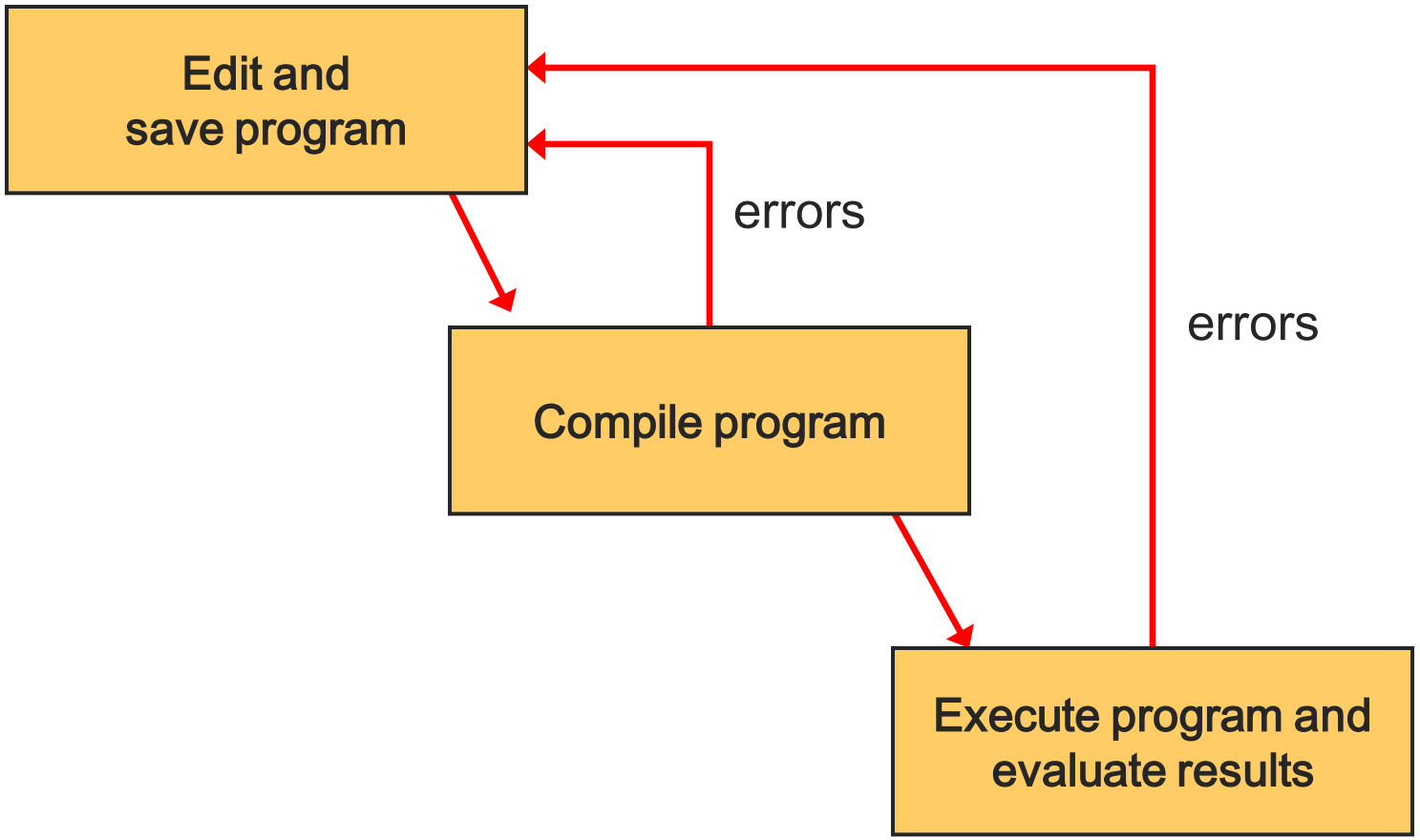
Program Development

- The mechanics of developing a program include several activities:
  - Writing the program in a specific programming language.
  - Translating the program into a form that the computer can execute.
  - Investigating and fixing various types of errors that can occur.
- Software tools can be used to help with all parts of this process.

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



Basics Program Development



Programming Languages

- Each type of CPU executes only a particular *machine language.*
- A program must be translated into machine language before it can be executed.
- A *compiler* is a software tool which translates *source code* into a specific target language.
- Often, that target language is the machine language for a particular CPU type.

Your First Program!

Using C Programming Language

C Programming Language

- A *programming language* specifies the words and symbols that we can use to write a program.
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements.*

The computer will always do what you tell it to do, not what you want it to do.

C Program Structure

```
#include <stdio.h>
main()
{
    //  comments about the program

    printf("Hello World! \n");


}
```

C Program Structure

```
#include <stdio.h>
main()
{



}
```

All programs must have a main function

C Program Structure

```
#include <stdio.h>
main()
{


}
```

main function body

Comments can be placed almost anywhere

C Language

Using C Programming Language

C History

- C evolved from two previous languages, BCPL (Basic Combined Programming Language) and B.
- BCPL developed in 1967 by Martin Richards as a language for writing OSes and compilers.
- Ken Thompson modeled many features in his language, B, after their counterparts in BCPL, and used B to create an early versions of UNIX operating system at bell Laboratories in 1970 on a DEC PDP-7 computer.
- Both BCPL and B were typeless languages: the only data type is machine word and access to other kinds of objects is by special operators or function calls.
- The C language developed from B by Dennis Ritchie at Bell Laboratories and was originally implemented on a DEC PDP-11 computer in 1972.
- It was named C for new language (after B).
- Initially, C used widely as the development language of the UNIX OS.
- Today, almost all new major OS are written in C including Windows.

C Program Structure – Identifiers

1. Is a unique name that simply references to memory locations, which can hold values (data).
2. Identifiers give unique names to various objects in a program.
3. Are formed by combining letters (both upper and lowercase), digits (0–9) and underscore (_).
4. Rules for identifier naming are:
   a) The first character of an identifier must be a letter (non-digit) including underscore (_).
   b) The blank or white space character is not permitted in an identifier. Space, tab, linefeed, carriage-return, formfeed, vertical-tab, and newline characters are "white-space characters“ - they serve the same purpose as the spaces between words and lines on a printed page.
   c) Can be any length but implementation dependent.
   d) Reserved words/keywords cannot be used.

C Program Structure – Identifiers

Examples: variable names

Correct	Wrong
`secondName`	`2ndName /* starts with a digit */`
`_addNumber`	`%calculateSum /* contains invalid character */`
`charAndNum`	`char /* reserved word */`
`annual_rate`	`annual rate /* contains a space */`
`stage4mark`	`my\nName /* contains new line character, \n */`

C Program Structure – Variables

- are named blocks of memory & is any valid identifier.
- Have two properties in syntax: name — a unique identifier & type — what kind of value is stored.
- It is identifier, that value may change during the program execution.
- Every variable stored in the computer's memory has a name, a value and a type.

C Program Structure – Variables

- More examples

Correct	Wrong	Comment
`int x, y, z;`	`int 3a, 1, -p;`	
`short number_one;`	`short number+one;`	
`long TypeofCar;`	`long #number`	
`unsigned int positive_number;`	…	
`char Title;`		
`float commission, yield = 4.52;`		
`int my_data = 4;`		
`char the_initial = 'M';`		A char
`Char studentName[20] = "Anita";`		A string

C Program Structure – Assignment

- An *assignment statement* changes the value of a variable.
- The assignment operator is the = sign

```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left.
- The value that was in total is overwritten.
- You can only assign a value to a variable that is consistent with the variable's declared type.

C Program Structure – Contants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- As the name implies, it is constant, not variable
- The compiler will issue an error if you try to change the value of a constant
- In C, we use the `const` modifier to declare a constant

```
const int MIN_HEIGHT = 69;
```

C Program Structure – Contants

- Constants are useful for three important reasons:

1. First, they give meaning to otherwise unclear literal values.
   - For example, MAX_LOAD means more than the literal 250
2. Second, they facilitate program maintenance.
   - If a constant is used in multiple places, its value need only be updated in one place
3. Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers.

C Program Structure – Keywords

- Reserved words in C & are not available for re-definition.
- have special meaning in C.

auto	extern	short	_Alignas (C11)
break	float	signed	_Alignof (C11)
case	for	sizeof	_Atomic (C11)
char	goto	static	_Bool (C99 beyond)
const	if	struct	_Complex (C99 beyond)
continue	inline (C99	switch	_Generic (C11)
default	beyond)	typedef	_Imaginary (C99
do	int	union	beyond)
double	long	unsigned	_Noreturn (C11)
else	register	void	_Static_assert (C11)
enum	restrict (C99	volatile	_Thread_local (C11)
	beyond)	while	
	return		

C Program Structure – Others

- Statements are terminated with a ';'
- e.g:

```
char acharacter;
int i, j = 18, k = -20;
printf("Initially, given j = 18 and k = -20\n");

for(; count != 0; count = count - 1)
```

C Program Structure – Others

- Group of statements (compound statement) are enclosed by curly braces: { and }.
- Mark the start and the end of code block.
- Also used in initializing a list of aggregate data values such as in array and enum type.

```
int    id[7] = {1, 2, 3, 4, 5, 6, 7};
float  x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};
char   vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};

enum days {Mon, Tue, Wed, Thu, Fri,
Sat, Sun};
```

```
#include <stdio.h>

int main()
{
    int i, j = 18, k = -20;
    printf("Initially, given j = 18 and k = -20\n");
    printf("Do some operations..."
           "i = j / 12, j = k / 18 and k = k / 4\n");
    i = j / 12;
    j = k / 8;
    k = k / 4;
    printf("At the end of the operations...\n");
    printf("i = %d, j = %d and k = %d\n", i, j, k);
    return 0;
}
```

C Program Structure - Comments

- Comments in a program are called *inline* documentation
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- C comments can take two forms:

```
// this comment runs to
the end of the line

/*  this comment runs to the
terminating
    symbol, even across line
breaks        */
```

```
// for printf()
#include <stdio.h>
#include <string.h> // for strcpy_s() and their family

/* main() function, where program
     execution starts */
int main()
{
      /* declares variable and initializes it*/
      int i = 8;
…
```

C Program Structure – White Apace

- Spaces, blank lines, and tabs are called white space.
- White space is used to separate words and symbols in a program.
- Extra white space is ignored.
- A valid C program can be formatted many ways.
- Programs should be formatted to enhance readability, using consistent indentation.

```
#include <stdio.h>

void main(void)
{
        int MyAge = 12;
        printf("My name is Mr. C. Cplusplus\n");
… }
```

C Program Structure - Commas

- Commas separate function arguments, list of variables, aggregate values. e.g.

```
#include <stdio.h>

int main(int argc, int argv)
{
    int i = 77, j, k;
    j = i + 1;  k = j + 1;  i = k + j;
    printf("Finally, i = %d\n", i);
    printf("... and j = %d\n", j);
    printf("... and k = %d\n", k);
    return 0;
}
```

```
int   id[7] = {1, 2, 3, 4, 5, 6, 7};
float  x[5] = {5.6, 5.7, 5.8, 5.9, 6.1};
char   vowel[6] = {'a', 'e', 'i', 'o', 'u', '\0'};


enum days {Mon, Tue, Wed, Thu, Fri,
Sat, Sun};
```

PROGRAM ERRORS

SYNTAX & SEMATIC

Program Errors

- A program can have three types of errors:

  1. The compiler will find syntax errors and other basic problems (*compile-time errors*)

     - If compile-time errors exist, an executable version of the program is not created

  2. A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)

  3. A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

Programming: Syntax & Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program.
- The *semantics* of a program statement define what that statement means (its purpose or role in a program).
- A program that is syntactically correct is not necessarily logically (semantically) correct.

Programming: Syntax & Semantics

- e.g.

To add an integer to a variable `q` and store the result in `q` (semantic), syntaxically (correct), we can write:

```
q = q + 3; or q += 3;
```

- *Pseudocode* - an informal high-level description of the operating principle of a computer program or other algorithm.
- Uses the structural conventions of a programming language, but is intended for human reading rather than machine reading.

PSEUDOCODE & ALGORITHM

PSEUDOCODE & ALGORITHM

- An informal high-level description of a computer program or algorithm operating principle.
- An *algorithm* is merely the sequence of steps taken to solve a problem which are normally a sequence, selection, iteration and a *case*-type statement.
- Algorithm is a procedure for solving a problem - actions to be executed and the order in which those actions are to be executed.
- e.g. to sort ascendingly, the given unsorted integers, we can achieve this by using several different algorithms.
- Every algorithm may have different number line of code, different repetition loops, different execution speeds etc.

PSEUDOCODE & ALGORITHM

- But all the program have similar purpose: to sort the given unsorted integers in ascending order.
- Pseudocode uses programming language's structural conventions , intended for human rather than machine reading.
- helps programmers develop algorithms.

PSEUDOCODE & ALGORITHM

```
Set sum to zero
Set grade counter to one
While grade counter is less than or equal to ten
        Input the next grade
        Add the grade into the sum
Set the class average to the sum divided by ten
Print the class average.
```

```
IF HoursWorked > NormalMax THEN
        Display overtime message
ELSE
        Display regular time message
ENDIF
```

```
SET total to zero
REPEAT
        READ Temperature
        IF Temperature > Freezing THEN
                INCREMENT total
        END IF
UNTIL Temperature < zero
Print total
```