PRINCIPLES OF PROGRAMMING USING C

- DR. SANJAY H A

COURSE OUTCOMES

At the end of the course the student will be able to:

1.Elucidate the basic architecture and functionalities of a computer and also recognize the hardware parts.

2.Apply programming constructs of C language to solve the real-world problem

3.Explore user-defined data structures like arrays in implementing solutions to problems like searching and sorting

4.Explore user-defined data structures like structures, unions and pointers in implementing solutions

5.Design and Develop Solutions to problems using modular programming constructs

UNIT I- INTRODUCTION TO C:

- **Introduction to C:** Introduction to computers, input and output devices, designing efficient programs. Introduction to C, Structure of C program, Files used in a C program, Compilers, Compiling and executing C programs, variables, constants, Input/output statements in C.

UNIT II

- Operators in C, Type conversion and typecasting.
- **Decision control and Looping statements:** Introduction to decision control, Conditional branching statements, iterative statements, nested loops, break and continue statements, go to statement.

UNIT III

- **Functions:** Introduction using functions, Function definition, function declaration, function call, return statement, passing parameters to functions, scope of variables, storage classes, recursive functions.
- **Arrays:** Declaration of arrays, accessing the elements of an array, storing values in arrays, Operations on arrays, Passing arrays to functions, two dimensional arrays, operations on two-dimensional arrays, two- dimensional arrays to functions, multidimensional arrays, applications of arrays.

UNIT IV

- **Strings :** Introduction, string taxonomy, operations on strings, Miscellaneous string and character functions, arrays of strings.
- **Pointers:** Introduction to pointers, declaring pointer variables, Types of pointers, passing arguments to functions using pointers

UNIT V

- **Structure, Union, and Enumerated Data Type**: Introduction, structures and functions, Unions, unions inside structures, Enumerated data type.
- **Files:** Introduction to files, using files in C, reading and writing data files., Detecting end of file

TEXT BOOK

1. Behrouz A. Forouzan, Richard F. Gilberg, Computer science a structured programming approach using C, Third Edition, Cengage Learning, and ISBN: 9788131503638, 8131503631, and 2007.

DELIVERY METHOD

- Blackboard Teaching/ Power Point Presentation
- Live Demonstration
- Hands-on Session

ASSESSMENT METHOD

- Continuous Internal Evaluation-50 Marks
  - Mid Term Test -30 Marks
  - Laboratory Test -20 Marks
- Semester End Examination – 50 Marks

DEFINITION OF A COMPUTER

"A computer is an electronic device,

operating under the control of instructions stored in its own memory unit,

that can accept data (input),

process data arithmetically and logically,

produce information (output) from the processing,

and store the results for future use."

FUNCTIONS OF A COMPUTER

- Four operations performed
- The four operations are referred to as the information processing cycle:
  - **Input, Process, Output, and Storage.**
- Computers transform raw data into information
- People who use this information are referred to as end users, computer users or users

INTRODUCTION TO COMPUTER HARDWARE

- Hardware vs Software
- Hardware is everything you can touch and see
- Examples: Monitor, hard drive, CD-ROM, computer cables, keyboard, mouse, modem, printer, etc.

HARDWARE VS SOFTWARE

- **Hardware:** Physical components of a computer.

Ex: CPU, RAM, Hard disk, keyboard, Mouse etc..

- **Software:** One or more programs along with their documentation.

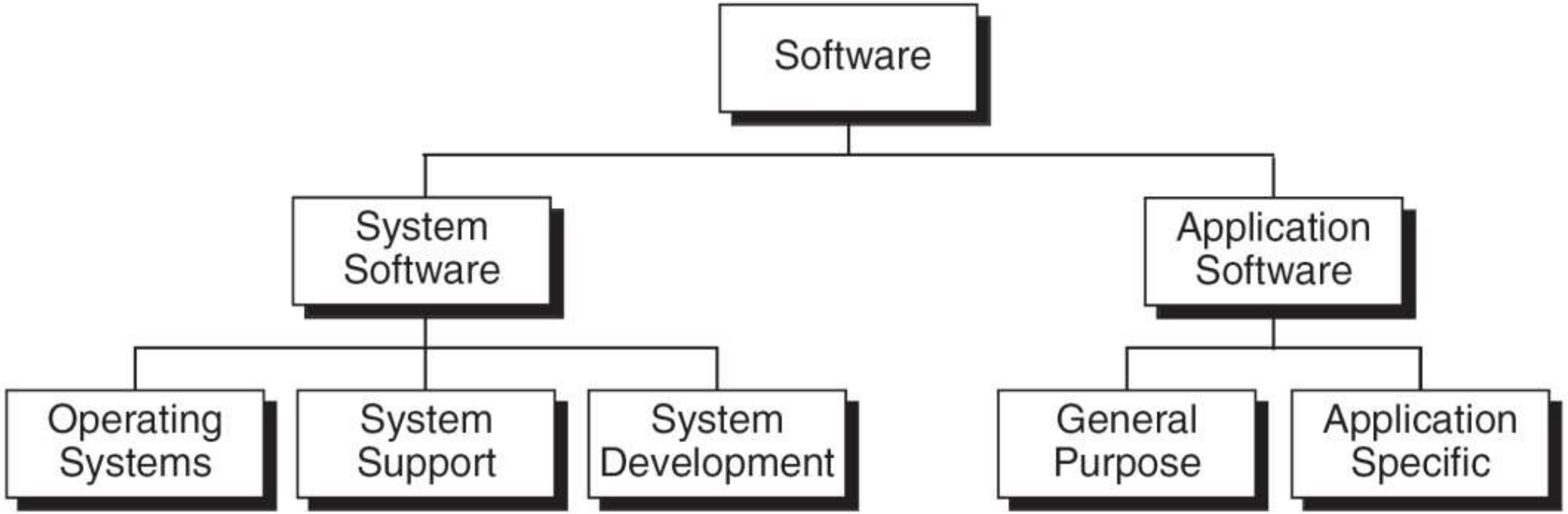
Two categories of software

1. System software
2. Application software

SOFTWARE

- Software are programs & applications
  - Part of computer that cannot be seen
  - Needed for computer to function
  - Designed to solve common or custom problems

Types of Software:



SYSTEM SOFTWARE

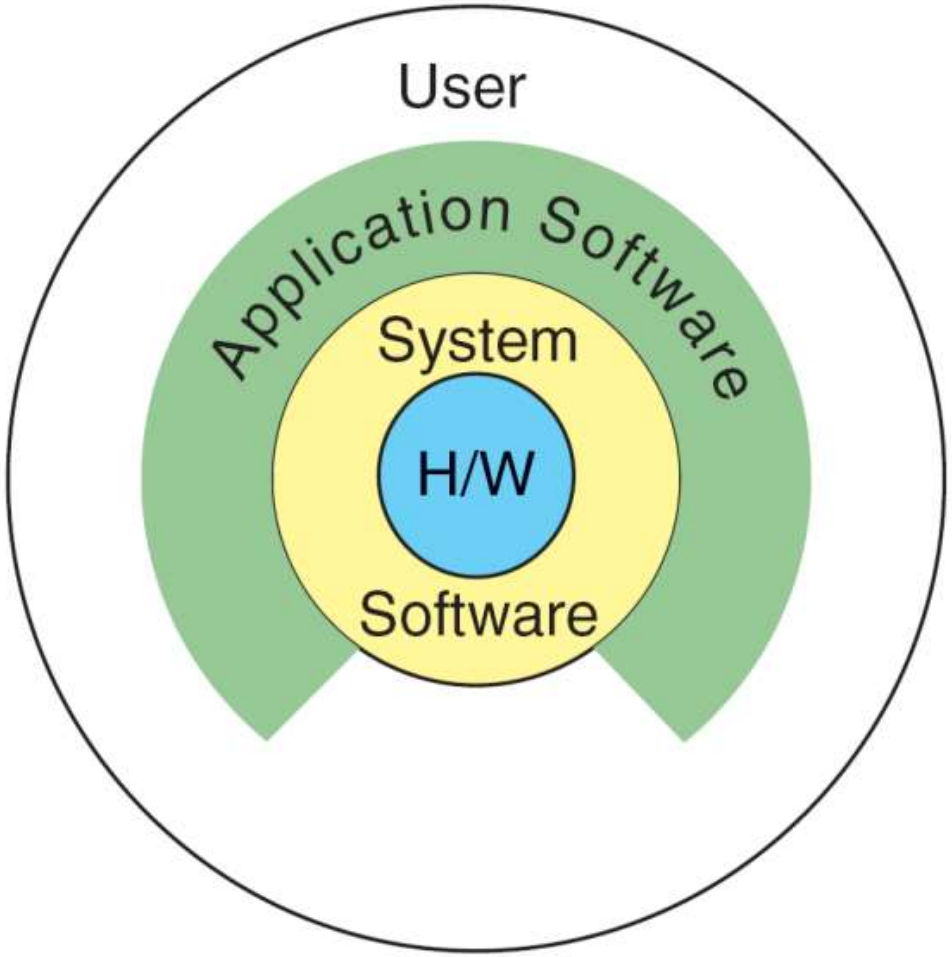
- **Operating system** is a Software component that provides an interface between user and system hardware.
  - Ex: Windows ,Linux,Unix,Solaris etc.
- **System support Software** provides system utilities and other operating services.
  - Eg: Sort programs, Formatting programs, Linkers, Loaders

SYSTEM SOFTWARE

- **System Development software** includes the language translators (Compilers, Assemblers etc.) that convert programs into machine language for execution, debugging tools to ensure that the programs are error-free and CASE tools for software engineering processes
- Eg: C Compiler, Java Compiler

APPLICATIONSOFTWARE

- **Application specific software** can be used for a specific intended purpose
  - Ex: Pay roll, Inventory Management, Library management etc.
- **General Purpose software** is intended for use in more than one application
  - Ex: Word Processors, Database Management systems and Computer- aided Design Systems.



Relation Ship Between System and Application Software

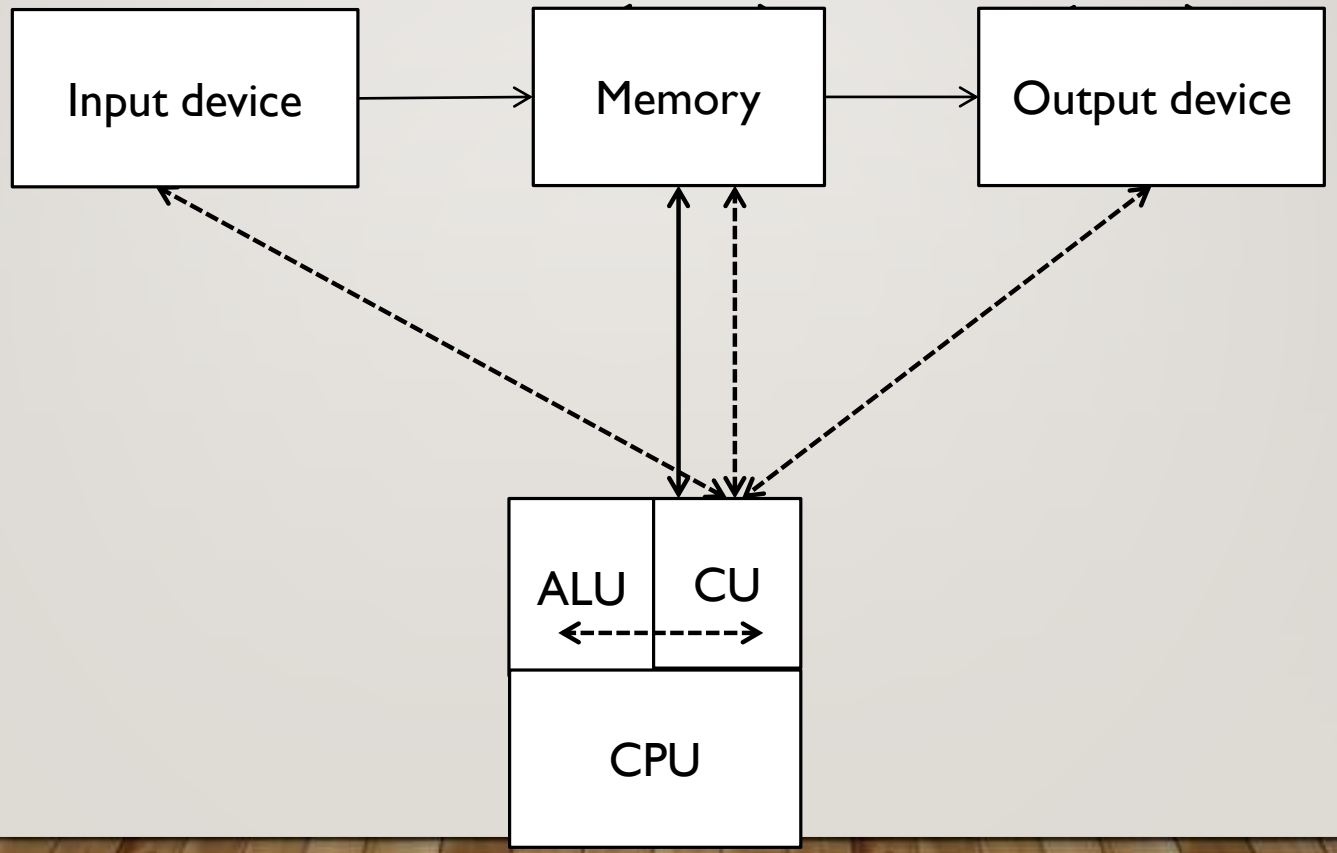
COMPONENTS OF A COMPUTER

- There are four components to a computer
  - Input Devices
  - Processor Unit
  - Output Devices
  - Auxiliary Storage Devices

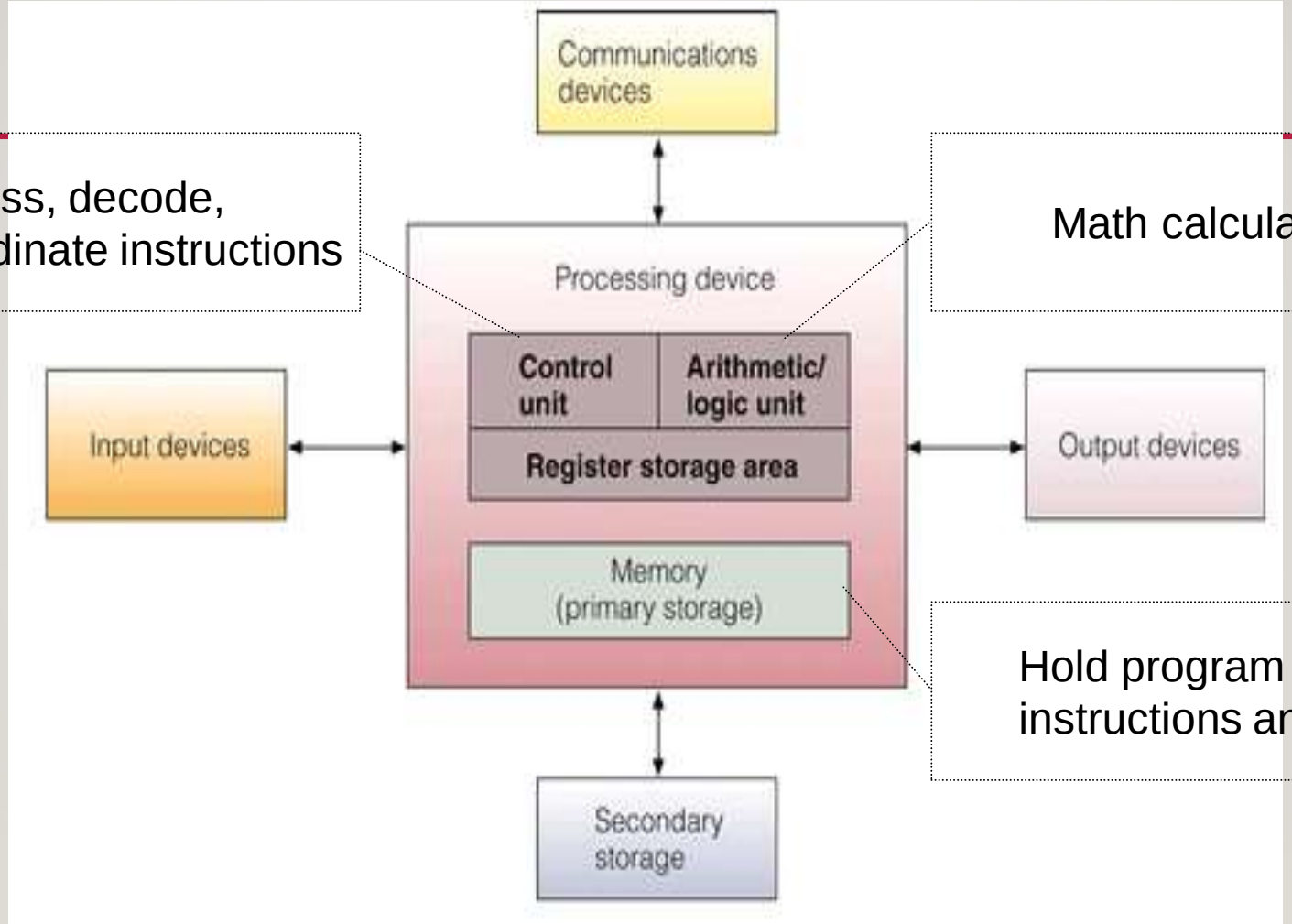
COMPUTER ORGANIZATION

- Six logical units in every computer:
  - Input unit
    - Obtains information from input devices (keyboard, mouse)
  - Output unit
    - Outputs information (to screen, to printer, to control other devices)
  - Memory unit
    - Rapid access, low capacity, stores input information
  - Arithmetic and logic unit (ALU)
    - Performs arithmetic calculations and logic decisions
  - Central processing unit (CPU)
    - Supervises and coordinates the various components of the computer
  - Secondary storage unit
    - Cheap, long-term, high-capacity storage
    - Stores inactive programs

BLOCK DIAGRAM OF COMPUTER



COMPUTER SYSTEM COMPONENTS



Access, decode, coordinate instructions

Math calculations

Hold program instructions and data

INPUT DEVICES

- Input Devices enable the user to enter data into memory
- Examples of input devices:
  - Keyboard
  - Mouse
  - Scanner
  - Touch Screen Input
  - OCR

THE PROCESSOR UNIT

- The Processor Unit is comprised of two components:
  - Central Processing Unit (CPU)
  - Memory

CPU

- Interprets instructions to the computer
- Performs logical and arithmetic operations
- Causes the input and output operations to occur
- A Pentium Pro Microprocessor can perform approximately 250 million instructions per second (MIPS)

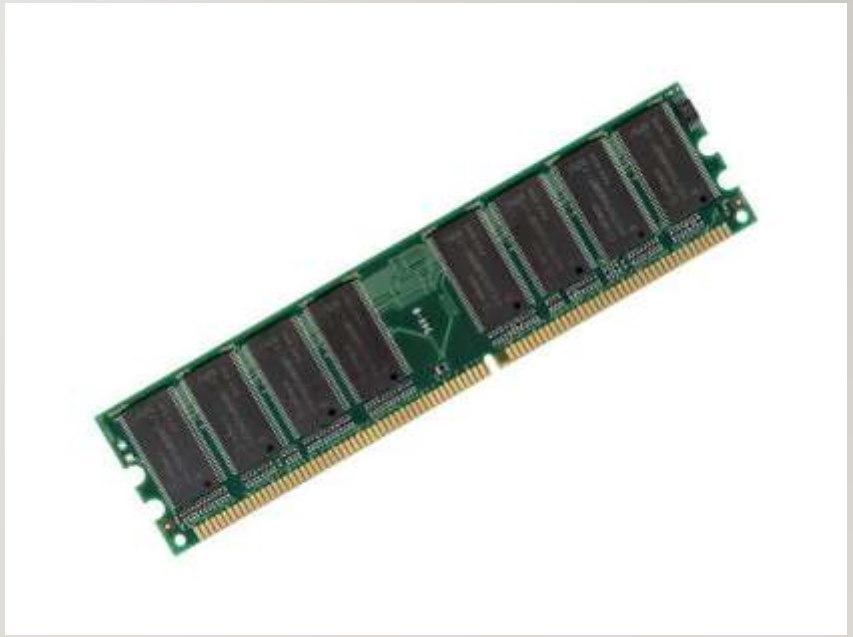
CPU TYPES

- Intel, Advanced Micro Devices (AMD), Motorola, Cyrix

1	AMD A8-3870k
2	AMD FX-8159
3	Intel Core i3 2120
4	Intel Core i5 2400S
5	Intel Core i5 2500K
6	Intel Core i7 3960X
7	Intel Core i7 3960X @ 4.2 GHz
8	Intel Core i7 3770K
9	Intel Core i7 3770K @ 4.3 GHz

COMPUTER MEMORY

- RAM - Random Access Memory
  - Computer's primary storage of data to be processed
  - Silicon chips that store data and instructions as electronic currents
  - Contents of RAM will be lost when power is turned off

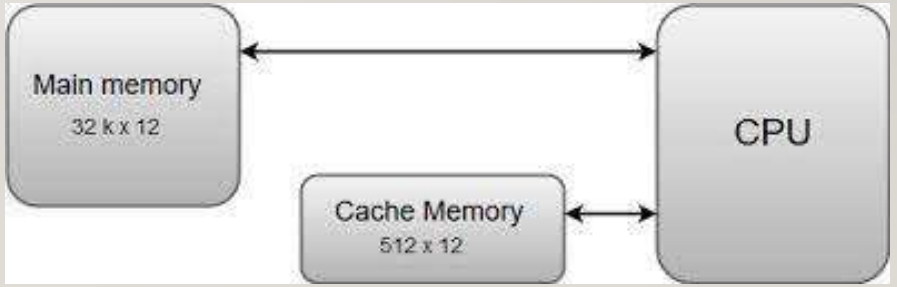


COMPUTER MEMORY

- ROM - Read Only Memory
  - Instructions and data are hard coded on the silicon chips
  - Examples: BIOS (Basic Input-Output System)
    - gives computer the initial instructions to get it started once computer is turned on.

CACHE

- Temporary holding area where the system stores frequently accessed information
- Allows the processor to operate faster
- Cache associated with the terms L1 or L2
  - L1 is internal to the microprocessor
  - L2 is separate from the microprocessor



OUTPUT DEVICES

- Output Devices make the information resulting from processing available for use
- Examples of Output Devices:
  - Printers
  - Computer Screens
  - Speakers

OUTPUT DEVICES

MONITOR

PRINTER

SPEAKER

HEADPHONE

PROJECTOR

AUXILIARY DEVICES

- Also know as Secondary Storage Devices
- Examples of Auxiliary Devices:
  - Floppy Disks
  - Hard Drives
  - CD-ROM
  - Tape Backup Drives
  - ZIP Drives

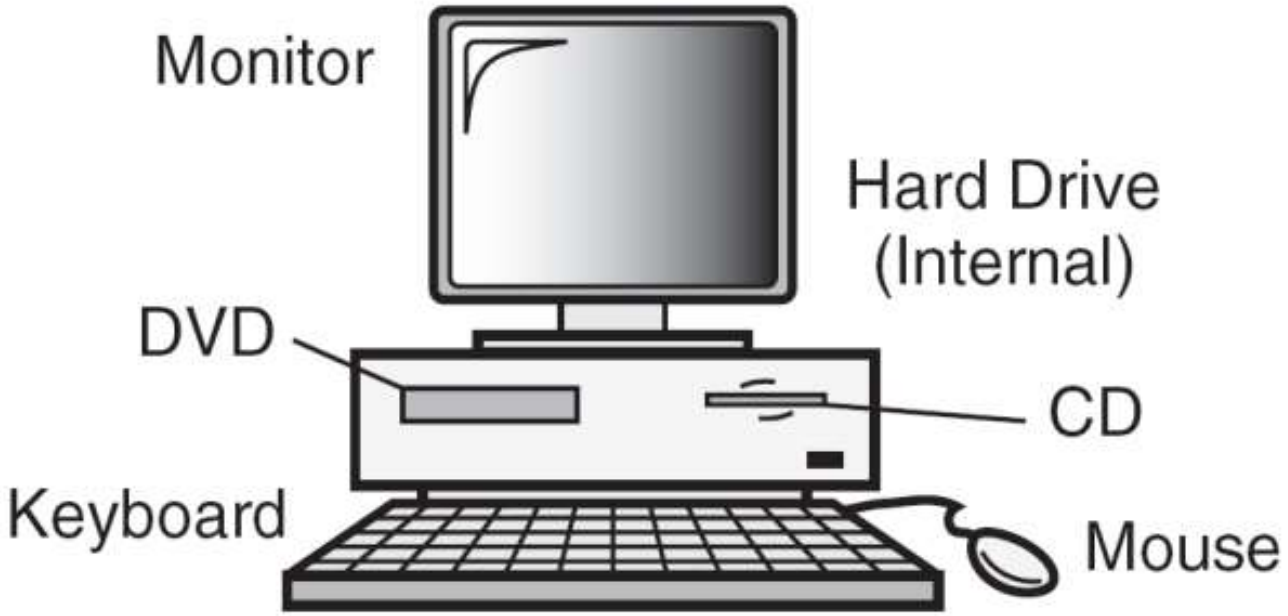

SSD
512GB

COMPUTING ENVIRONMENTS

- There are four computing environments:
  - 1) Personal Computing Environment
  - 2) Time-Sharing Environment
  - 3) Client/Server Environment
  - 4) Distributed Computing

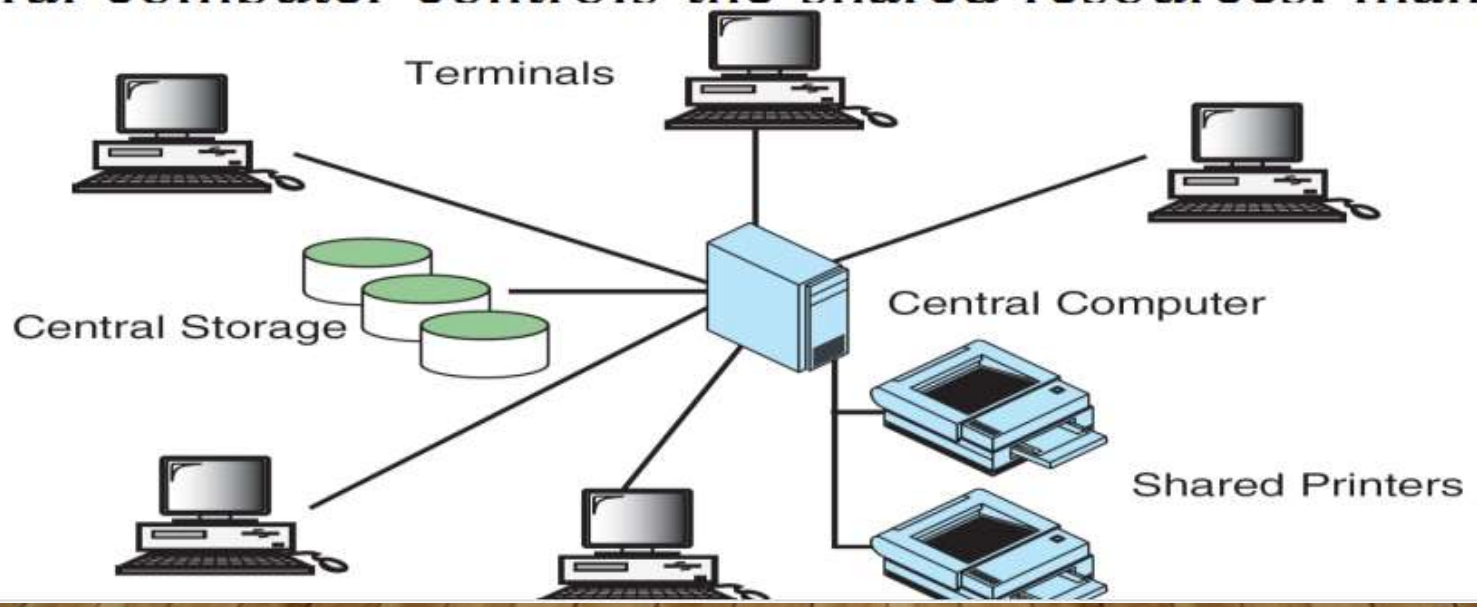
Personal Computing Environment

All of the computer hardware components are tied together in the personal computer. The whole computer is available to the user.



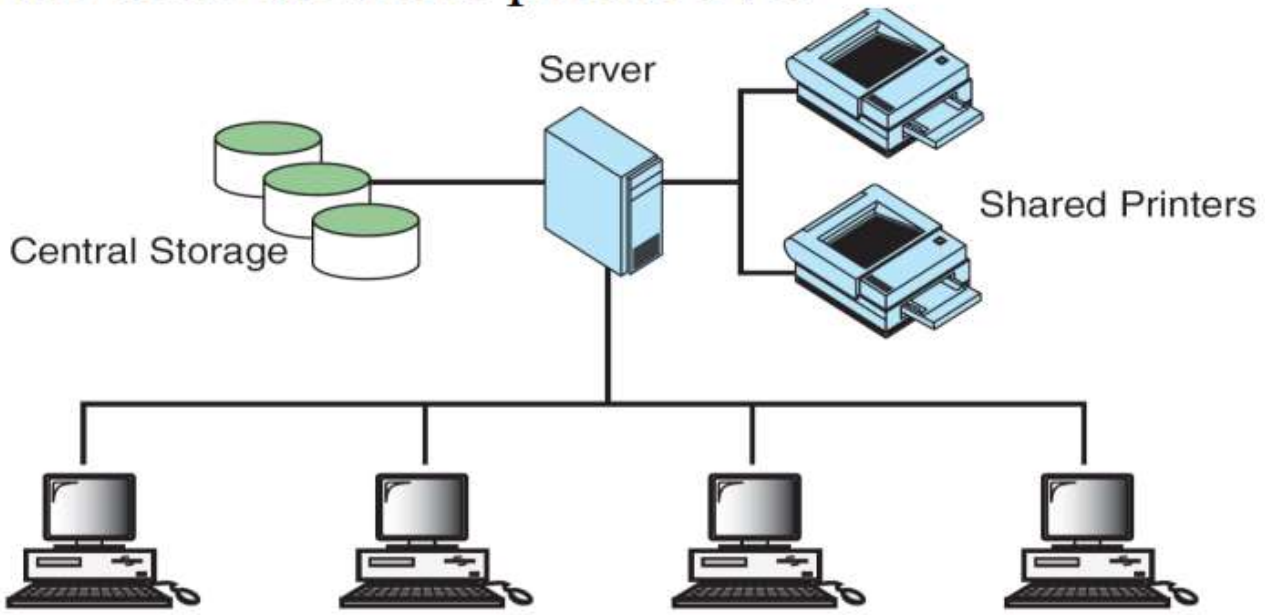
Time-Sharing Environment

- Many users are connected to a computer system. The terminals used are often non-programmable.
  - The output devices and auxiliary storage devices are shared by all of the users.
  - In the time-sharing environment, all computing is done by the central computer.
  - Central computer controls the shared resources, manages the



Client/Server Environment

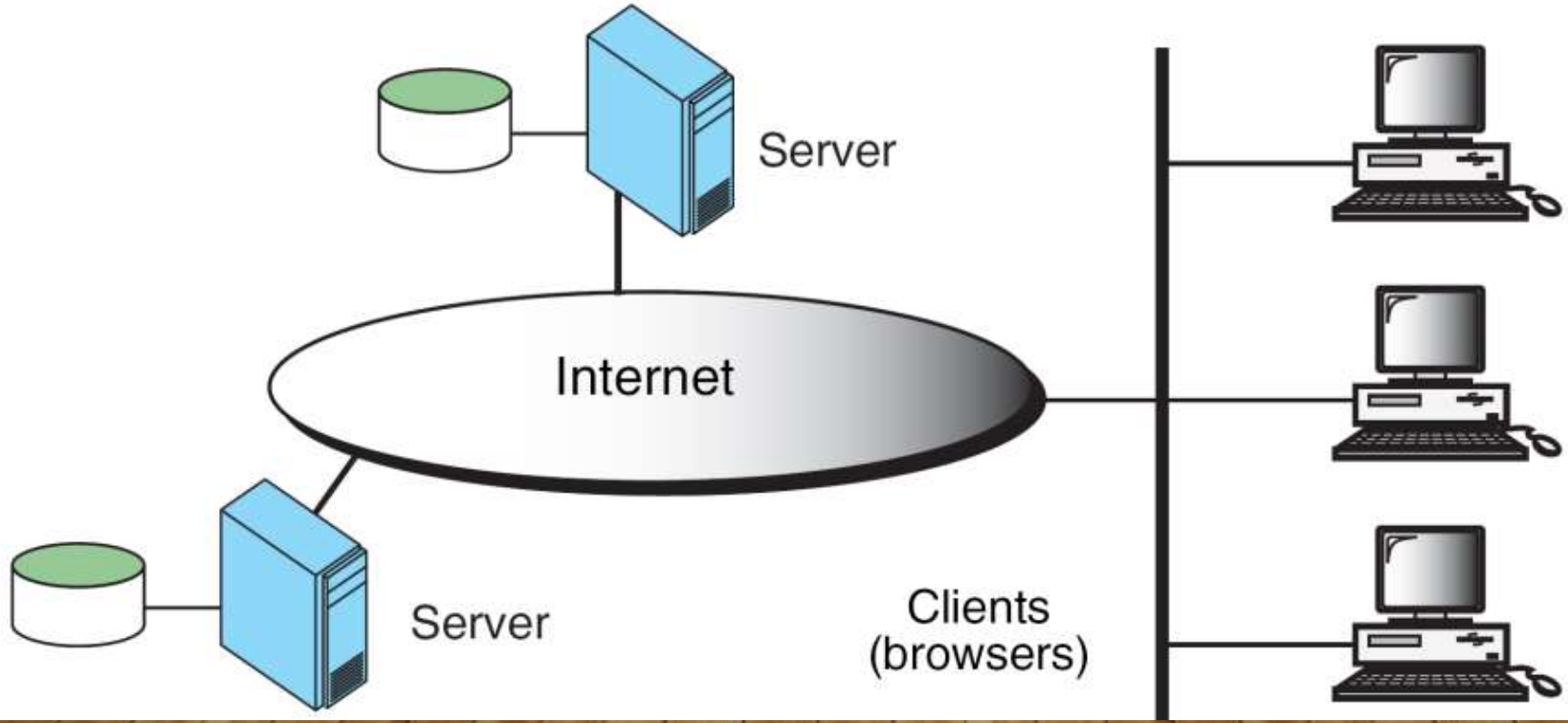
- A client /server computing environment splits the computing function between a central computer and users' computers.
- Some of the computational responsibility is moved from the central computer and assigned to the client computers.
- The central computer called server that manages the shared data and does some part of the computing
- Because of the work sharing, the response time and monitor display are faster and the users are more productive.



Distributed Computing

A distributed computing environment provides integration of computing functions between different clients and servers.

Distributed computing utilizes a network of many computers, each accomplishing a portion of an overall task, to achieve a computational result much more quickly than with a single computer.

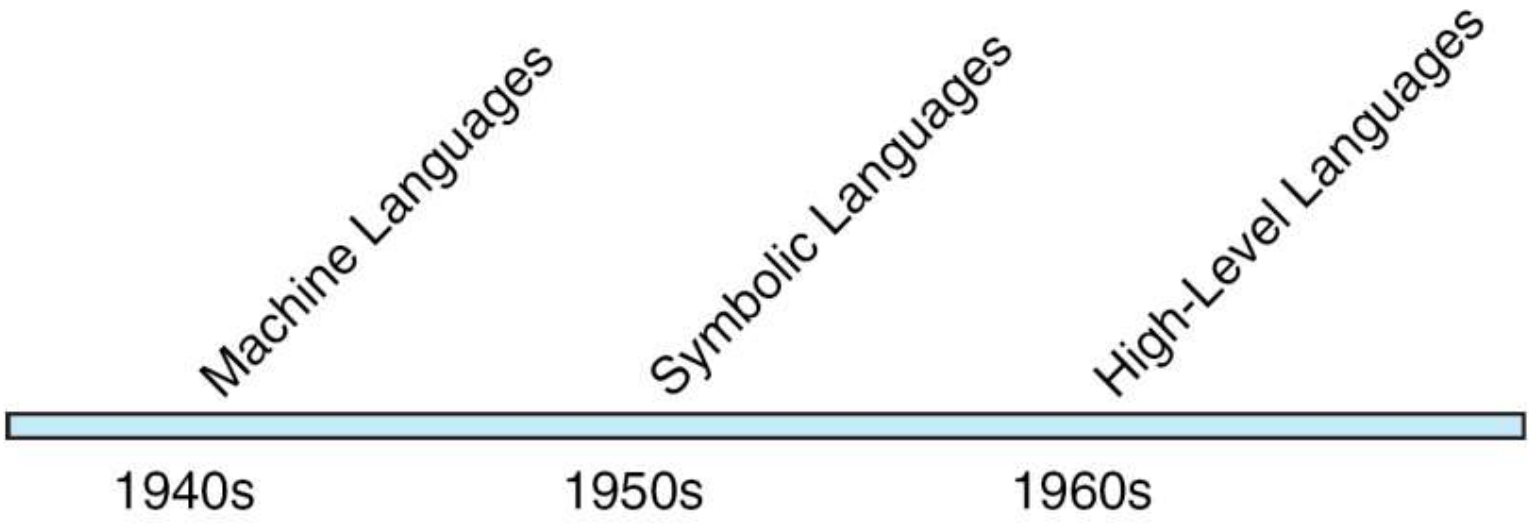


Computer Languages

To write a program for a computer, we must use a computer language.

- There are three types of Computer Languages

1) Machine Languages
2) Symbolic Languages
3) High-Level Languages



Machine Languages

- Each Computer has its own machine language which is made up of 0"s and 1's.

- The instructions in machine language must be in streams of 0's and 1's because internal circuits of a computer are made of devices that can be in one of two states: on or off.

- Machine language is highly difficult to program as it needs the complete knowledge of the computer's instructions
- Difficult and Rarely used
- NOTE:

  The only language understood by computer hardware is machine language.

Symbolic Languages

- The computer operations are represented in symbols or mnemonics to represent various machine language instructions .

  Ex: Add instead of 0001

- These languages are also called Assembly Languages.
- Each assembly language instruction translates to one machine language instruction
- Programming is easier in Assembly language compared to developing programs in machine language
- Symbolic languages are machine dependent
- Assemblers convert the assembly language instructions to machine language instructions

High-level Languages

- The need to improve programmer efficiency and to change the focus from the computer to the problem being solved led to the development of high-level languages
- High level languages are portable to many different computers, allowing the programmers to concentrate on the application problem rather than on the computer
- Compilers are used to convert high-level language programs to machine language programs
- The process of converting high-level language into machine language is called compilation.
- One high-level language statement can get converted to one or more machine language statements

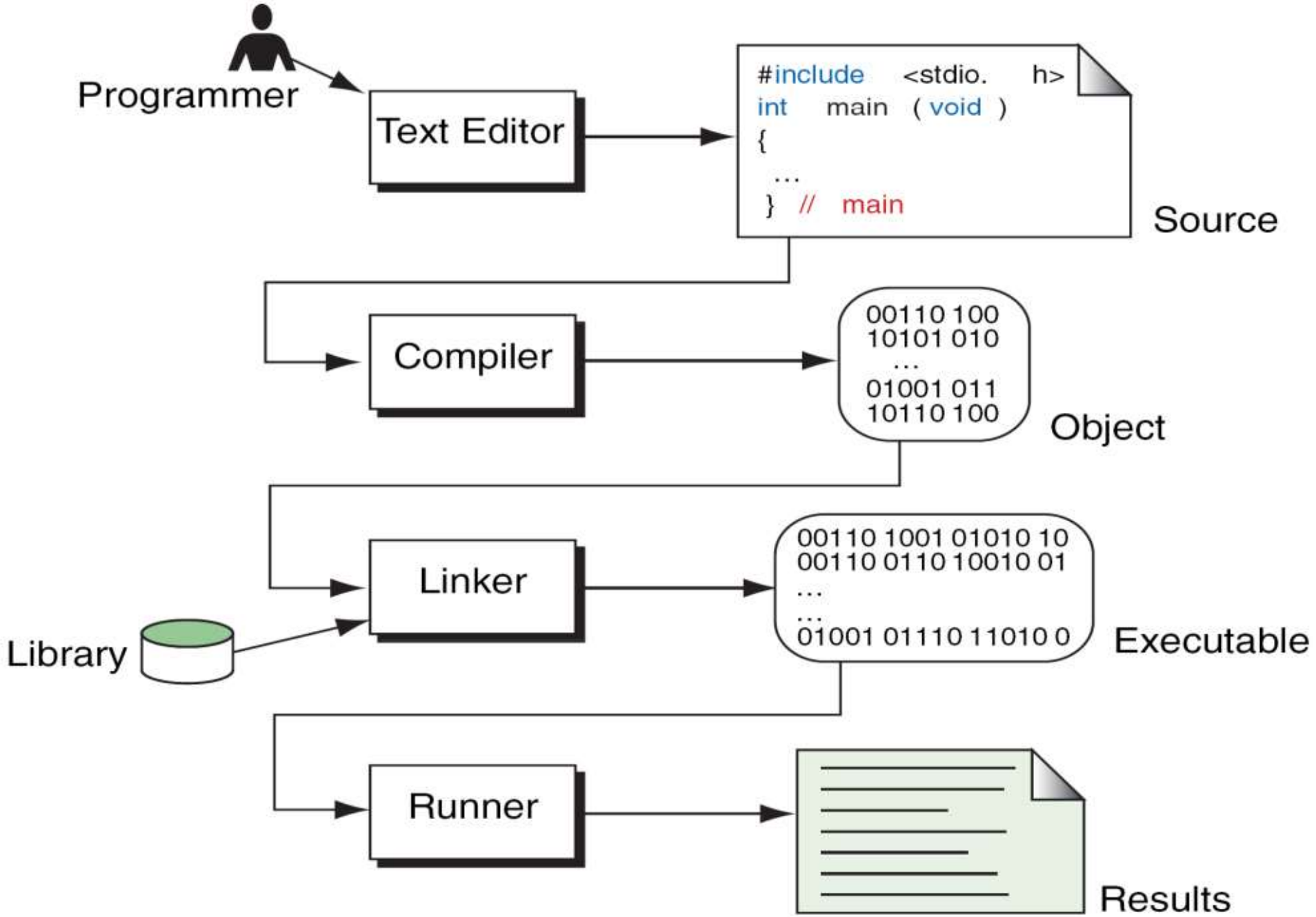
Example

A Statement A= A+B in Different Languages

Machine Language	Assembly Language	High Level Language
0000 0000	CLA	A= A+B
0001 0101	ADD A	
0001 0110	ADD B	
0011 0101	STA A	

CREATING AND RUNNING PROGRAMS

- The steps involved in Creating and Running Programs are:
- 1) Writing and Editing Programs
- 2) Compiling Programs
- 3) Linking Programs
- 4) Executing Programs


Programmer
Text Editor
#include <stdio. h>
int main (void)
{
...
} // main
Source
Compiler
00110 100
10101 010
...
01001 011
10110 100
Object
Linker
Library
00110 1001 01010 10
00110 0110 10010 01
...
...
01001 01110 11010 0
Executable
Runner
Results

WRITING AND EDITING PROGRAMS:

- To solve a particular problem a Program has to be created as a file using text editor / word processor. This is called source file.
- The program has to be written as per the structure and rules defined by the high-level language that is used for writing the program (C, JAVA etc)

COMPILING PROGRAMS:

- The compiler corresponding to the high-level language will scan the source file, checks the program for the correct grammar (syntax) rules of the language.
- If the program is syntactically correct, the compiler generates an output file called Object File‘ which will be in a binary format and consists of machine language instructions corresponding to the computer on which the program gets executed.

COMPILING PROGRAMS:

- If the source program contains syntax errors, the compiler lists these errors and will not generate the object file. The program is to be corrected for the errors and recompiled.
- The object file contains references to other programs which will be needed for the execution of the program. These programs are called library functions. These programs are to be combined with the Object File.

LINKING PROGRAMS:

- Linker program combines the Object File with the required library functions to produce another file called — executable filell.
- Object file will be the input to the linker program.
- The executable file is created on disk.
- This file has to be put into (loaded) the memory

EXECUTING PROGRAMS:

- Loader program loads the executable file from disk into the memory and directs the CPU to start execution.
- The CPU will start execution of the program that is loaded into the memory .
- During Program Execution, the program reads data for processing, either from the user (key-board) or from a file.
- After the program processes the data, it prepares the output.
- Output can be to the user‘s monitor or to a file.
- When the program has finished its job, it informs the Operating System.
- OS then removes the program from memory.

WHAT IS A COMPUTER?

A DIGITAL COMPUTER IS AN ELECTRONIC PROGRAMMABLE MACHINE THAT CAN PROCESS ALMOST ALL KINDS OF DATA.

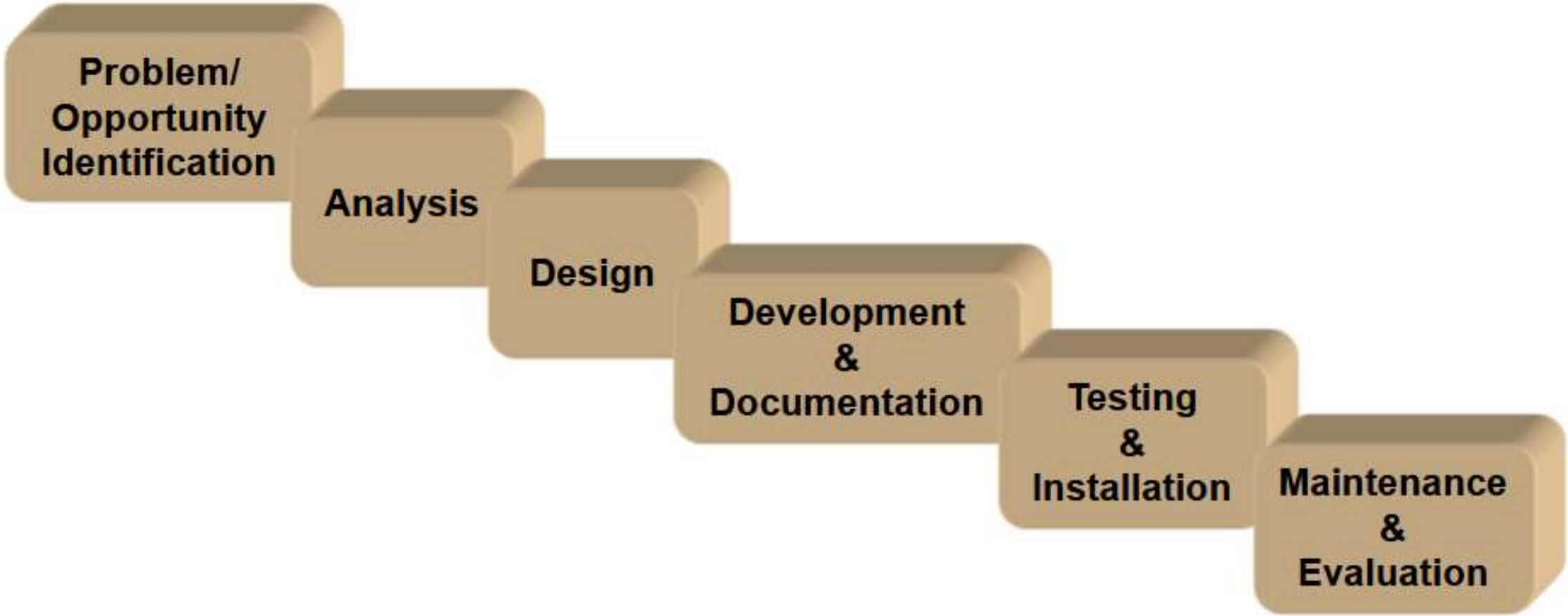
What is a Program?

It is a set of instructions, written in a particular sequence in a computer-related language.

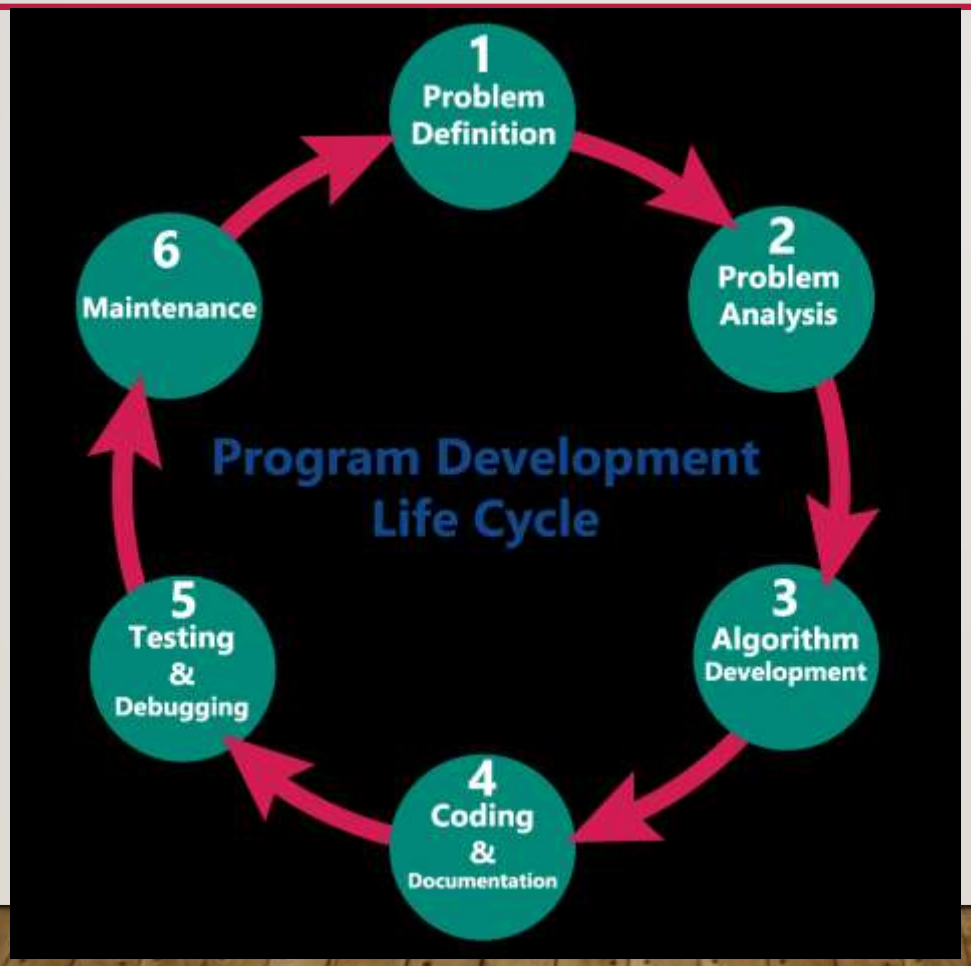
SYSTEMS

- A collection of pieces working together to achieve a common goal
- Information system includes:
  - Data
  - People
  - Procedures
  - Hardware
  - Software
- System development life cycle (SDLC):
  - An organized process used to develop systems in an orderly fashion

System Development Life Cycle



PROGRAM DEVELOPMENT LIFE CYCLE



PROGRAM DEVELOPMENT LIFE CYCLE

- **Analyze the problem.** The computer user must figure out the problem, then decide how to resolve the problem - choose a program.
- **Design the program.** A flow chart is important to use during this step of the PDLC. This is a visual diagram of the flow containing the program. This step will help you break down the problem.
- **Code the program.** This is using the language of programming to write the lines of code. The code is called the listing or the source code. The computer user will run an object code for this step.

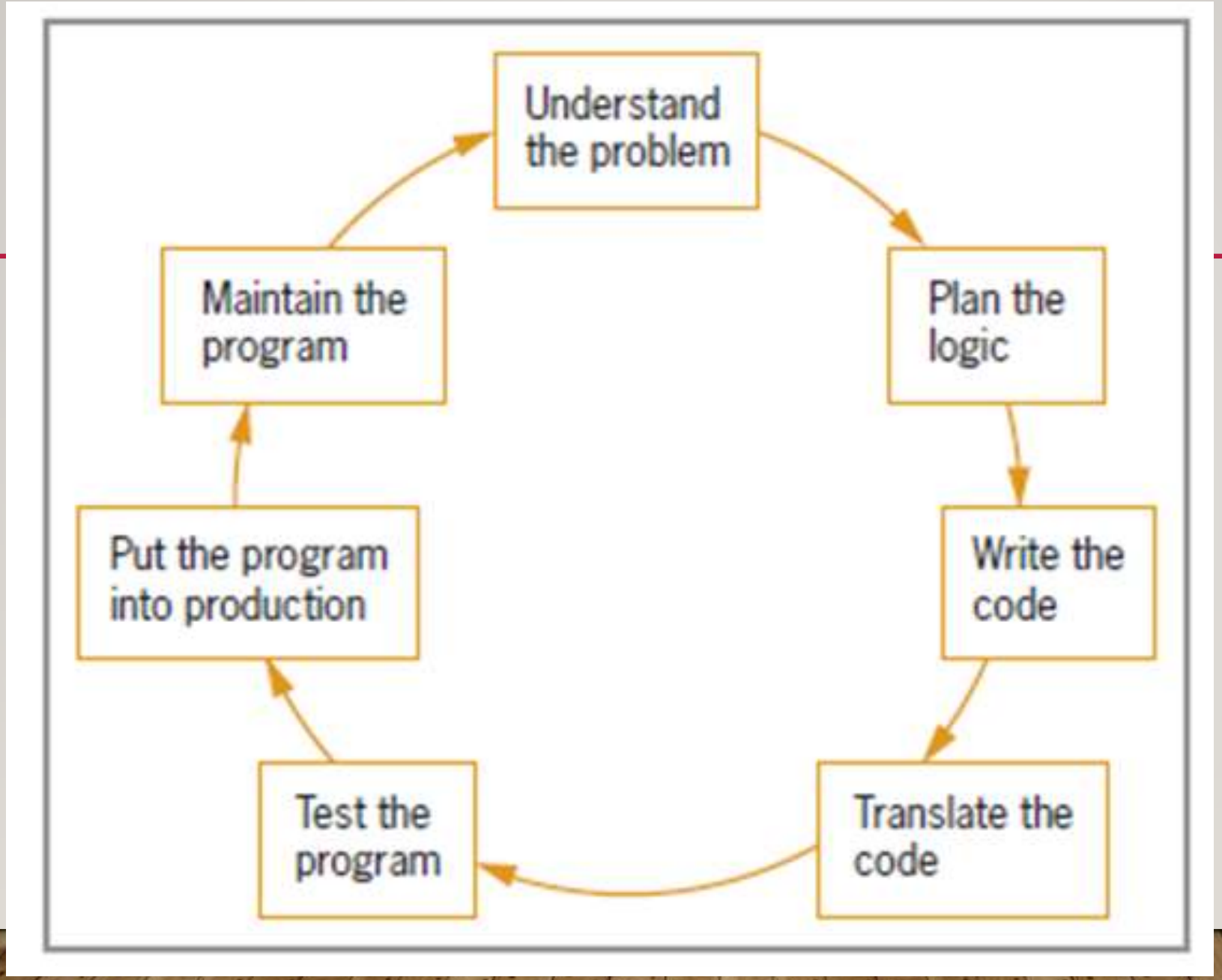
PROGRAM DEVELOPMENT LIFE CYCLE

- **Debug the program.** The computer user must debug. This is the process of finding the "bugs" on the computer. The bugs are important to find because this is known as errors in a program.
- **Formalize the solution.** One must run the program to make sure there are no syntax and logic errors. Syntax are grammatical errors and logic errors are incorrect results.
- **Document and maintain the program**. This step is the final step of gathering everything together. Internal documentation is involved in this step because it explains the reasoning one might of made a change in the program or how to write a program

The Life Cycle of a Program

- Programming is the process of translating a task into a series of commands a computer will use to perform that task
- Programming involves:
  - Identifying the parts of a task the computer can perform
  - Describing tasks in a specific and complete manner
  - Translating the tasks into a language that is understood by the computer's CPU

NEXT


Understand the problem
Plan the logic
Write the code
Translate the code
Test the program
Put the program into production
Maintain the program

ALGORITHM

- Precise step-by-step plan for a computational procedure that begins with an input value and yields an output value in a finite number of steps.
- It is an effective method which uses a list of well-defined instructions to complete a task, starting from a given initial state to achieve the desired end state.
- An algorithm is written in simple English and is not a formal document

ALGORITHM

Read n;

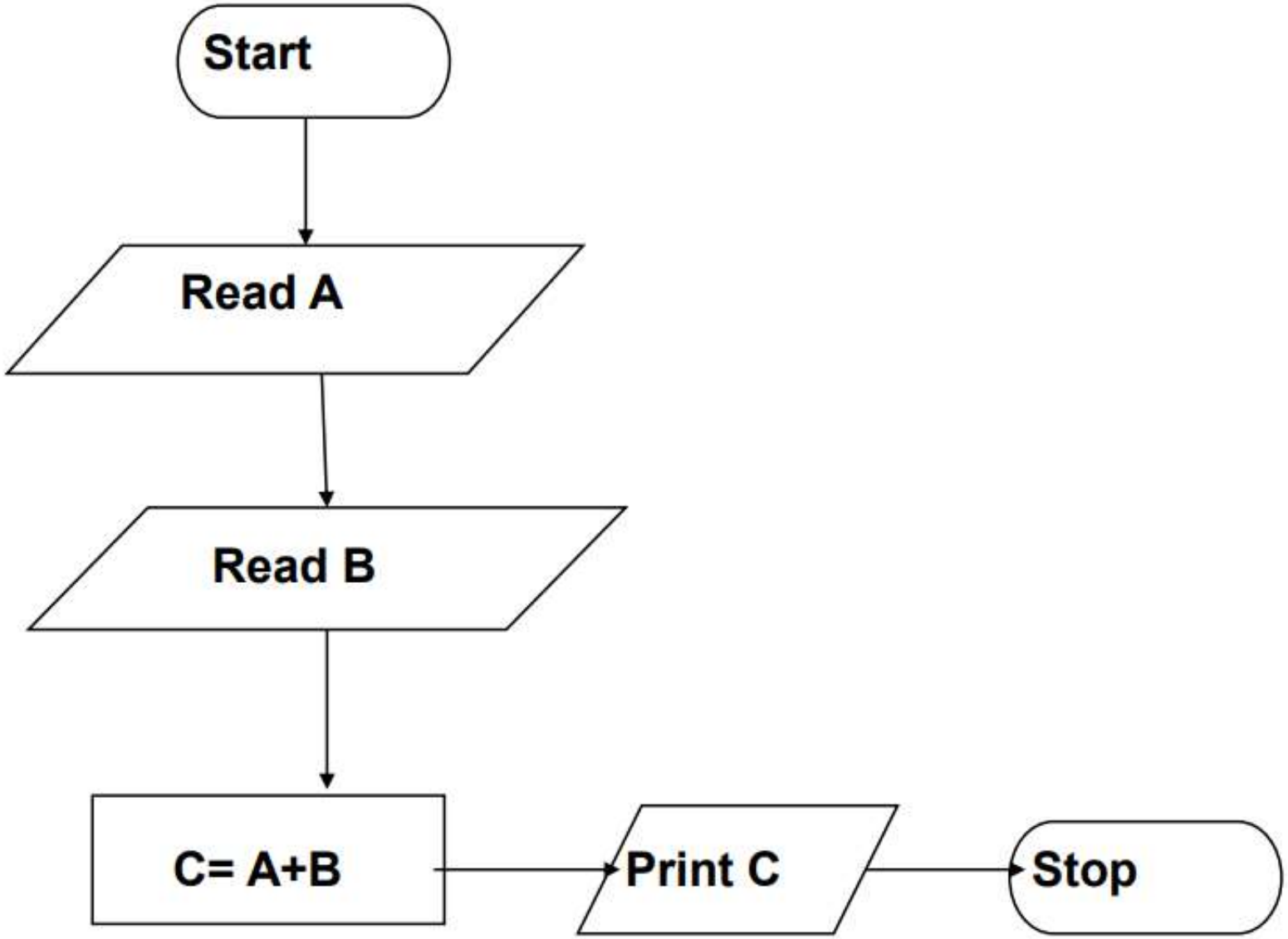
For i=1 to n add all values of A[i] in sum;

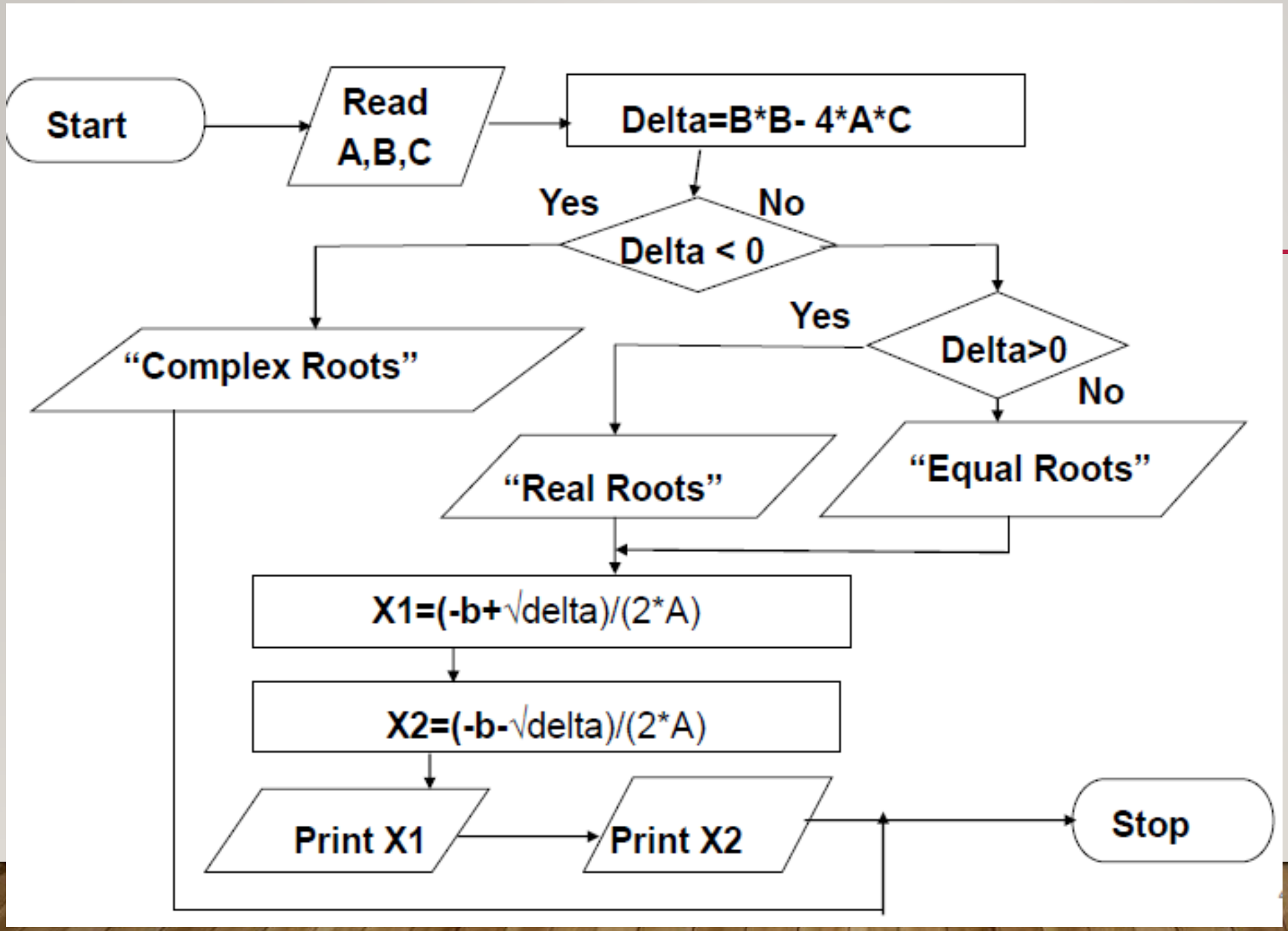
Print sum/n;

- An algorithm must:
  - Be lucid, precise and unambiguous
  - Give the correct solution in all cases
  - Eventually end
- it is important to use indentation when writing solution in algorithm because it helps to differentiate between the different control structures.

- Example 1: Finding the sum of two numbers.
- – Variables:
  - – • A: First Number
  - – • B: Second Number
  - – • C: Sum (A+B)
- – Algorithm:
  - – • Step 1 – Start
  - – • Step 2 – Input A
  - – • Step 3 – Input B
  - – • Step 4 – Calculate C = A + B
  - – • Step 5 – Output C
  - – • Step 6 – Stop

Flowcharts:

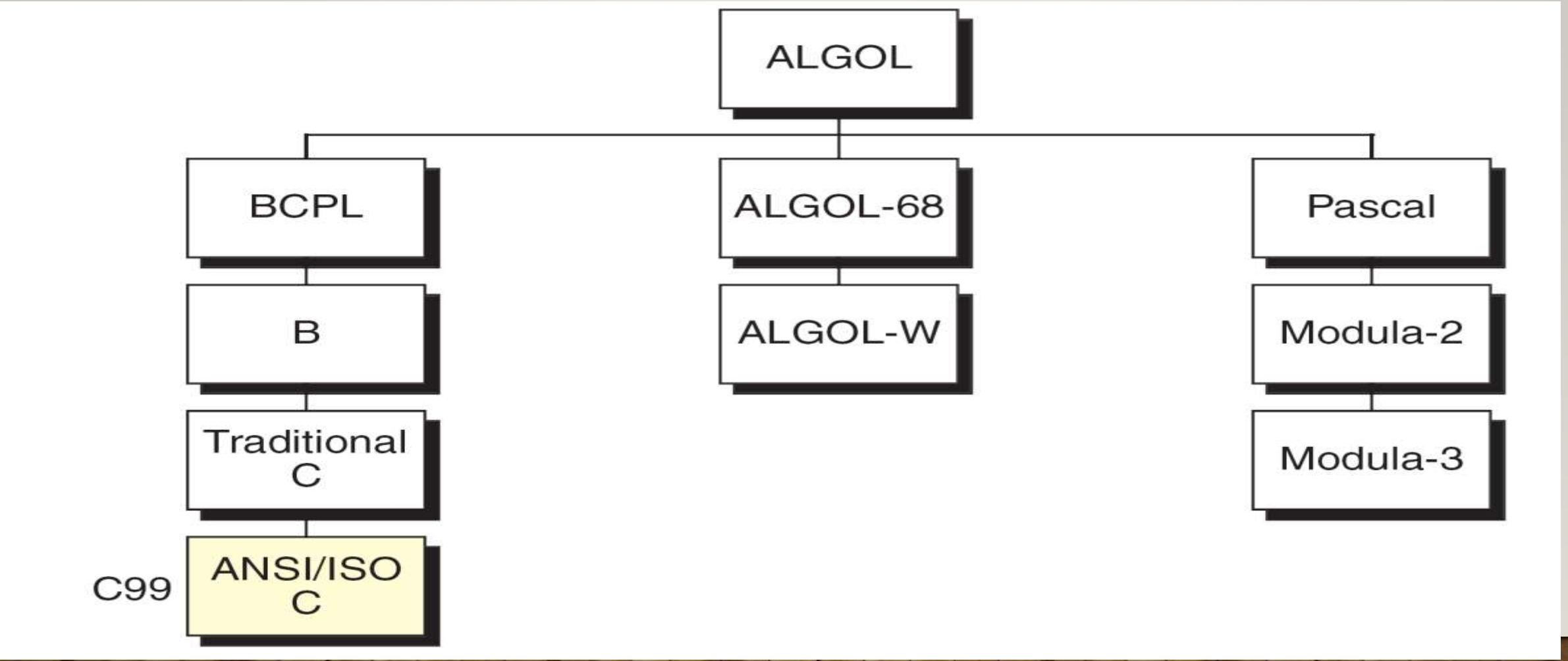



Start
Read
A,B,C
Delta=B*B- 4*A*C
Yes
No
Delta < 0
"Complex Roots"
Yes
Delta>0
No
"Real Roots"
"Equal Roots"
X1=(-b+√delta)/(2*A)
X2=(-b-√delta)/(2*A)
Print X1
Print X2
Stop

INTRODUCTION TO C :

- A high-level programming language developed in 1972 by Dennis Ritchie at AT&T Bell Labs.
- C is a structured Language
- It was designed as a language to develop UNIX operating system
- American National Standards Institute(ANSI) approved first version of C in 1989 which called C89
- Major changes were made in 1995 and is known as C95
- Another update was made in 1999 and is known as C99

INTRODUCTION TO C :



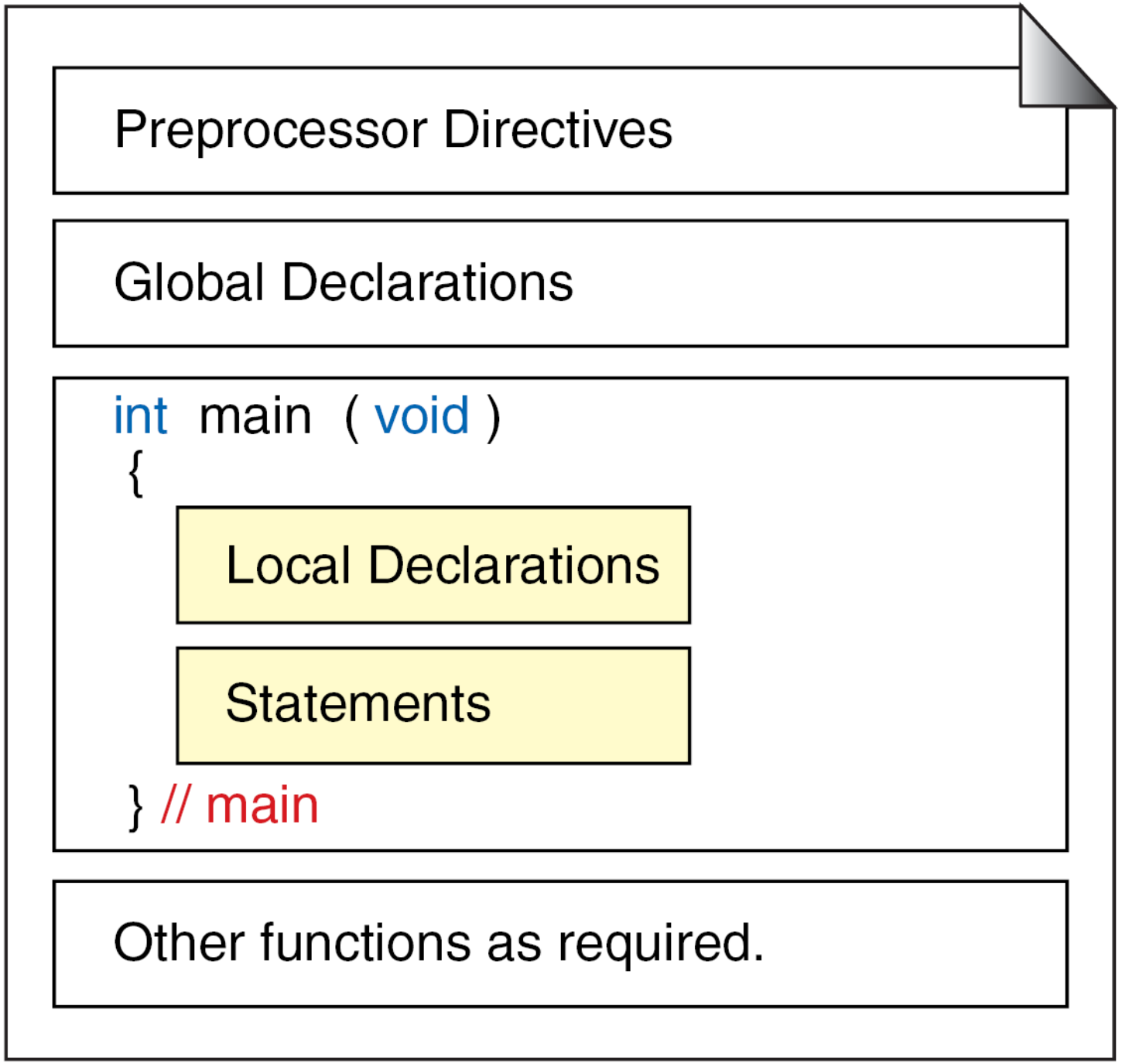
CHARACTERISTICS OF C

- A high level programming language .
- Small size. C has only 32 keywords. This makes it relatively easy to learn.
- Makes extensive use of function calls.
- C is a structured programming.
- It supports loose typing (as a character can be treated as an integer and vice versa).

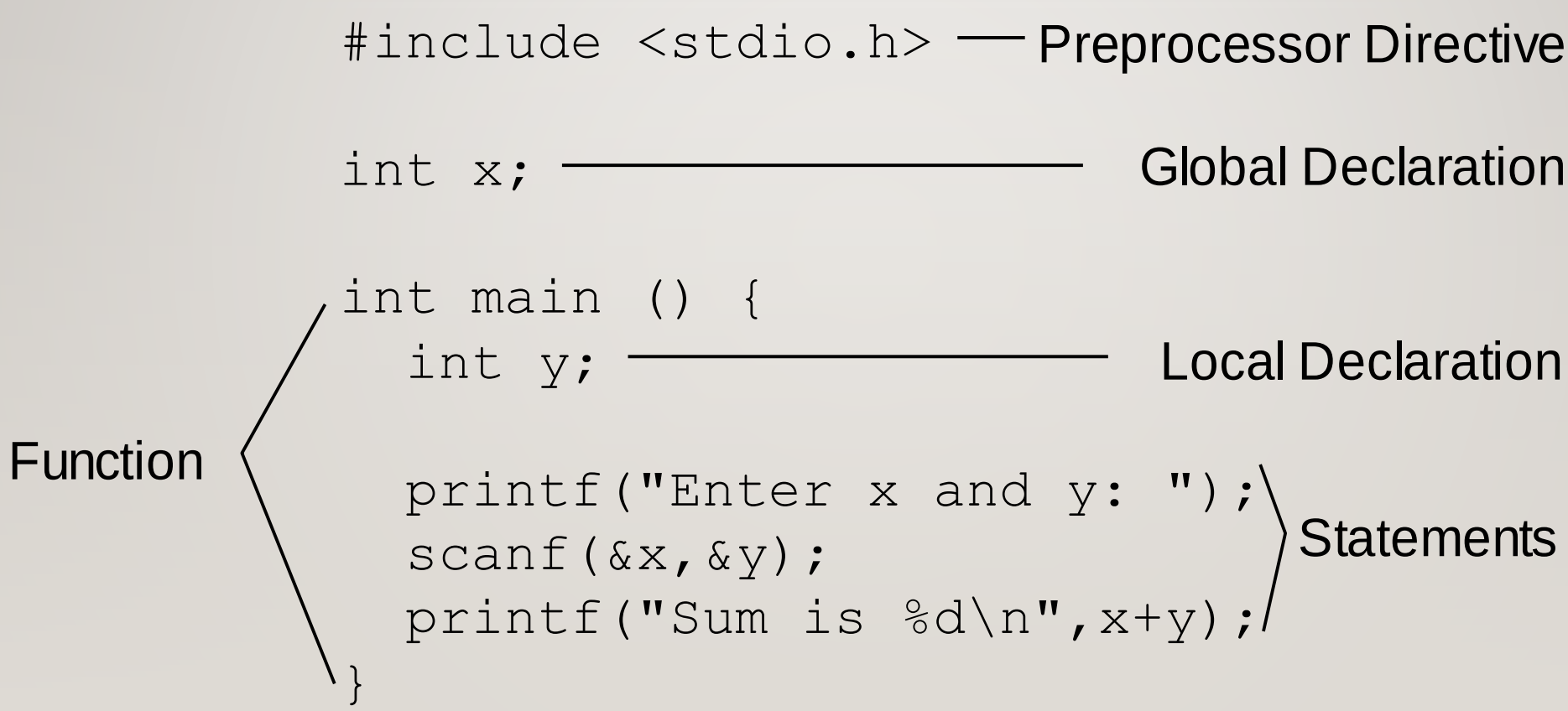
CHARACTERISTICS OF C

- Facilitates low level (bitwise) programming
- Supports pointers to refer computer memory, array, structures and functions.
- C is a Portable language.
- C is a core language
- C is an extensible language

STRUCTURE OF C PROGRAM



PARTS OF A PROGRAM



PREPROCESSOR DIRECTIVES

- Begin with #
- Instruct compiler to perform some transformation to file before compiling
- Example: #include <stdio.h>
  - add the *header* file stdio.h to this file
  - .h for header file
  - stdio.h defines useful input/output functions

DECLARATIONS

- Global
  - visible throughout program
  - describes data used throughout program
- Local
  - visible within function
  - describes data used only in function

FUNCTIONS

- Consists of *header* and *body*
  - header: int main ()
  - body: contained between { and }
    - starts with location declarations
    - followed by series of statements
- More than one function may be defined
- Functions are *called* (invoked) - more later

MAIN FUNCTION

- Every program has one function **main**
- Header for main: int main ()
- Program is the sequence of statements between the { } following main
- Statements are executed one at a time from the one immediately following to main to the one before the }

COMMENTS

- Text between /* and */
- Used to "document" the code for the human reader
- Ignored by compiler (not part of program)
- Have to be careful
  - comments may cover multiple lines
  - ends as soon as */ encountered

COMMENT EXAMPLE

```
#include <stdio.h>

/* This comment covers
 * multiple lines
 * in the program.
 */

int main () /* The main header */ {
  /* No local declarations */

  printf("Too many comments\n");
} /* end of main */
```

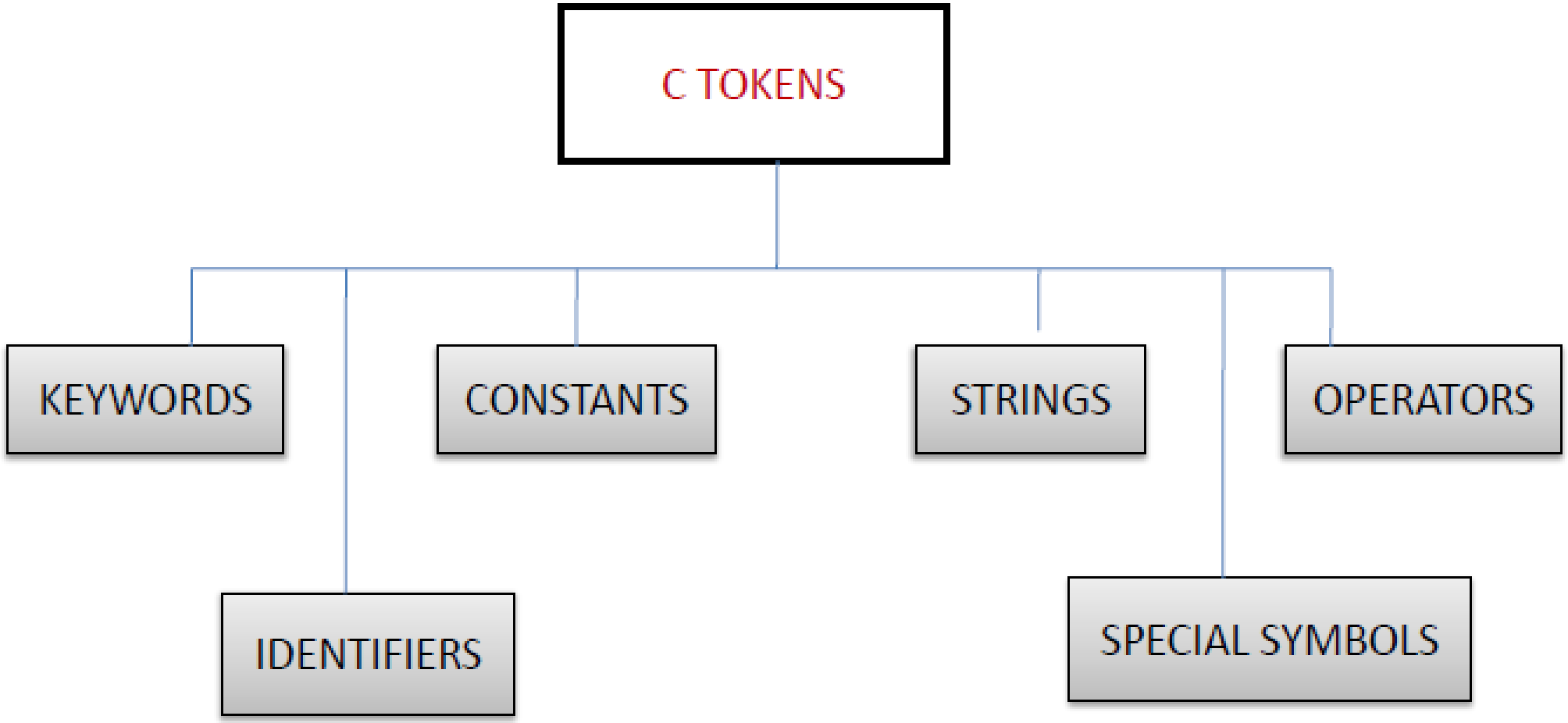
DOCUMENTATION

- Global - start of program, outlines overall solution, may include structure chart
- Module - when using separate files, indication of what each file solves
- Function - inputs, return values, and logic used in defining function
- Add documentation for key (tough to understand) comments
- Names of variables - should be chosen to be meaningful, make program readable

SYNTAX OF C

- Rules that define C language
  - Specify which tokens are valid
  - Also indicate the expected order of tokens
- Some types of tokens:
  - reserved words: include printf int ...
  - identifiers: x y ...
  - literal constants: 5 ‘a’ 5.0 ...
  - punctuation: { } ; < > # /* */

C TOKENS


C TOKENS
KEYWORDS
IDENTIFIERS
CONSTANTS
STRINGS
SPECIAL SYMBOLS
OPERATORS

IDENTIFIER

- Names used for objects in C
- Rules for identifiers in C:
  - first char alphabetic [a-z,A-Z] or underscore (_)
  - has only alphabetic, digit, underscore chars
  - first 31 characters are significant
  - cannot duplicate a reserved word
  - case (upper/lower) matters

RESERVED WORDS

- Identifiers that already have meaning in C
- Examples:
  - include, main, printf, scanf, if, else, …
  - more as we cover C language

VALID/INVALID IDENTIFIERS

Valid	Invalid
sum	7of9
c4_5	x-name
A_NUMBER	name with spaces
longnamewithmanychars	1234a
TRUE	int
_split_name	AXYZ&

VARIABLES

- Named memory location
- Variables declared in global or local declaration sections
- Syntax: *Type Name*;
- Examples:

```
int sum;
float avg;
char dummy;
```

VARIABLE TYPE

- Indicates how much memory to set aside for the variable
- Also determines how that space will be interpreted
- Basic types: char, int, float
  - specify amount of space (bytes) to set aside
  - what can be stored in that space
  - what operations can be performed on those vars

VARIABLE NAME

- Legal identifier
- Not a reserved word
- Must be unique:
  - not used before
  - variable names in functions (local declarations) considered to be qualified by function name
  - variable x in function main is different from x in function f1

MULTIPLE VARIABLE DECLARATIONS

- Can create multiple variables of the same type in one statement:

  int x, y, z;

  is a shorthand for

  int x;

  int y;

  int z;

  - stylistically, the latter is often preferable

VARIABLE INITIALIZATION

- Giving a variable an initial value
- Variables not necessarily initialized when declared (value is unpredictable - *garbage*)
- Can initialize in declaration:
- Syntax: *Type Name* = *Value*;
- Example:

```
int x = 0;
```

MULTIPLE DECLARATION INITIALIZATION

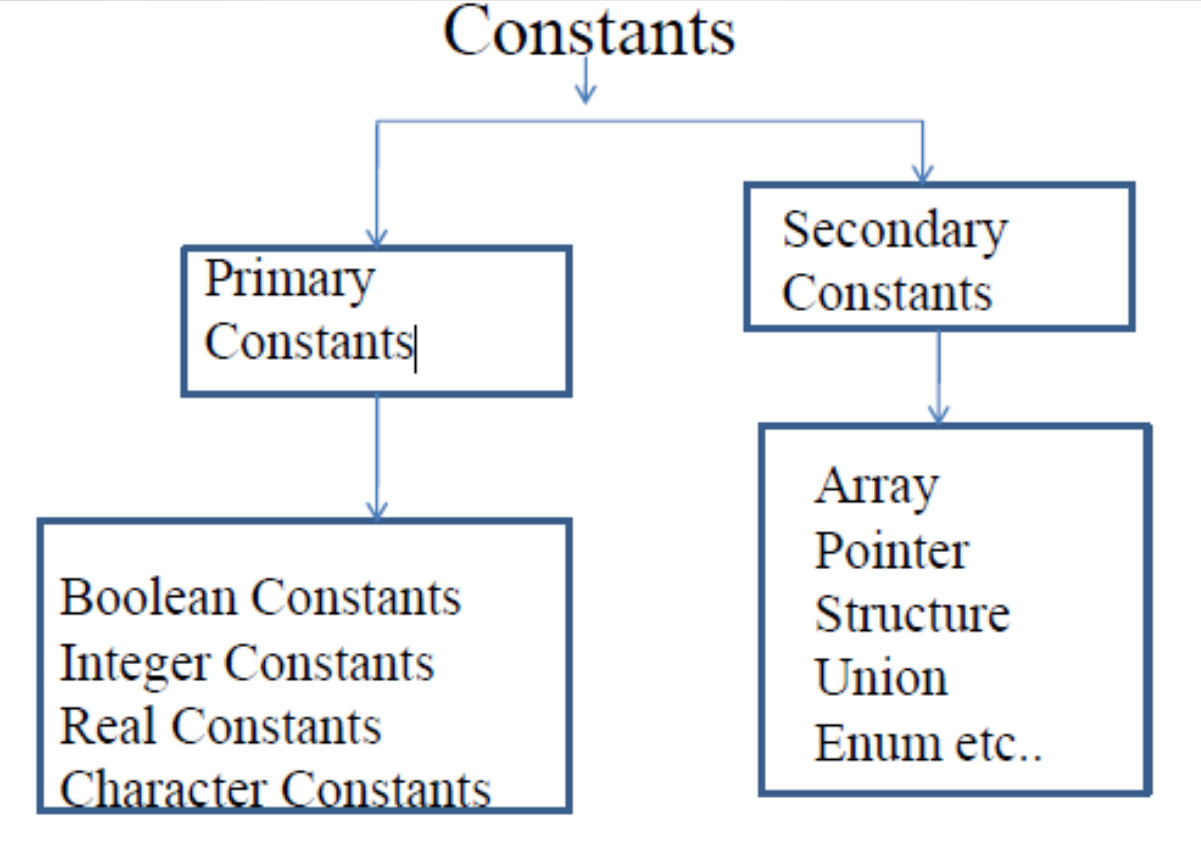
- Can provide one value for variables initialized in one statement:

```
int x, y, z = 0;
```

- Each variable declared and then initialized with the value

CONSTANTS:

- Constants are data values that cannot be changed during the execution of a program.
- C constants can be divided into two major categories:



- Boolean constants
  - A Boolean data type can take only two values. The values are *true* and *false*.
- Integer Constants

Rules for Constructing Integer Constants

- An integer constant must have at least one digit.
- It must not have a decimal point.
- It can be either positive or negative.
- If no sign precedes an integer constant it is assumed to be positive.
- No commas or blanks are allowed within an integer constant.

EXAMPLE OF INTEGER CONSTANS

Representation	Value	Type
+123	123	int
–378	–378	int
–32271L	–32,271	long int
76542LU	76,542	unsigned long int
12789845LL	12,789,845	long long int

Real Constants:

- Rules for Constructing Real Constants
  - Real constants are often called Floating Point constants. It consists of integral part and fractional part. The real constants can be in
    - Fractional form
    - Exponential form.
  - In Fractional Form
    - A real constant must have at least one digit.
    - It must have a decimal point.
    - It could be either positive or negative.
    - Default sign is positive.
    - No commas or blanks are allowed within a real constant.

EXAMPLES OF REAL CONSTANTS

Representation	Value	Type
0.	0.0	double
.0	0.0	double
2.0	2.0	double
3.1416	3.1416	double
–2.0f	–2.0	float
3.1415926536L	3.1415926536	long double

- Exponential Form
  - In exponential form of representation, the real constant is represented in two parts. The part appearing before 'e' is called mantissa, whereas the part following 'e' is called exponent.

➢ Rules for constructing real constants expressed in exponential form:

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa and exponent part may have a positive or negative sign.
- Default sign of mantissa part is positive.
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

Ex: $1.23 \times 10^5 = 123000.0$ is written as 1.23e5 or 1.23E5

- 0.34e-4 = 0.000034

Character Constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within two single quotes (apostrophes).
- The maximum length of a character constant can be 1 character.
- The character in the character constant comes from the character supported by the computer.

Ex: `A` `I` `5` `=` `a`

SYMBOLIC NAMES FOR CONTROL CHARACTERS

ASCII Character	Symbolic Name
null character	`'\0'`
alert (bell)	`'\a'`
backspace	`'\b'`
horizontal tab	`'\t'`
newline	`'\n'`
vertical tab	`'\v'`
form feed	`'\f'`
carriage return	`'\r'`
single quote	`'\''`
double quote	`'\"'`
backslash	`'\\'`

CODING CONSTANTS

Literal Constant

- A literal is an unnamed constant used to specify data.
  Ex: a=b+5;
  hear 5 is a literal constant.

Defined constants

- A defined constant uses the preprocessor command #define
  Ex: #define rate 0.85

Preprocessor replaces each defined name, rate with the value 0.85 wherever it is found in the source program

Memory Constants

- Memory constant use a C type qualifier, const to indicate that the data cannot be changed
  Ex:const float PI = 3.14159;

STRINGS

- A string in C is merely an array of characters. The length of a string is determined by a terminating **null** character: '\0' .
- So, a string with the contents, say, "abc" has four characters: 'a' , 'b' , 'c' , and the terminating **null** character. The terminating **null** character has the value zero.

SPECIAL SYMBOLS

- The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.
- [] () {} , ; : * ... = #
- **Braces{}:** These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.
- **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

VARIABLES:

- Variables are named memory locations that have a type, such as integer or character, which is inherited from their type.
- The type determines the values that a variable may contain and the operations that may be used with its values.
- To declare a variable specify data type of the variable followed by its name. Variable declaration always ends with a semicolon
- Variable names should always be meaningful and must reflect the purpose of their usage in the program.

Variable Declaration

Synatax: Type var_name;

Ex:

```
int   emp_num;
float   alary;
char   grade;
double balance_ amount;
unsigned short  int    acc_no;
```

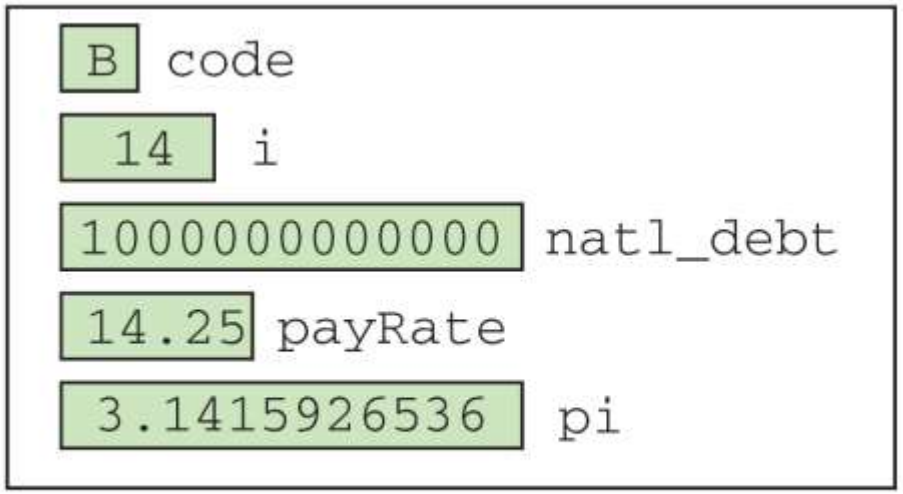
VARIABLE INITIALIZATION

- When a variable is defined, it contains unknown value. The variable has to be initialized with a known value
- If a variable is not initialized, the value of variable may be either 0 or garbage depending on the storage class of the variable.
- We must initialize any variable with known data before executing the function

VARIABLE INITIALIZATION

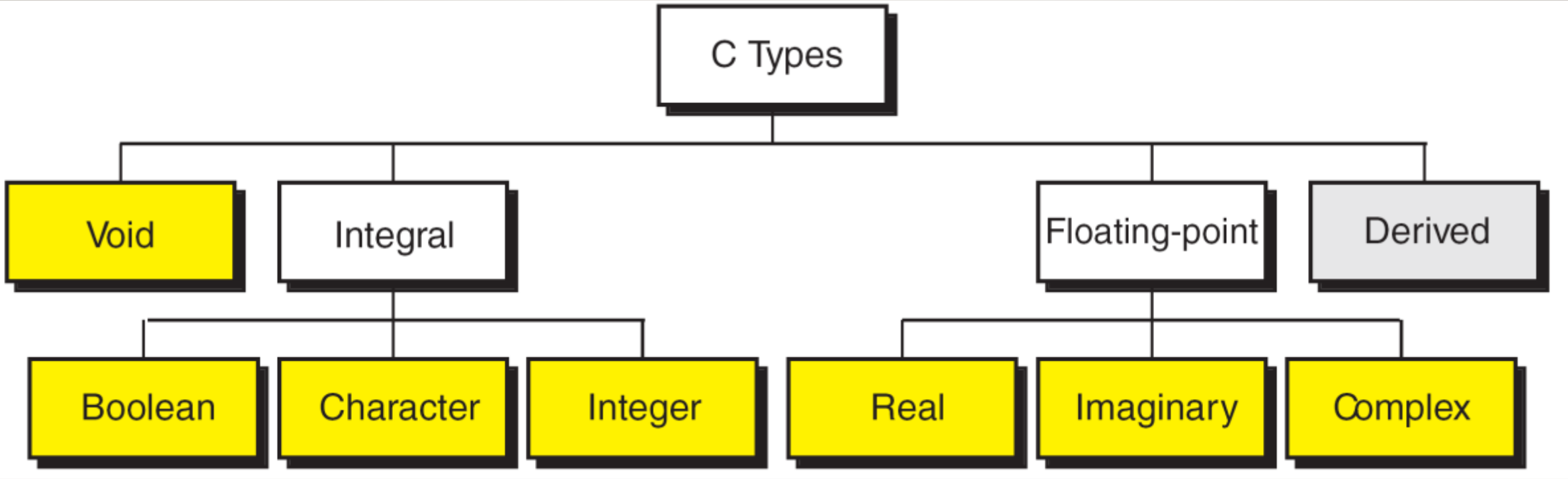
```
char code = 'b';
int  i    = 14;
long long natl_debt = 1000000000000;
float     payRate   = 14.25;
double    pi        = 3.1415926536;
```

Program



Memory

TYPES


C Types
Void
Integral
Floating-point
Derived
Boolean
Character
Integer
Real
Imaginary
Complex

Void Type:

- Is identified by the key word 'void' and no operations.
- It is used to designate that a function has no parameters.
- It can also be used to define that a function has no return value.

Integral Type:

Boolean:

- Boolean type can represent only two values: *true* or *false*
- Referred by the keyword *Bool*
- Is stored in memory as 0 (false) or 1 (true)

Character:

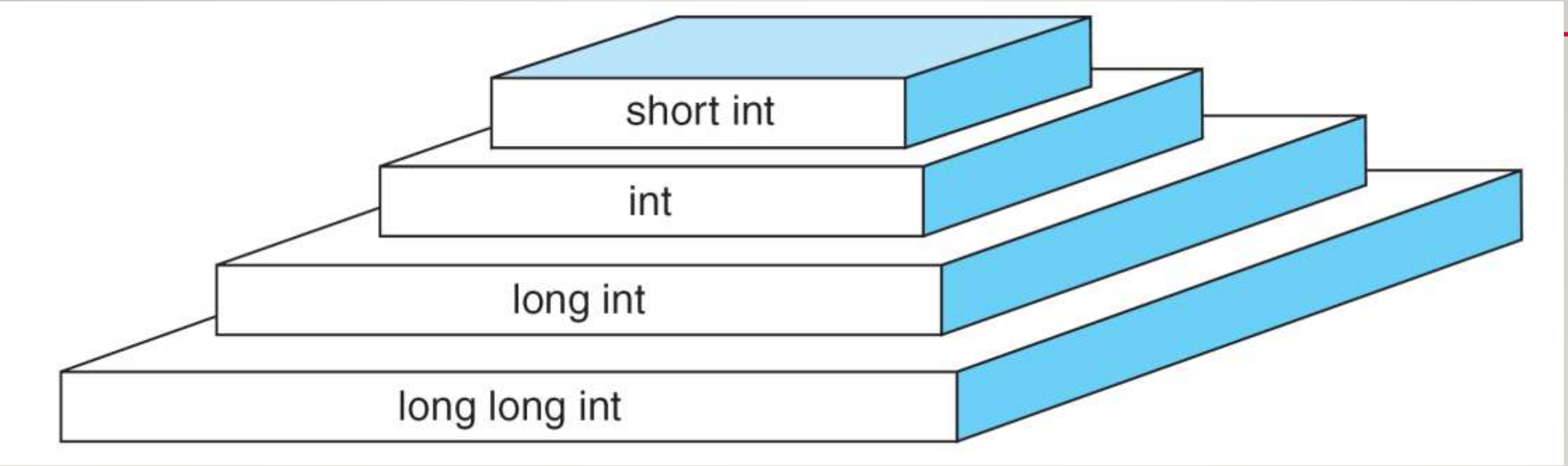
- A character is any value that can be represented in the computer's alphabet
- It is referred by the keyword *char*
- One byte is used to store *char*. With 8 bits, 256 different values can be possible for the *char* type
- Character can be signed or unsigned.

Integer

- An integer type is a number without a fraction part
- C supports four different sizes of the integer type and is denoted by the keyword int
  - » short int
  - » int
  - » long int
  - » long long int

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

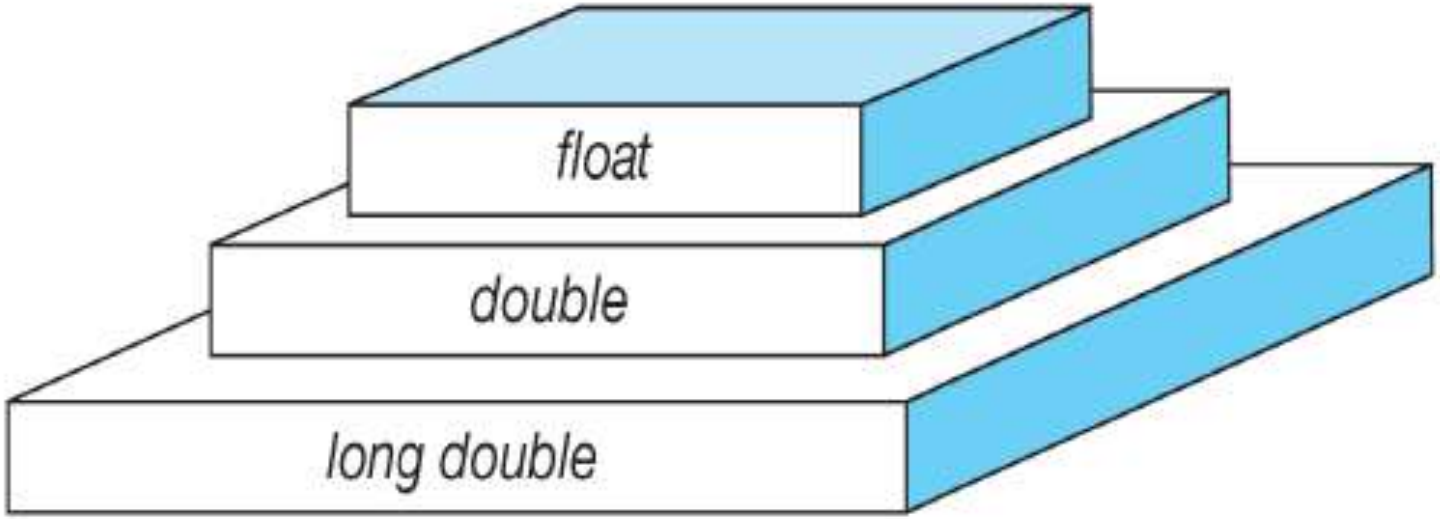
- Each integer size can be signed or unsigned integer. If the integer is signed, one bit is used for signed(0 is plus, 1 is minus). An unsigned integer can store a positive number that is twice as large as the signed integer of the same size.


short int
int
long int
long long int

Floating-point type:

Real

- Real type holds values that consists of integral and fractional part.
- C support types float and double.
- Real type values are always signed.

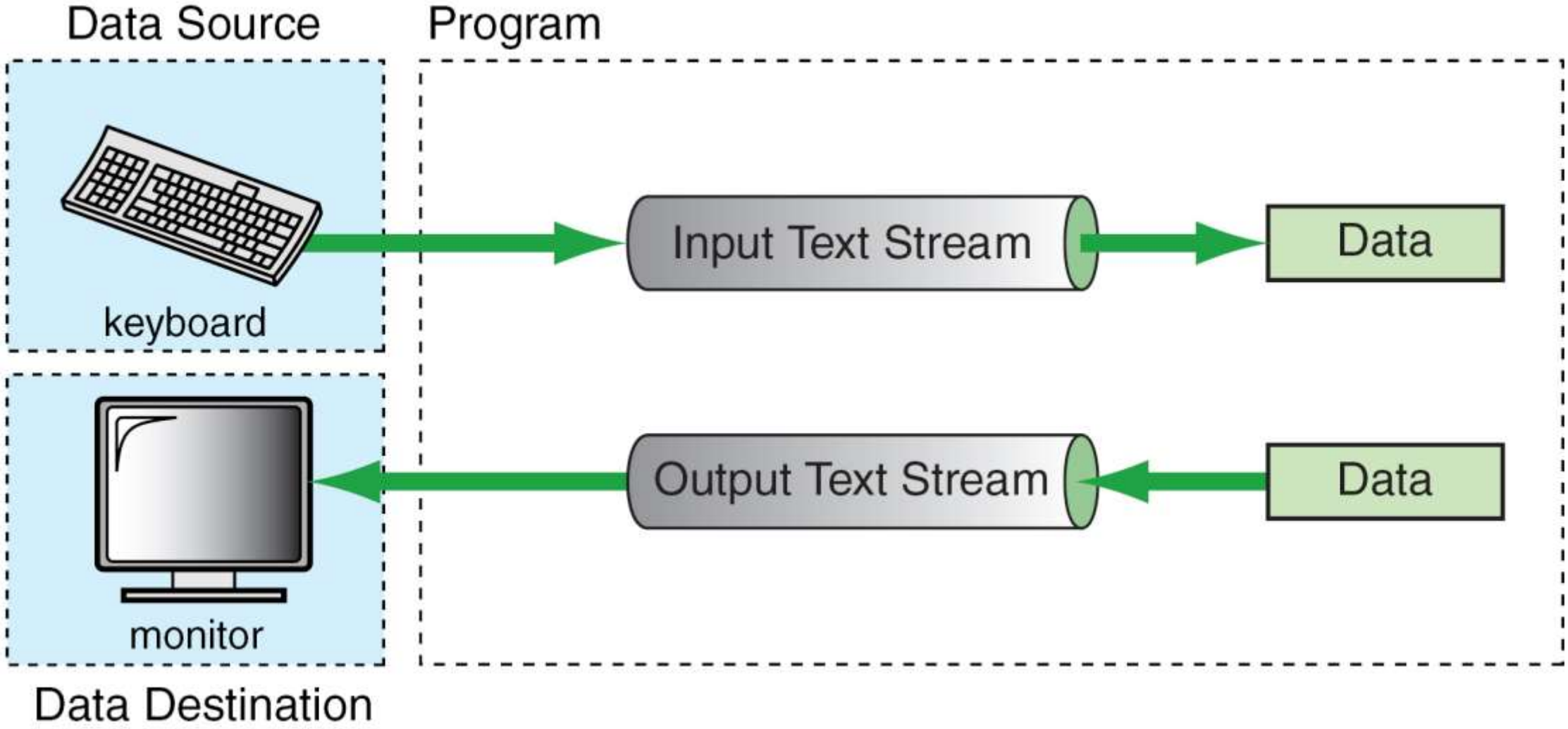


C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

- Data Types

Data Type		Abbreviation	Size (byte)	Range
char	char		1	-128 ~ 127
	unsigned char		1	0 ~ 255
int	int		2 or 4	$-2^{15} \sim 2^{15}-1$ or $-2^{31} \sim 2^{31}-1$
	unsigned int	unsigned	2 or 4	0 ~ 65535 or $0 \sim 2^{32}-1$
	short int	short	2	-32768 ~ 32767
	unsigned short int	unsigned short	2	0 ~ 65535
	long int	long	4	$-2^{31} \sim 2^{31}-1$
	unsigned long int	unsigned long	4	$0 \sim 2^{32}-1$
float			4	
double			8	

BASIC INPUT - OUTPUT



FORMATTED INPUT/OUTPUT

- Input comes from files
- Output sent to files
- Other objects treated like files:
  - keyboard - standard input file (stdin)
  - monitor - standard output file (stdout)
- Generally send/retrieve characters to/from files

FORMATTED OUTPUT

- Command: ***printf*** - print formatted
- Syntax: printf(*Format String, Data List*);
  - Format string any legal string
  - Characters sent (in order) to screen
- Ex.: printf("Welcome to\nCS 1621!\n");

  causes

```
Welcome to
CS 1621!
```

  to appear on monitor

FORMATTED OUTPUT (CONT)

- Successive printf commands cause output to be added to previous output
- Ex.

  printf("Hi, how ");
  printf("is it going\nin 1621?");
  prints
  Hi, how is it going
  in 1621?
  To the monitor

FIELD SPECIFICATIONS

- Format string may contain one or more field specifications
  - Syntax: *%[Flag][Width][Prec][Size]Code*
  - Codes:
    - c - data printed as character
    - d - data printed as integer
    - f - data printed as floating-point value
  - For each field specification, have one data value after format string, separated by commas

Type	Size	Code	Example
char	None	c	%c
short int	h	d	%hd
int	None	d	%d
long int	None	d	%ld
long long int	ll	d	%lld
float	None	f	%f
double	None	f	%f
long double	L	f	%Lf

Data Type		Format
Integer	Integer	%d
	Short	%d
	Short unsigned	%u
	Long	%ld
	Long assigned	%lu
	Hexadecimal	%x
	Long hexadecimal	%lx
	Octal	%O (letter 0)
	long octal	%lo
Real	float,double	%f, %lf, %g
Character		%c
String		%s

FIELD SPECIFICATION EXAMPLE

printf("%c %d %f\n",'A',35,4.5);

produces

```
A 35 4.50000
```

(varies on different computers)

Can have variables in place of literal constants (value of variable printed)

WIDTH AND PRECISION

- When printing numbers, generally use width/precision to determine format
  - Width: how many character spaces to use in printing the field (minimum, if more needed, more used)
  - Precision: for floating point numbers, how many characters appear after the decimal point, width counts decimal point, number of digits after decimal, remainder before decimal

WIDTH/PRECISION EXAMPLE

printf("%5d%8.3f\n",753,4.1678);

produces

```
  753    4.168
```

values are right justified

If not enough characters in width, minimum number used

use 1 width to indicate minimum number of chars should be used

LEFT JUSTIFICATION (FLAGS)

Put - after % to indicate value is left justified

printf("%-5d%-8.3fX\n",753,4.1678);

produces

```
753   4.168    X
```

For integers, put 0 after % to indicate should pad with 0's

printf("%05d",753);

produces

```
00753
```

PRINTF NOTES

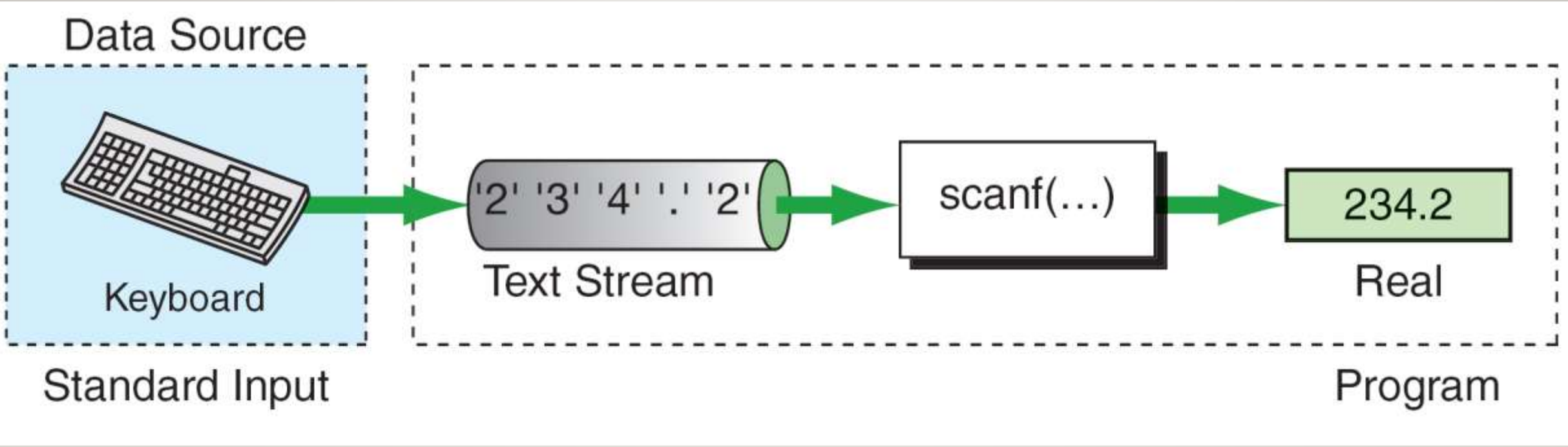
- Important to have one value for each field specification
  - some C versions allow you to give too few values (garbage values are formatted and printed)
- Values converted to proper type
  - printf("%c",97); produces the character a on the screen

Flag Type	Flag Code	Formatting
Justification	None	right justified
	–	left justified
Padding	None	space padding
	0	zero padding
Sign	None	positive value: no sign negative value: –
	+	positive value: + negative value: –
	None	positive value: space negative value: –

FORMATTED INPUT

- Command: ***scanf*** - scan formatted
- Syntax: scanf(*Format String, Address List*);
  - Format string a string with one or more field specifications
  - Characters read from keyboard, stored in variables
- scanf("%c %d %f",&cVar,&dVar,&fVar);

  attempts to read first a single character, then a whole number, then a floating point number from the keyboard


Data Source
Keyboard
Standard Input
'2' '3' '4' '.' '2'
Text Stream
scanf(…)
234.2
Real
Program

FORMATTED INPUT (CONT)

- Generally only have field specifications and spaces in string
  - any other character must be matched exactly (user must type that char or chars)
  - space characters indicate white-space is ignored
  - “white-space” - spaces, tabs, newlines
  - %d and %f generally ignore leading white space anyway (looking for numbers)
  - %d and %f read until next non-number char reached

FORMATTED INPUT (CONT)

- More notes
  - can use width in field specifications to indicate max number of characters to read for number
  - computer will not read input until return typed
  - if not enough input on this line, next line read, (and line after, etc.)
  - inappropriate chars result in run-time errors (x when number expected)
  - if end-of-file occurs while variable being read, an error occurs

ADDRESS OPERATOR

- & - address operator
- Put before a variable (as in &x)
- Tells the computer to store the value read at the location of the variable
- More on address operators later

SCANF RULES

- Conversion process continues until
  - end of file reached
  - maximum number of characters processed
  - non-number char found number processed
  - an error is detected (inappropriate char)
- Field specification for each variable
- Variable address for each field spec.
- Any character other than whitespace must be matched exactly

SCANF EXAMPLE

```
scanf("%d%c %f",&x,&c,&y);
```

and following typed:

```
-543A


4.056 56
```

-543 stored in x, A stored in c, 4.056 stored in y, space and 56 still waiting (for next scanf)

PROMPTING FOR INPUT

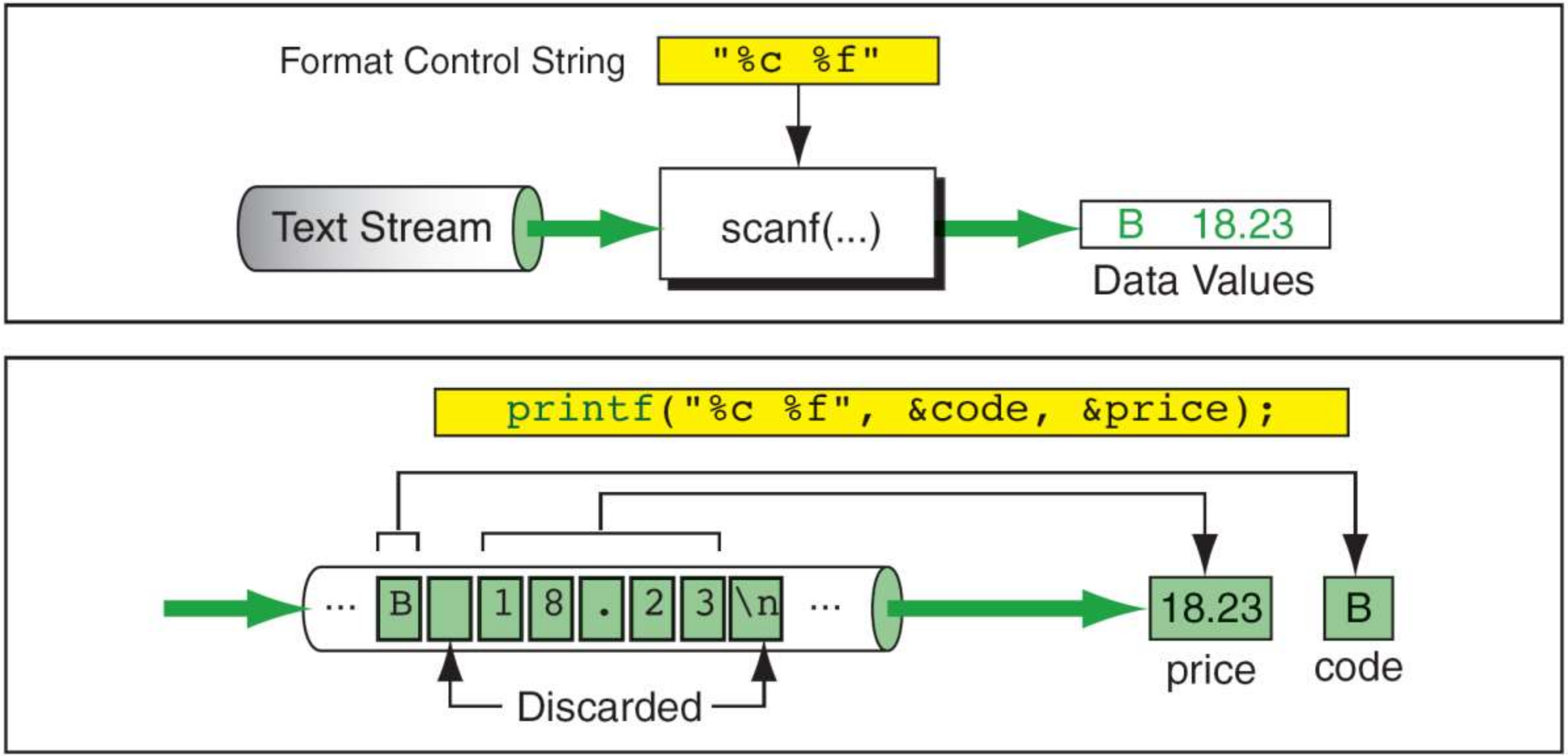
- Using output statements to inform the user what information is needed:

```
printf("Enter an integer:");
scanf("%d",&intToRead);
```

- Output statement provides a cue to the user:

Enter an integer: *user types here*

(a) Basic Concept



(b) Implementation

- There are three differences between the conversion codes for input and output formatting:

1) There is no precision in an input conversion specification.

-It is an error to include a precision

2)There is only one flag for input formatting , the assignment suppression flag(*).

-The assignment suppression flag tells the scanf the next input field is to be read but not store.

Ex: scanf("%d %*c %f",&x, &y);

3) The width modifier specifies the maximum number of characters that are to be read for one format code.

Input Parameters

- Every conversion specification there must be a matching variable in the address list.
- The address of a variable is indicated by prefixing the variable name with an ampersand (&).
- In C language '& ' is known as address operator.
- The first conversion specification matches the first variable address , the second conversion specification matches the second variable address and so on.
- The variable type match the conversion type.
- The c compiler does not verify that they match . If they don't match the input data will not be formatted properly when they are stored in the variables.

End of File and Errors

- The scanf terminate input process when the user signal that there is no more input by keying end of file(EOF).

  Ex: Ctrl+Z

- If scanf encounters an invalid character when it is trying to convert the input to the stored data type ,it stops.

  Ex: Character is trying to read a numeric .

```
1  /* Demonstrate printing Boolean constants.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  #include <stdbool.h>
7
8  int main (void)
9  {
10 // Local Declarations
11    bool x = true;
12    bool y = false;
13
14 // Statements
15    printf ("The Boolean values are: %d %d\n", x, y);
16    return 0;
17 }  // main
```

```
Results:
The Boolean values are: 1 0
```

```c
#include <stdio.h>

int main (void)
{
// Local Declarations
   char A          = 'A';
   char a          = 'a';
   char B          = 'B';
   char b          = 'b';
   char Zed        = 'Z';
   char zed        = 'z';
   char zero       = '0';
   char eight      = '8';

// Statements
   printf("ASCII for char 'A'  is: %d\n",  A);
   printf("ASCII for char 'a'  is: %d\n",  a);
   printf("ASCII for char 'B'  is: %d\n",  B);
   printf("ASCII for char 'b'  is: %d\n",  b);
   printf("ASCII for char 'Z'  is: %d\n",  Zed);
   printf("ASCII for char 'z'  is: %d\n",  zed);
   printf("ASCII for char '0'  is: %d\n",  zero);
   printf("ASCII for char '8'  is: %d\n",  eight);
```

```
Results:
ASCII for character 'A'   is: 65
ASCII for character 'a'   is: 97
ASCII for character 'B'   is: 66
ASCII for character 'b'   is: 98
ASCII for character 'Z'   is: 90
ASCII for character 'z'   is: 122
ASCII for character '0'   is: 48
ASCII for character '8'   is: 56
```

```
/* This program calculates the area and circumference
   of a circle using PI as a defined constant.
      Written by:
      Date:
*/
#include <stdio.h>
#define PI  3.1416

int main (void)
{
// Local Declarations
   float circ;
   float area;
   float radius;
```

```
// Statements
   printf("\nPlease enter the value of the radius: ");
   scanf("%f", &radius);

   circ   = 2  * PI      * radius;
   area   = PI * radius * radius;

   printf("\nRadius is :                %10.2f", radius);
   printf("\nCircumference is : %10.2f", circ);
   printf("\nArea is :                  %10.2f", area);

   return 0;
} // main
```

```
Results:
Please enter the value of the radius: 23

Radius is :                    23.00
Circumference is :            144.51
Area is :                    1661.91
```

INPUT/OUTPUT IN C

getchar ( ) ;

- This function provides for getting exactly one character from the keyboard.
- Example:

```
char ch;
ch = getchar ( ) ;
```

INPUT/OUTPUT IN C

putchar (char) ;

- This function provides for printing exactly one character to the screen.
- Example:

```
char ch;
ch = getchar ( ) ;  /* input a character from kbd*/
putchar (ch) ;      /* display it on the screen */
```