Problem Solving using C

Text Books:

1. Byron Gottfried, Schaum's Outline of Programming with C, McGraw-Hill.
2. E. Balaguruswamy, Programming in ANSI C, Tata McGraw-Hill.

Reference Books:

3. Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall of India.

UNIT-1

Topics to be covered in UNIT-1

Problem solving Techniques

- Algorithms
- Pseudo code
- flowcharts with examples

Introduction to Computer Programming Languages

- Machine Languages
- Symbolic Languages
- High-Level Languages

Introduction to C language

- Characteristics of C language
- Structure of a C Program
- Syntax and semantics
- Data Types
- Variables – declarations and initialization
- Formatting input and output

Algorithm

Algorithm: It is an **ordered sequence** of **unambiguous** and **well-defined instructions** that **performs some task** and **halts in finite time**

Let's examine the four parts of this definition more closely.

1. **Ordered Sequence:** You can number the step.
2. **Unambiguous** and well defined instructions: Each instruction should be clear, understand without difficulty.
3. **Performs** some task
4. Halts in **finite time**: Algorithm must terminate at some point.

Properties of an Algorithm:-

- **Finiteness**: An algorithm must terminate in a finite number of steps.
- **Definiteness**: Each step of an algorithm must be precisely and unambiguously stated.
- **Effectiveness**: Each step must be effective, and can be performed exactly in a finite amount of time.
- **Generality**: The algorithm must be complete in itself.
- **Input/Output**: Each algorithm must take zero, one or more inputs and produces one or more output.

Three Categories of Algorithmic Operations

An algorithm must have the ability to alter the order of its instructions. An instruction that alters the order of an algorithm is called a control structure.

Three categories of an algorithmic operations:

1. **Sequential operations:** Instructions are executed in order

2. **Conditional/Selection ("question asking") Operations:** A control structure that asks a true/false question and then selects the next instruction based on the answer.

3. **Iterative Operations (loops):** A control structure that repeats the execution of a block of instructions.

Pseudo-code and Algorithm Construction

Assignment:

```
        variable  =  "expression"
```

Input/Output:

```
        Get/enter/read  "variable", "variable", ...
        Display/print "variable", "variable", ...
```

Conditional:

```
    1.  if  "condition"
        1.1    (subordinate) statement 1
        1.2     etc ...
    2.  else
        2.1    (subordinate) statement 2
        2.2    etc ...
```

Iterative:

```
    3.  while "condition"
        3.1    (subordinate) statement 1
        3.2    etc ...
```

Pseudo Code

Definition: pseudocode is an informal description of a sequence of steps for solving problems

English-like statements that follow a loosely defined syntax and are used to convey the design of an algorithm.

Pseudocode is sometimes used as a detailed step in the process of developing a program.

Example1: To determine whether a student is passed or not

Pseudo Code:

```
1. If student's grade is greater than or equal to 60
    1.1 Print "passed"
2. else
    2.1 Print "failed"
```

Algorithm:

```
Begin
    1. If grade >= 60
        1.1 Print "passed"
    2. else
        2.1 Print "failed”
End
```

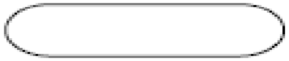
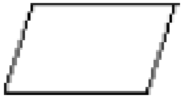
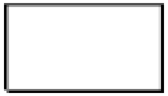
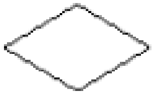
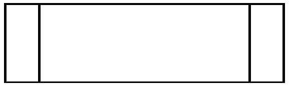
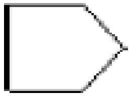
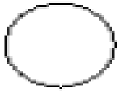
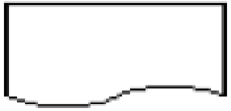
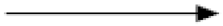
Flowchart

- Pictorial representation of an algorithm is called flowchart.

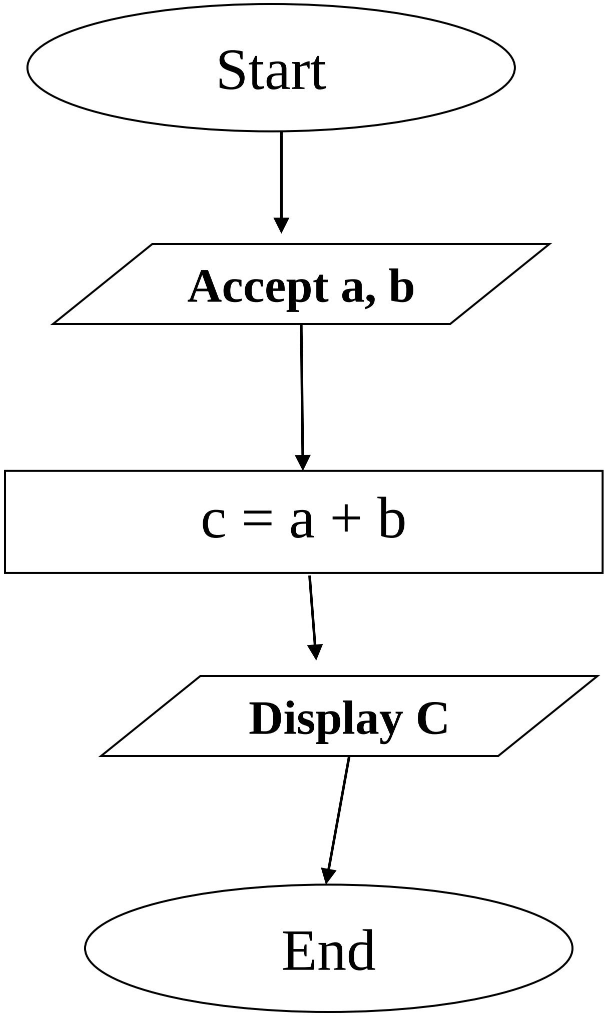
or

- A diagram that uses graphic symbols to depict the nature and flow of the steps in a process.

SYMBOLS USED WITH FLOWCHARTS

	Represents Start, End
	Represents Input, Output data
	Represents Process (actions, calculations)
	Represents Decision Making
	Represents Pre-defined Process / module
	Represents off page connector which are used to indicate that the flow chart continues on another page. Page numbers are usually placed inside for easy reference.
	**Connector Symbol** represents the exit to, or entry from, another part of the same flow chart. It is usually used to break a flow line that will be continued elsewhere.
	The **Document Symbol** is used to represent any type of hard copy input or output (i.e. reports).
	Represents control flow

Flowchart to find the addition of two numbers

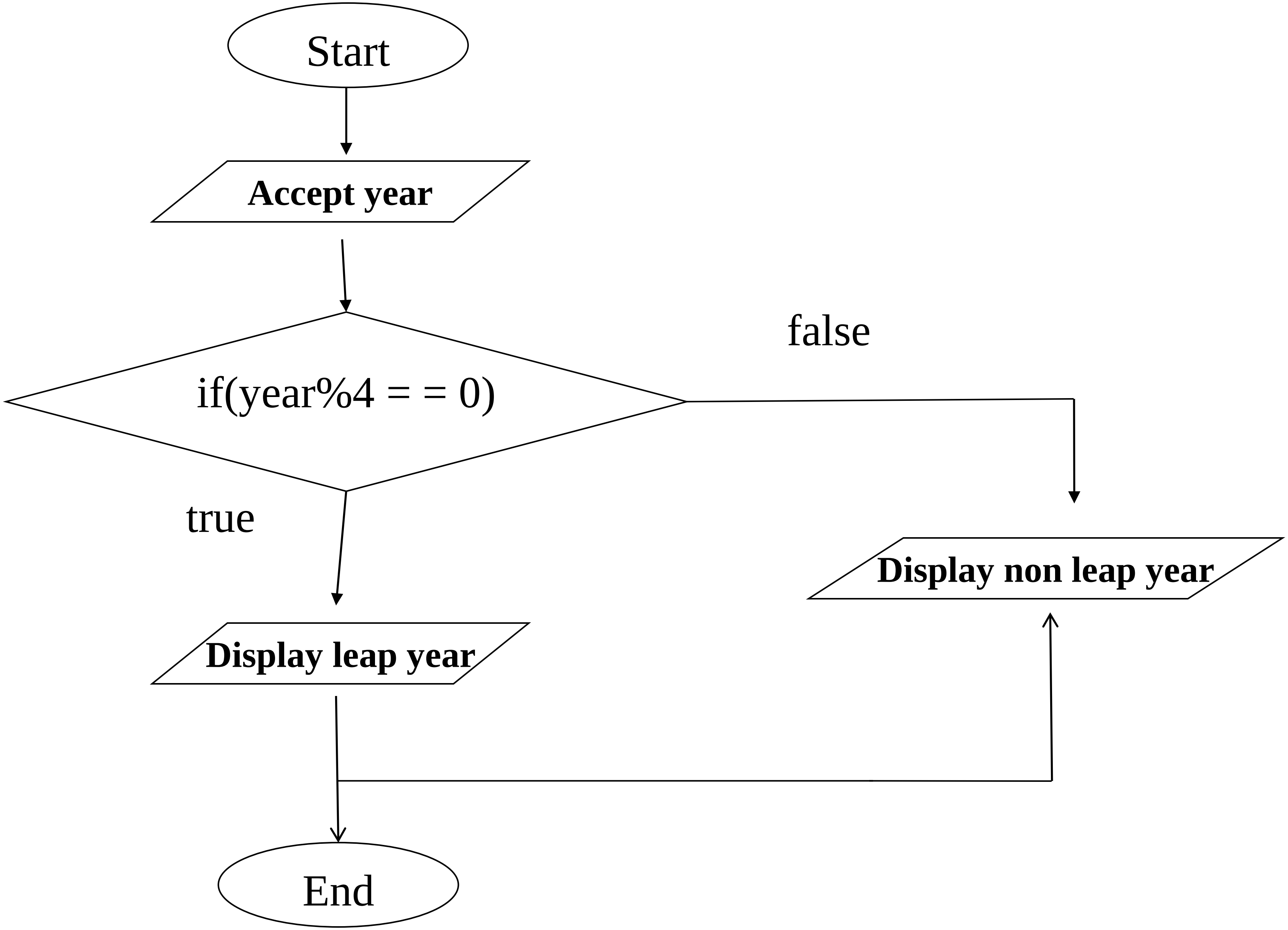


Algorithm to find whether a given year is a leap year or not.

Algorithm:

```
BEGIN
      Step 1:  Accept the YEAR
      Step 2:  if (YEAR %4 == 0) then
                     Display “Year is a leap year”
               else
                     Display “Year is not a leap year”
               end if
 END
```

Flowchart to find whether a given year is a leap year or not



Computer Software

- **Software** refers to a program or set of instructions that is written to achieve a specified task.

- These instructions need to be written in a **programming language** that the computer can understand.

- Programming Language: An artificial set of rules, vocabulary and syntax used to instruct the computer to execute certain tasks.

Software is mainly categorized into two groups.

- **System Software**
- **Application Software**

Operating System:

An Operating System (OS) is an interface between a computer user and computer hardware.

An operating system is a software which performs the following tasks:

- File Management
- Memory Management
- Process Management
- Device Management
- Controlling peripheral devices such as disk drives and printers.

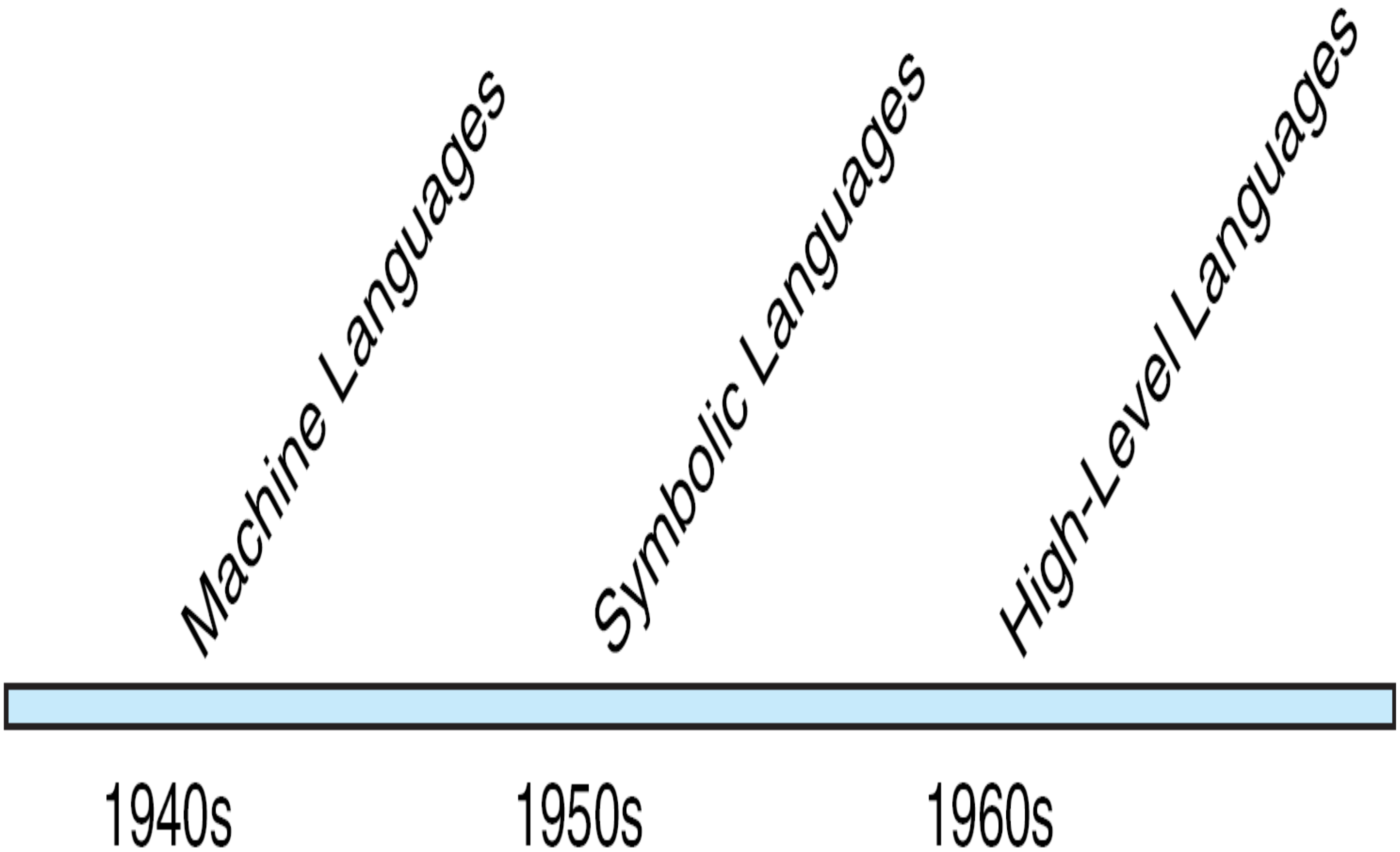
Computer Languages

- A program's instructions need to be written in a **programming language** that the computer can understand.

- The first programming language is **machine language**.

- Computer languages were evolved from machine language to natural language (like English language).

- Computer Languages are basically divided into three categories:
  - Machine language
  - Symbolic language
  - High level languages



Computer Language Evolution

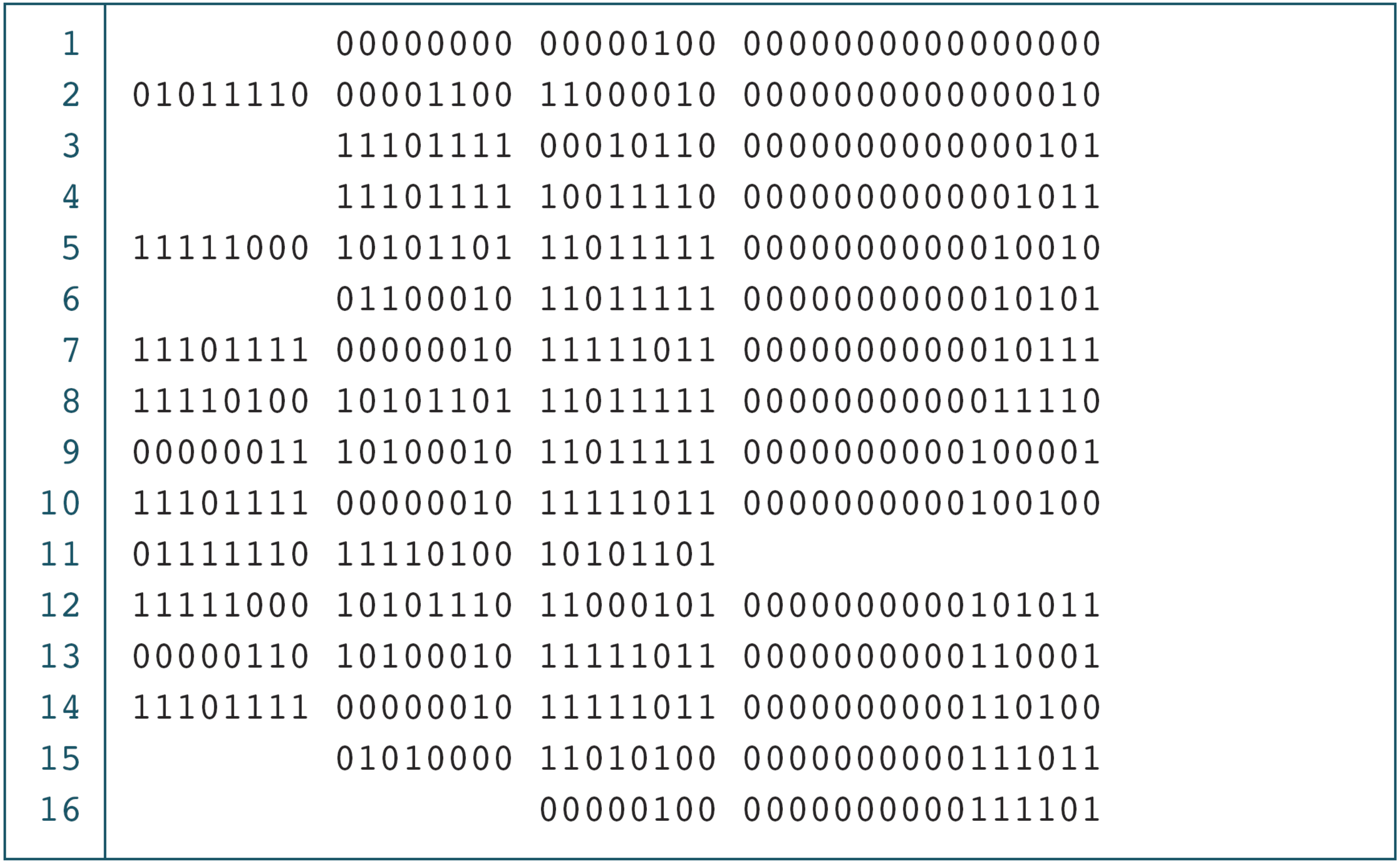
Machine Language

- Machine language was the first programming language in the early days of computers.

- The language which is understand by the computer hardware is called **machine language**.

- It consists of **0's** and **1's**.

The Multiplication Program in Machine Language



```
         00000000 00000100 0000000000000000
01011110 00001100 11000010 0000000000000010
         11101111 00010110 0000000000000101
         11101111 10011110 0000000000001011
11111000 10101101 11011111 0000000000010010
         01100010 11011111 0000000000010101
11101111 00000010 11111011 0000000000010111
11110100 10101101 11011111 0000000000011110
00000011 10100010 11011111 0000000000100001
11101111 00000010 11111011 0000000000100100
01111110 11110100 10101101
11111000 10101110 11000101 0000000000101011
00000110 10100010 11111011 0000000000110001
11101111 00000010 11111011 0000000000110100
         01010000 11010100 0000000000111011
                  00000100 0000000000111101
```

Example for Machine Language

Symbolic Language

- Writing program in machine language is difficult.

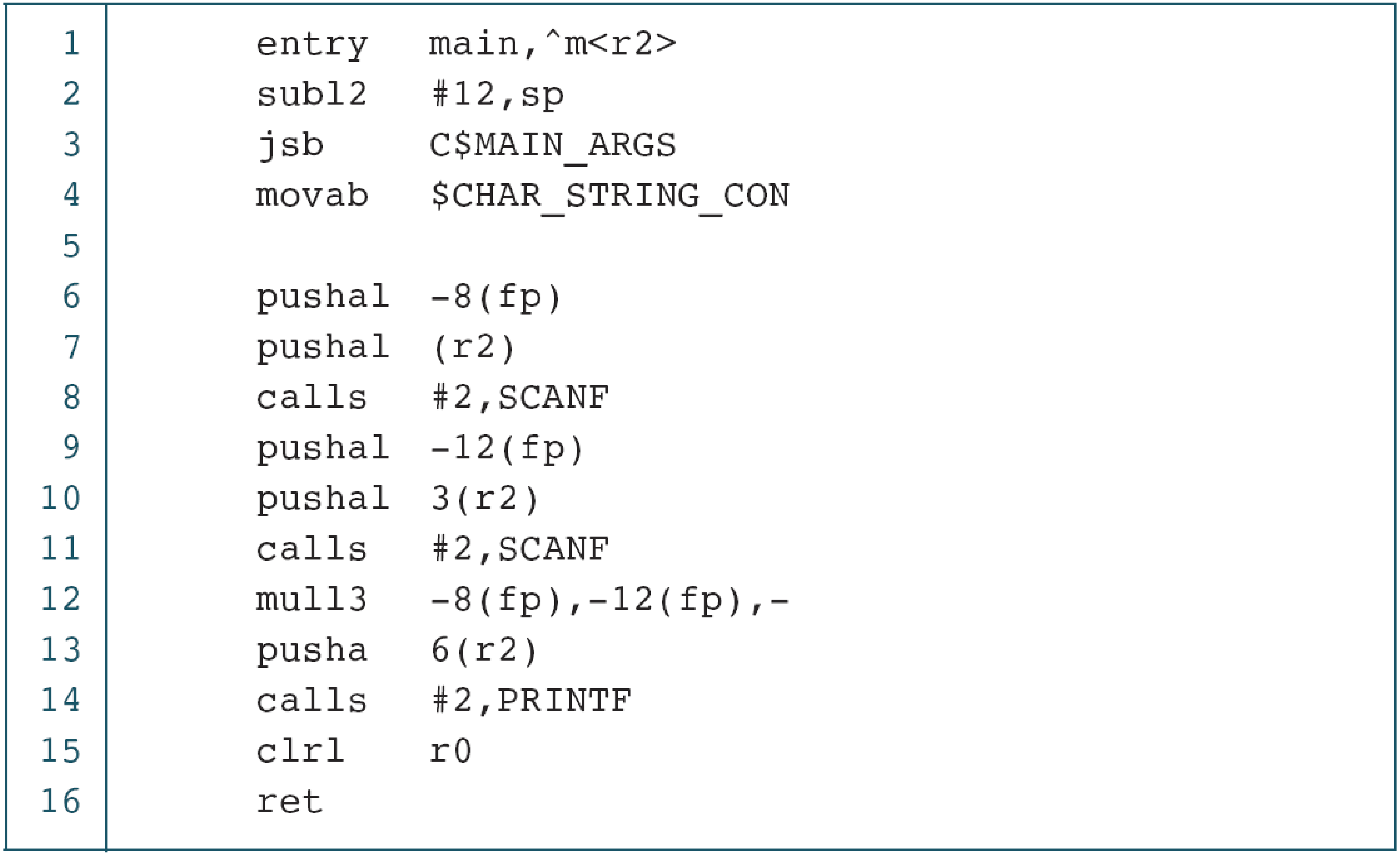
- The language which is represented using symbols or mnemonics is called as **symbolic language**.

- This language is not understandable by the computer .

- Hence, it must be translated to the machine language using the **assembler**.

- Another name of the symbolic language is **assembly language**.

The Multiplication Program in Symbolic Language



```
entry   main,^m<r2>
subl2   #12,sp
jsb     C$MAIN_ARGS
movab   $CHAR_STRING_CON

pushal  -8(fp)
pushal  (r2)
calls   #2,SCANF
pushal  -12(fp)
pushal  3(r2)
calls   #2,SCANF
mull3   -8(fp),-12(fp),-
pusha   6(r2)
calls   #2,PRINTF
clrl    r0
ret
```

Example for Symbolic Language

High Level Language

- It is like natural language which can understandable by the **programmer**.

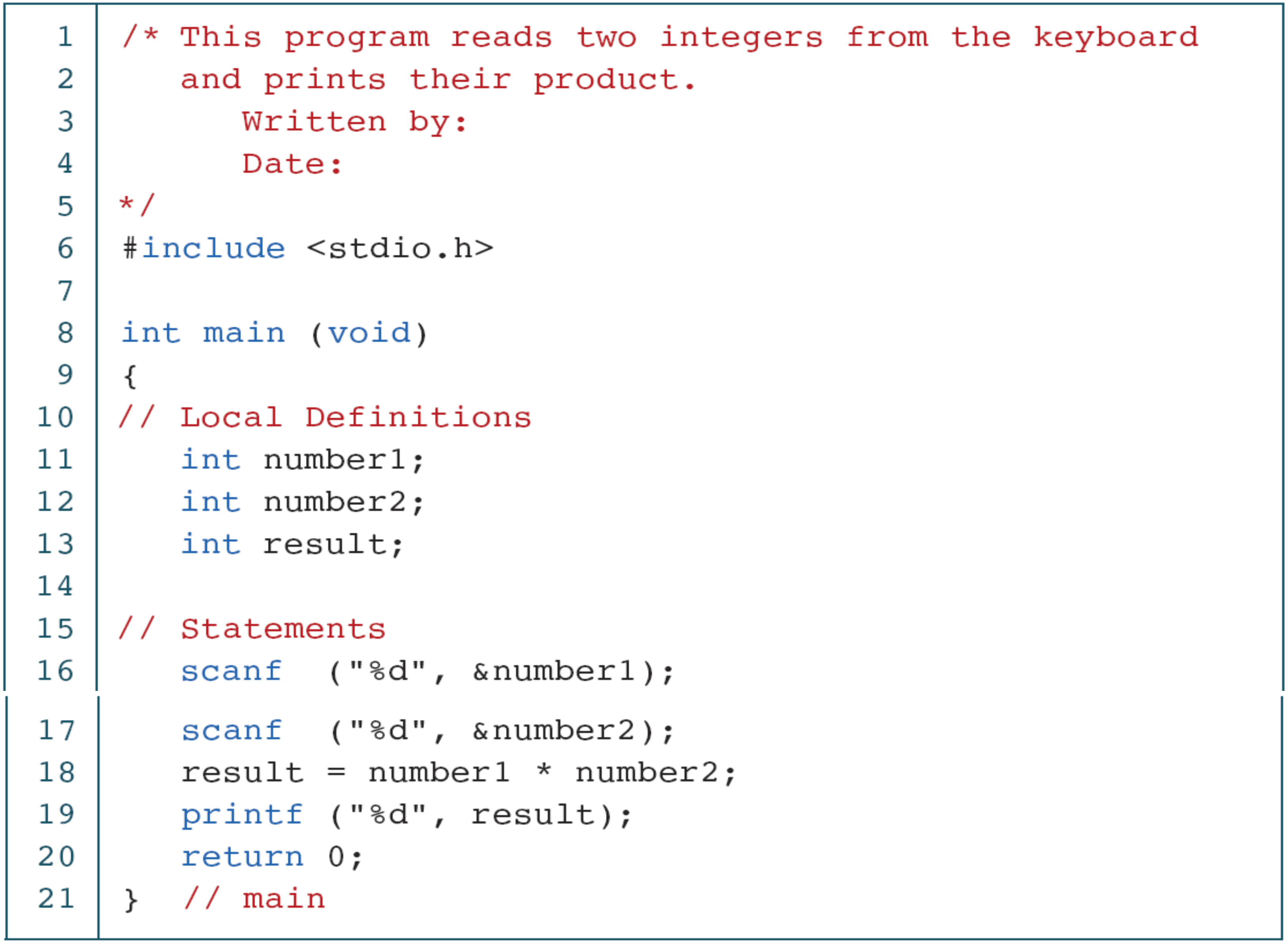
- The High Level Language instructions are not understandable by the **machine**.

- **Compiler** is used to convert High Level Language instructions into the Machine Language instructions.

- First High Level Language is **FORTRAN**.

- Examples for High Level Languages are : C, C++,JAVA, COBOL etc.,

The Multiplication Program in C

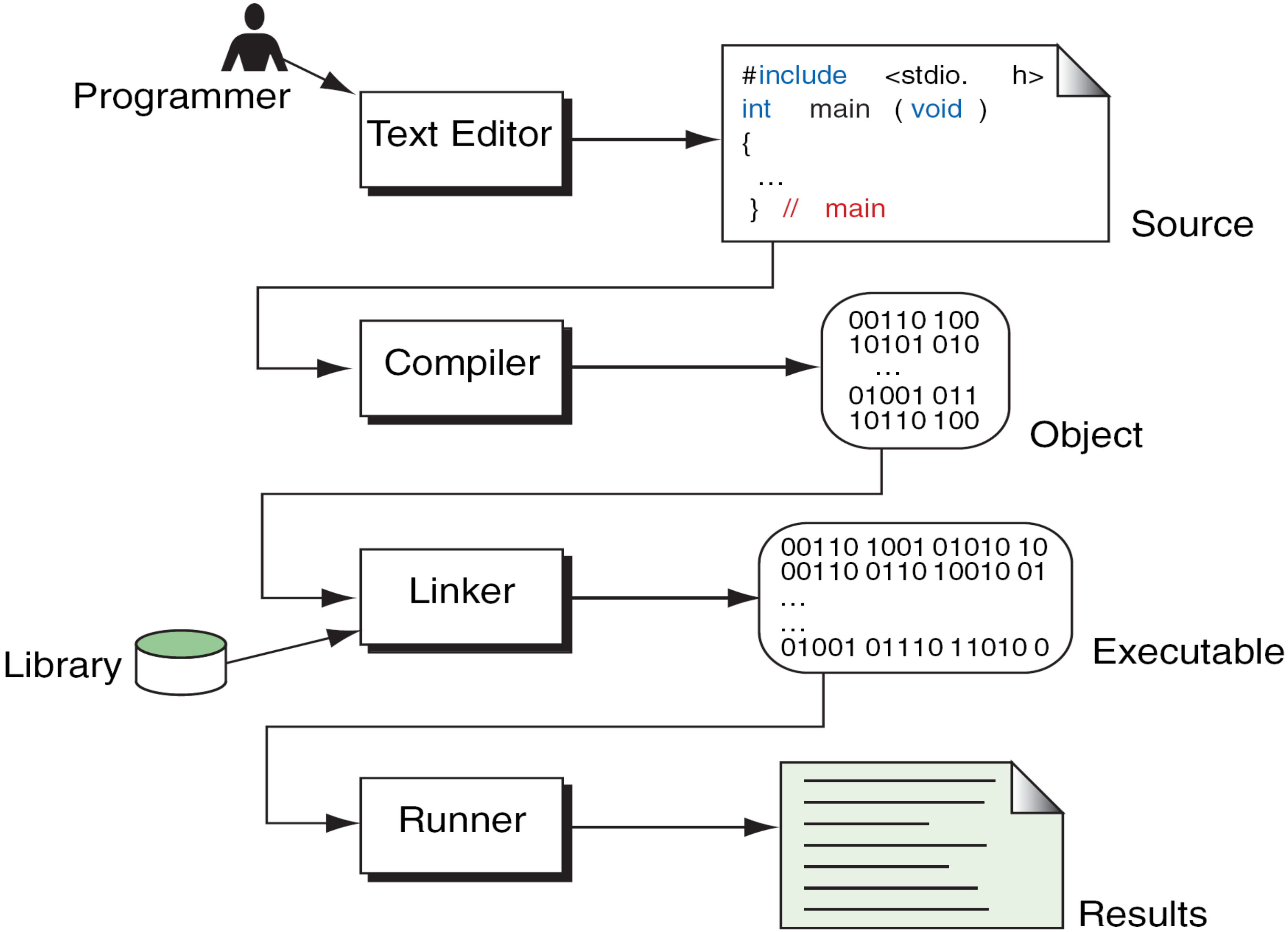


```c
/* This program reads two integers from the keyboard
   and prints their product.
      Written by:
      Date:
*/
#include <stdio.h>

int main (void)
{
// Local Definitions
   int number1;
   int number2;
   int result;

// Statements
   scanf  ("%d", &number1);
   scanf  ("%d", &number2);
   result = number1 * number2;
   printf ("%d", result);
   return 0;
}  // main
```

Creating and Running Programs

Creating and running programs takes place in 4 steps.

1. Writing and Editing the program.
2. Compiling.
3. Linking the program with the required library functions.
4. Executing the program.



```
#include    <stdio.    h>
int    main   ( void  )
{
  …
 }  //   main
```

```
00110 100
10101 010
   …
01001 011
10110 100
```

```
00110 1001 01010 10
00110 0110 10010 01
…
…
01001 01110 11010 0
```

Building a C Program

1. Writing and Editing the program

- Software used to write programs is known as a **text editor**, where you can type, edit and store the data.

- You can write a **C** program in text editor and save that file on to the disk with **“.c”** extension. This file is called **source file**.

2. Compiling Program

- **Compiler** is used to convert High Level Language instructions into the Machine Language instructions.
- It could complete its task in two steps.
  - i) Preprocessor
  - ii) Translator

Preprocessor:

- It reads the source file and checks for special commands known as preprocessor commands (instructions which starts with # symbols).
- The result of preprocessor is called as **translation unit**.
- Preprocessor processes the source file before compilation only.

Translator:

- It is a program which reads the **translation unit** and converts the program into machine language and gives the **object module**.
- This module is not yet ready to run because it does not have the required C and other functions included.

3. Linking a program with required library functions

- C program is made up of different functions in which some functions can be written by the programmer, other functions like **input/output** functions and **mathematical** library functions, that exist **elsewhere** and must be attached to our program.

- The **linker** assembles all of these functions and produces the **executable** file which is ready to run on the computer.

4. Executing the program.

- Once a program has been linked, it is ready for execution.
- Now, you can execute the program by using the run command.
- **Loader** is a program which is used to load the program from the disk to main memory.

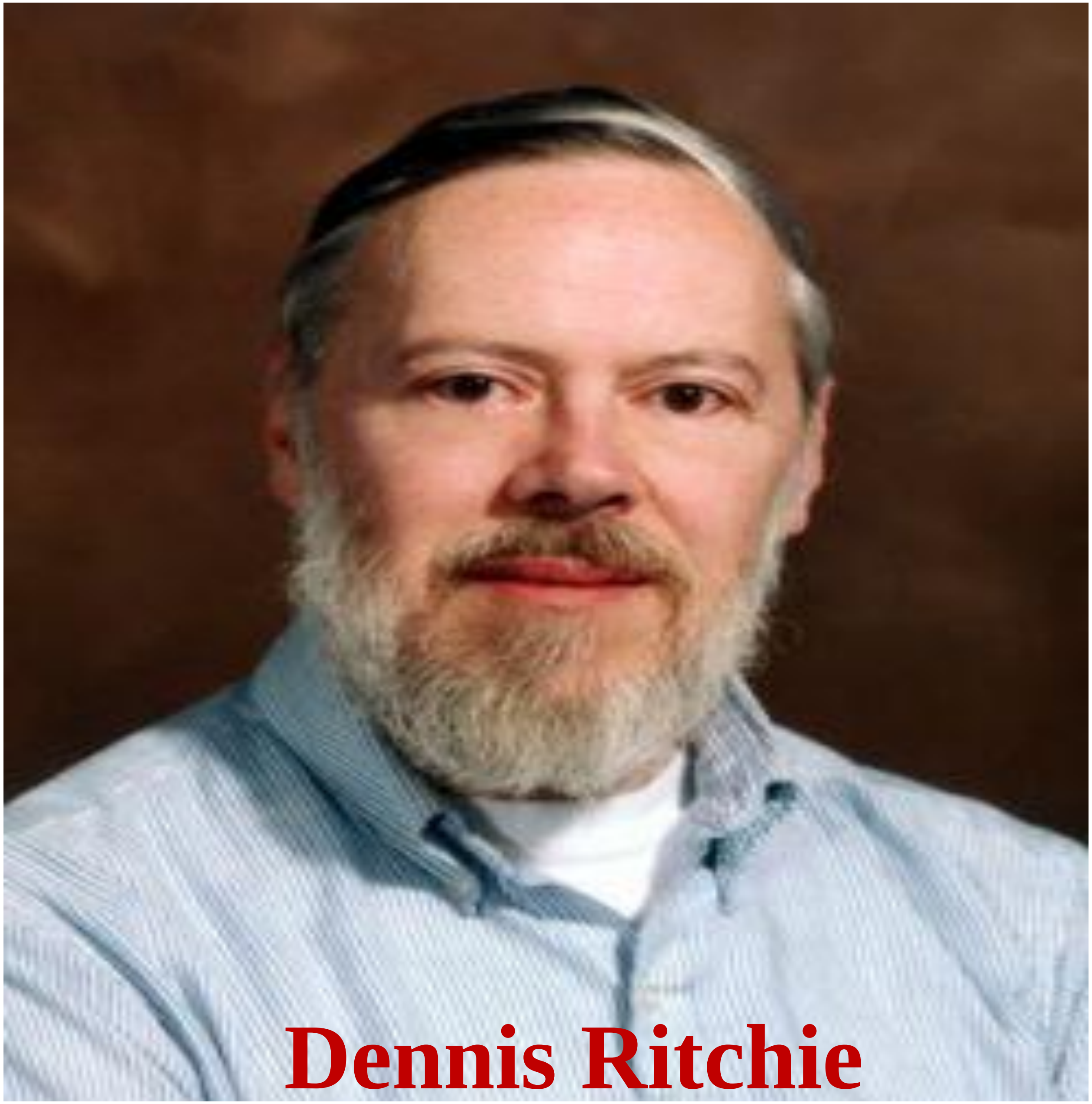
Writing and Running Programs

1. Write a program (source code) using vi editor and save it with **.c** extension. Ex: **$vi sample.c**

2. Run the compiler to convert a program into to “binary” code. Ex: **$cc sample.c**

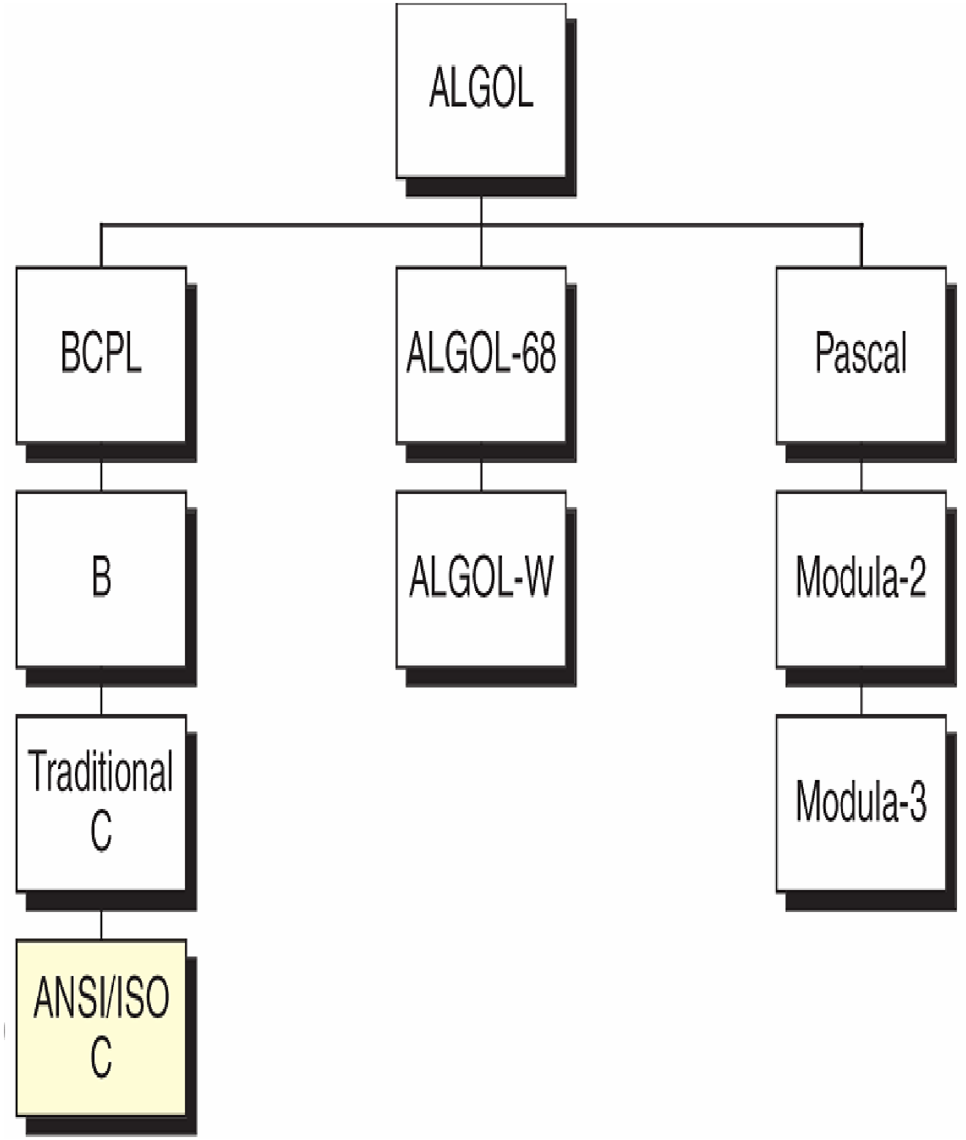
3. Compiler gives errors and warnings if any, then edit the source file, fix it, and re-compile.

4. Run it and see the output. Ex: **$ ./a.out**



Dennis Ritchie

History of C language



Taxonomy of the C Language

History of C language (contd…)

- ALGOL was the first computer language.
- In 1967, **Martin Richards** developed a language called BCPL (Basic Combined Programming Language) at University of Cambridge primarily, for writing system software.
- In 1969, language B was developed by Ken Thompson.
- ‘B’ was used to create early versions of UNIX operating system at Bell Laboratories.
- In 1972, C was developed by Dennis M. Ritchie at Bell Labs(AT&T) in USA.
- In 1983 C language was standardized by ANSI as ANSI C (ANSI-American National Standards Institute).
- UNIX operating system was coded almost entirely in C.

Features of C language

The increasing popularity of C is due to its various features:

- **Robust:** **C** is a robust language with rich set of built-in functions and operators to write any complex programs.

- **C compilers combines the capabilities of low level languages with features of high level language**. Therefore it is suitable for writing the system software, application software and most of the compilers of other languages also developed with **C** language.

- **Efficient and Fast:** Programs written in **C** are efficient and fast. This is due to its variety of data types.

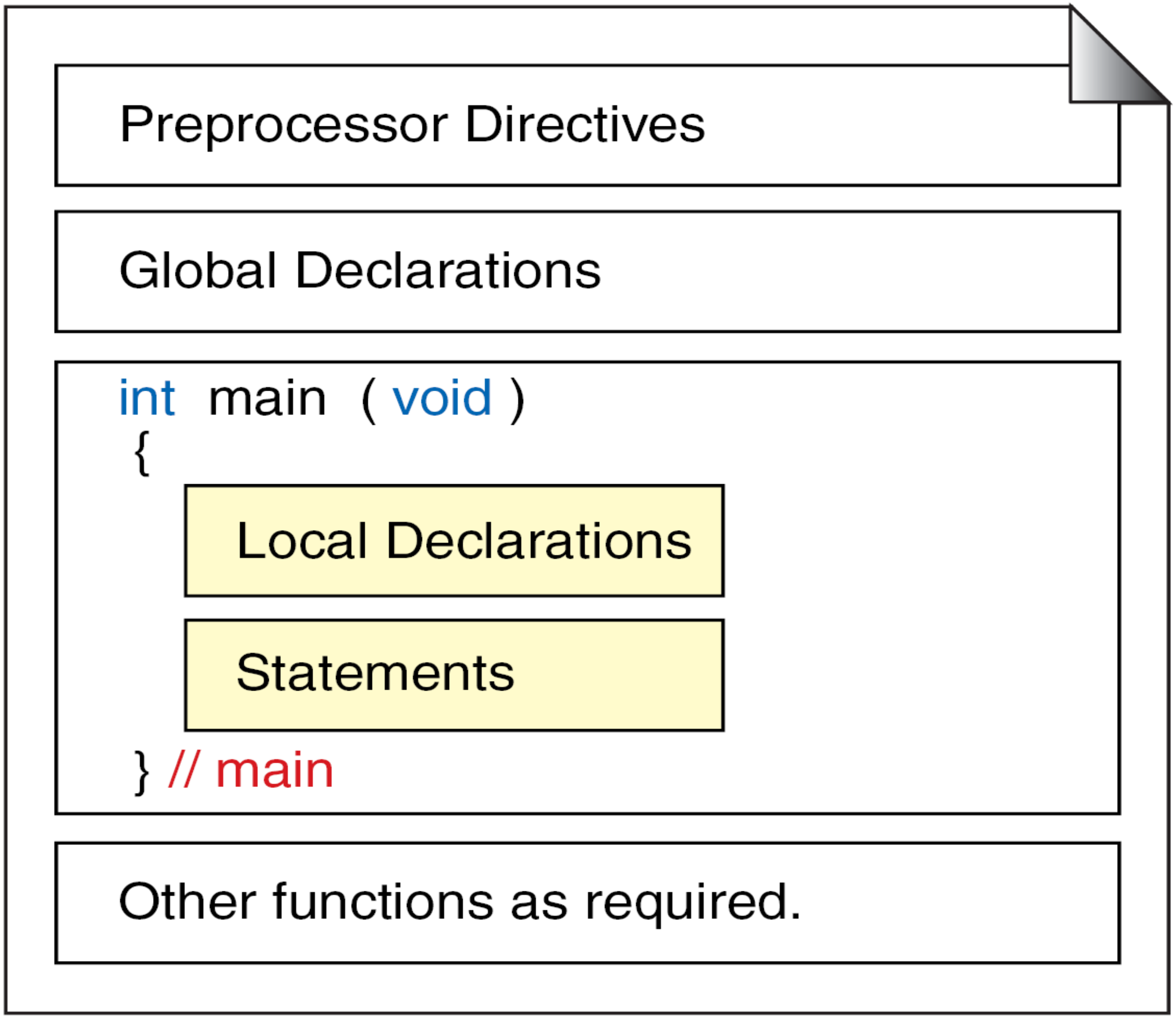
- **Portable:** **C** program written on one computer can also run on the other computer with small or no modification.

Features of C language (contd…)

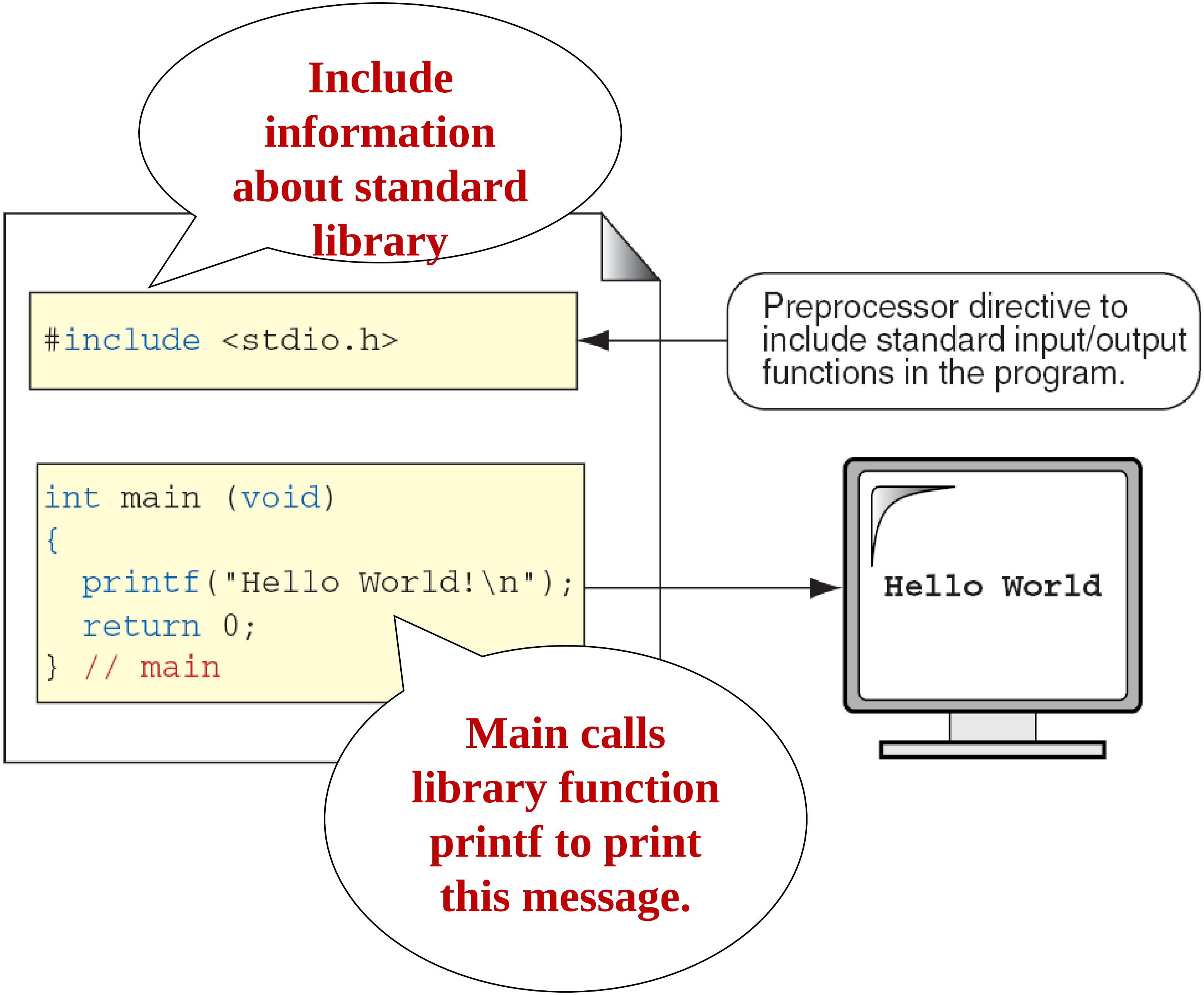
- **Structured Programming:** Every program in C language is divided into small modules or functions so that it makes the program much simple, debugging, and also maintenance of the program is easy.

- **Ability to extend itself:** A C program is basically a collection of various functions supported by C library (also known as header files). We can also add our **own functions** to the **C library**. These functions can be reused in other applications or programs.

Structure of C Program



General structure of a C program



Include information about standard library

```
#include <stdio.h>
```

Preprocessor directive to include standard input/output functions in the program.

```
int main (void)
{
  printf("Hello World!\n");
  return 0;
} // main
```

```
Hello World
```

Main calls library function printf to print this message.

Preprocessor Directives

- The **preprocessor directives** provide instructions to the preprocessor, to include functions from the system library, to define the symbolic constants and macro.

- The preprocessor command always starts with symbol **#.**

- Example: #include<stdio.h>

- **Header file** contains a collection of library files.

- #include<stdio.h> includes the information about the standard input/output library.

- The variables that are used in common by more than one function are called **Global Variables** and are declared in global declaration section.

- Every C program must have one **main()** function. All the statements of main are enclosed in braces.

- The program execution begins at main() function and ends at closing brace of the main function.

- C program can have any number of **user-defined functions** and they are generally placed immediately after the main () function, although they may appear in any order.

- All sections except the main () function may be absent when they are not required.

- In the previous program, main() function **returns** an integer value to the **operating system**.

- Each statement in C program must end with **;** specifies that the instruction is ended.

- A function can be called by **it's name**, followed by a parenthesized list of arguments and ended with semicolon.

- In previous program main() function calls printf() function.
  Example: **printf**("Hello World!\n")**;**

Comments

- To make the program more readable use the comments.

- They may used to make the program easier to understand.

- Two types of comments
  1. Block comment
  2. Line comment.

Block Comment

```
/* Write a program to add two integer numbers */
#include<stdio.h>
main()
{
    int a=10,b=20,c;
    c=a+b;
    printf("Sum of a and b=%d",c);
    return 0;
}
```

- Any characters between **/*** and ***/** are ignored by the compiler.
- Comments may appear anywhere in a program.
- /* and */ is used to comment the multiple lines of code which is ignored by the compiler.
- Nested block comments are invalid like **/*   /*   */**

Line Comment

- To comment a single line use two slashes //

```
/*  Write a program to add two integer numbers */
#include<stdio.h>        // It includes input, output header file
main()
{
    int a,b,c;      // Variables declaration
    a=10;
   B=20;                              //initialization
    c=a+b;                      // Adding two numbers
    printf("Sum of a and b=%d",c);
    return 0;
}
```

Errors in compilation, object and executable code

There are 3 types of errors

1. Compiler Errors
2. Linker Errors
3. Runtime Errors

Compiler Errors: An error that occur during compilation stage is Called a compiler error. These are given by compiler

Linker Errors: An error that occur during linking stage is called a linker error. These are given by linker.

Runtime Errors: An error that occur during the execution of a program is called a runtime error. These are given by operating system.

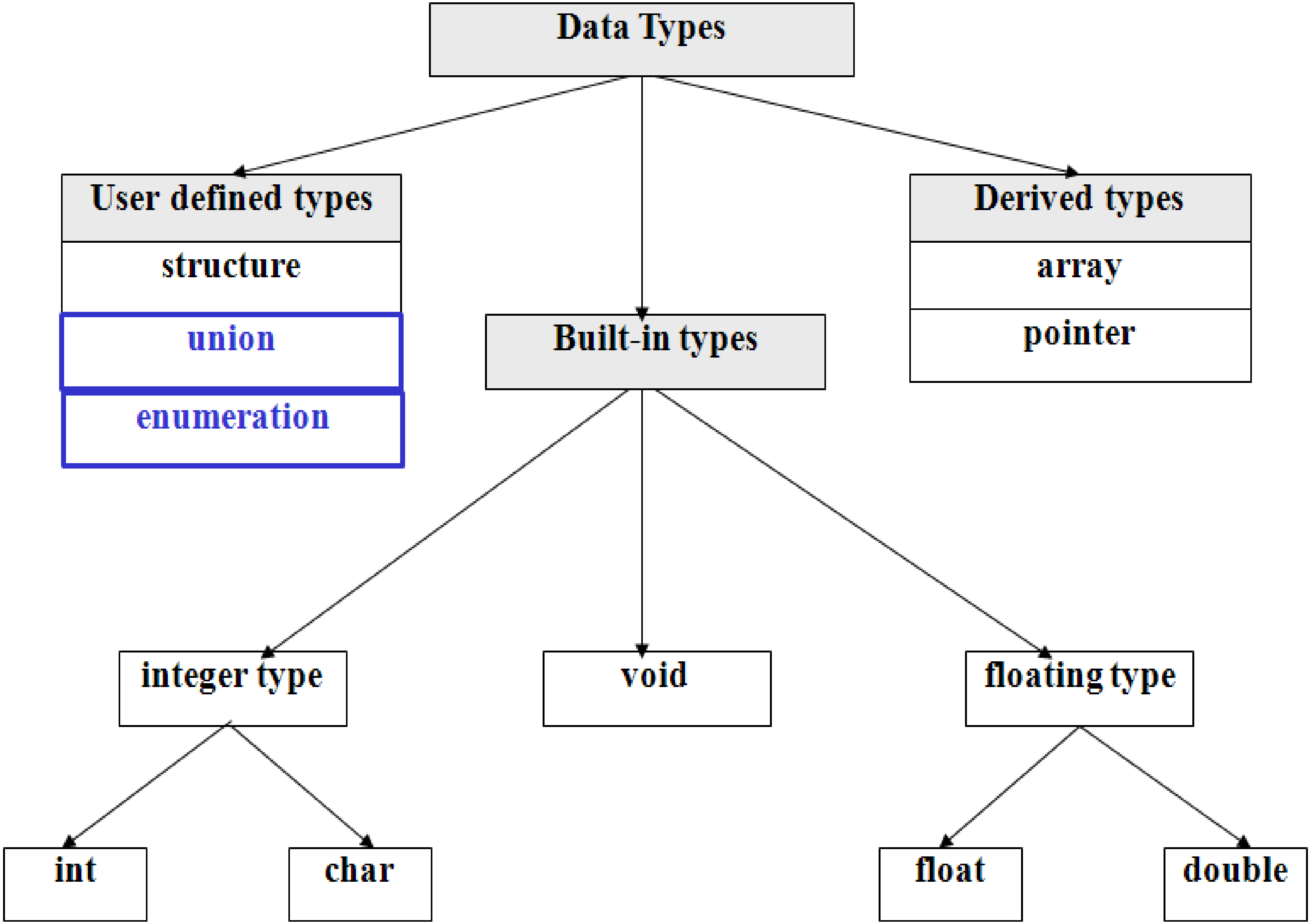
Logical Errors are errors in a program that executes without Performing the intended action. In this case, the program compiles and executes without complaints, but it produce incorrect results. It occurs when the logic of the program as written is different from what was actually intended.

Data types in C

- Data types are used to indicate the **type** of value represented or stored in a variable, the **number of bytes** to be reserved in memory, the **range of values** that can be represented in memory, and the **type of operation** that can be performed on a particular data value.

- ANSI C supports 3 categories of data types:
  - Built-in data types
  - Derived data types
  - User Defined data types

Data types in C (contd…)



Built-in or Primary or Primitive data types:

Built-in data types are also known as primitive data types. C uses the following primitive data types.

int	integer quantity
char	character (stores a single character)
float	floating point number
double	floating point number

Built-in data types (contd…)

Integer data type :

- An **integer number** (also called whole number) has no fractional part or decimal point.

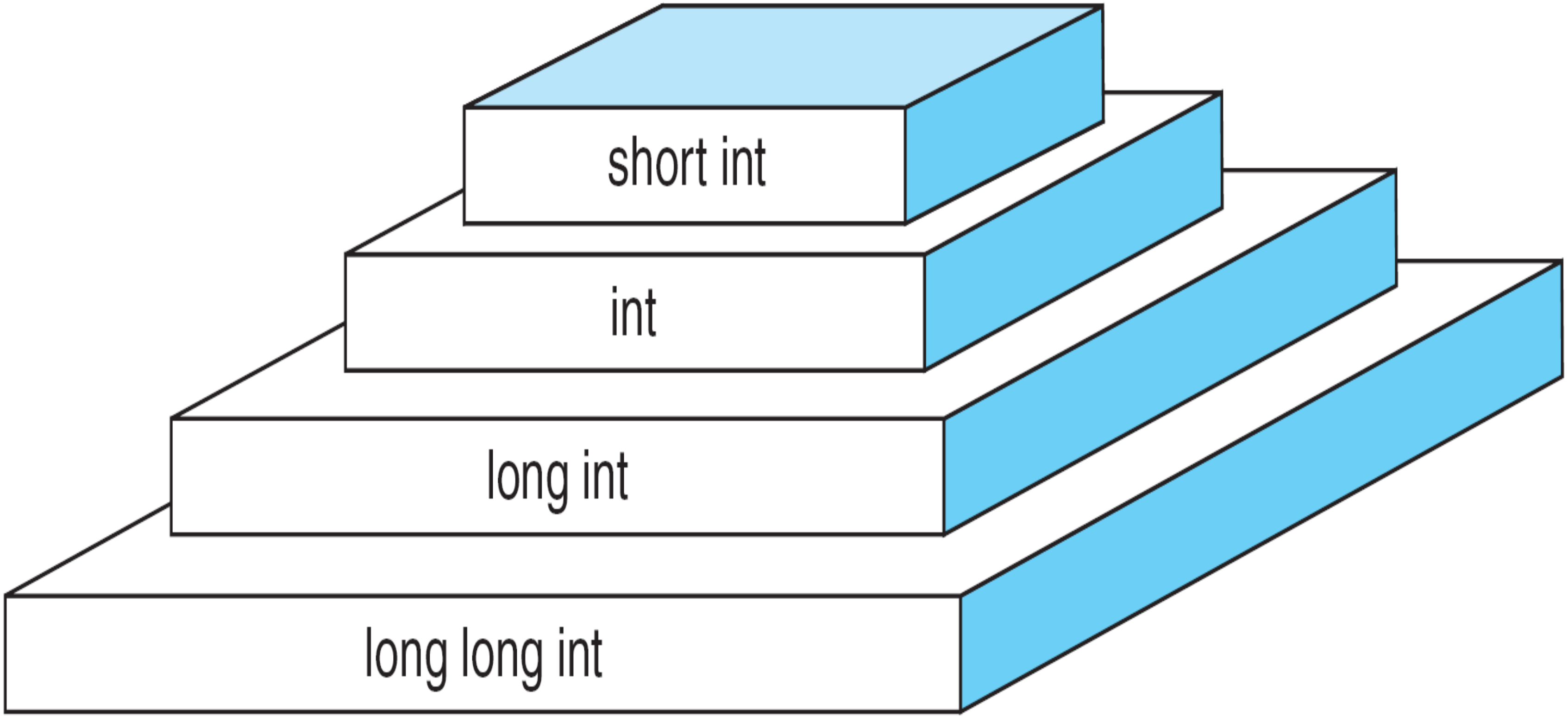
- The keyword **int** is used to specify an integer variable.

- It occupies 2 bytes (16 bits) or 4 bytes (32 bits), depending on the machine architecture.

- 16-bit integer can have values in the range of **-32768** to **32767**
- One bit is used for sign.

void data type:

- Defines an empty data type which can then be associated with some data types. It is useful with pointers.



Note

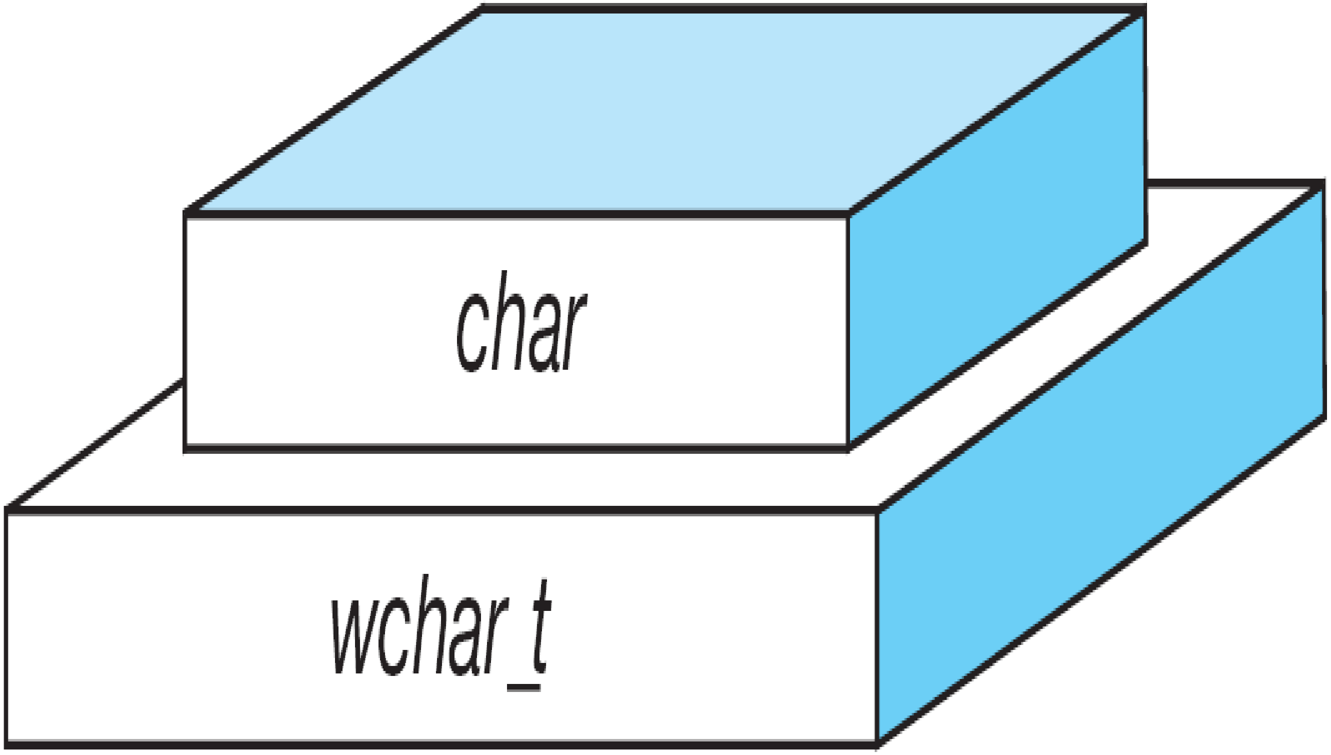
sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Integer Types

Built-in data types (contd…)

Character Data Type :

- The shortest data type is character.
- The keyword char is used to declare a variable of a character type.
- It is stored in 1 byte in memory.
- Corresponding integer values for all characters are defined in ASCII (American Standard Code for Information Interchange).
- **Example:** character constant ‘a’ has an int value 97,
  ‘b’ has 98, ‘A’ has 65 etc.
- Character can have values in the range of **-128** to **127**.



Character Types

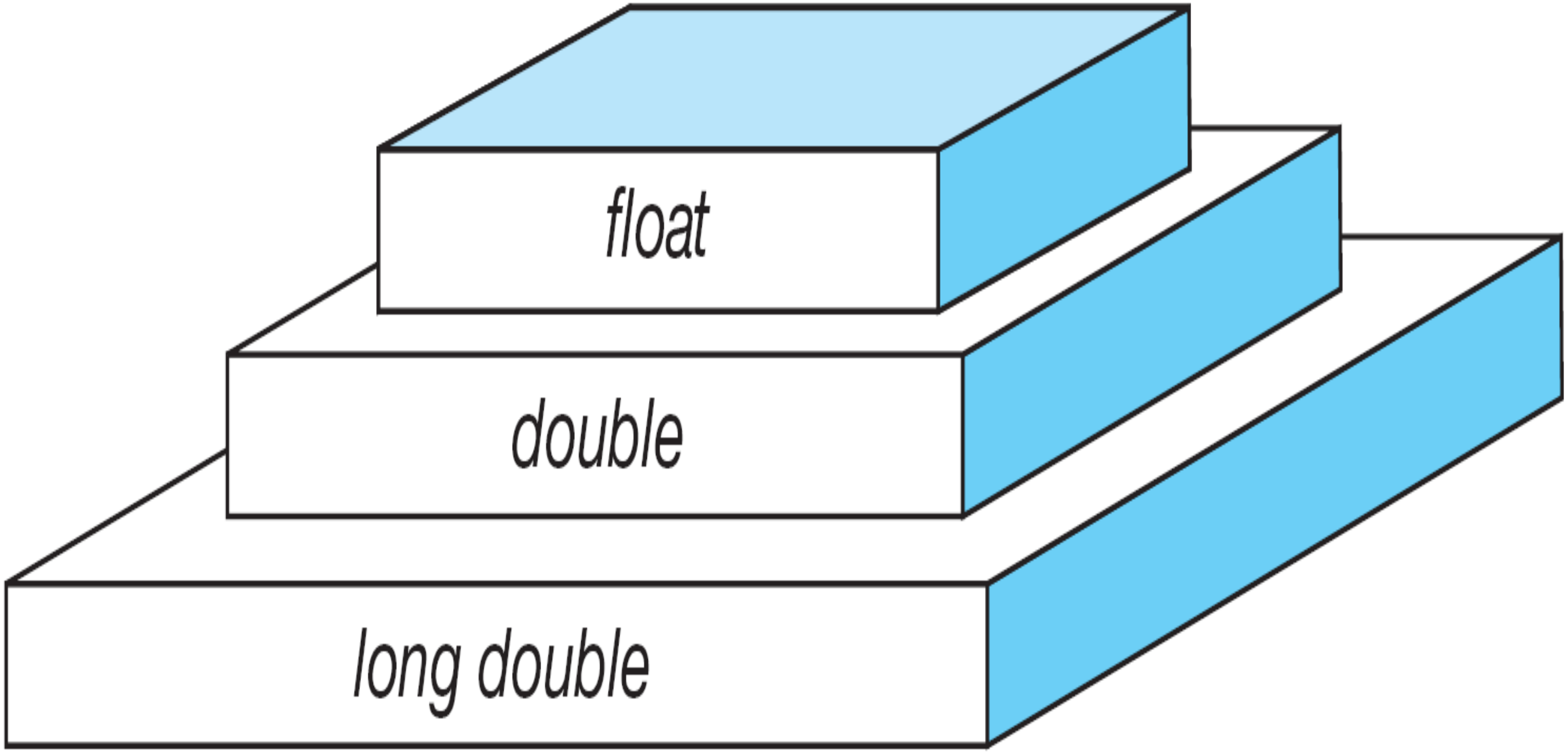
Built-in data types (contd…)

The floating point data type:

- The keyword **float** is used to declare a variable of the type float.
- The float type variable is usually stored in 32 bits, with 6 digits of precision.
- A float variable can have values in the range of **3.4E-38** to **3.4 E+38**.

The double data type:

- A floating point number can also be represented by the double data type.
- The data type double is stored on most machines in 64 bits which is about 15 decimal places of accuracy.
- To declare a variable of the type double, use the keyword **double**.
- A double variable can have values in the range of **1.7E-308** to **+1.7E+308**.



Note

sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Floating-point Types

Type Modifiers:

- The basic data types may have various **modifiers** (or **qualifiers**) preceding them, except type ‘void’.

- A **modifier** is used to alter the meaning of the base data type to fit the needs of various situations more precisely.

- The modifiers **signed**, **unsigned**, **long, short** may be applied to **integer** base types.

- The modifiers **unsigned and signed** may be applied to **characters**.

- The modifier **long** may also be applied to **double**.

- The difference between signed and unsigned integers is in the way high-order bit (sign bit) of the integer is interpreted.

- If sign bit is 0, then the number is positive; if it is 1, then the number is negative.

Data types in C (contd…)

Type	Size (bytes)	Format specifier	Range
short int	2	%hd	-32,768 to 32767
unsigned short int	2	%hu	0 to 65535
int	2	%d	-32,768 to 32,767
unsigned int	2	%u	0 to 65535
long int	4	%ld	-2,147,483,648 to 2,147,483,647
unsigned long int	4	%lu	0 to 4,294,967,295
long long int	8	%lld	-(2^63) to (2^63)-1
unsigned long long int	8	%llu	0 to (2^64)-1

- Floating types are used to store real numbers.

Type	Size(bytes)	Format specifier	Range
Float	4	%f	3.4E-38 to 3.4E+38
double	8	%lf	1.7E-308 to 1.7E+308
long double	10	%Lf	3.4E-4932 to 1.1E+4932

- Character types are used to store characters value.
- **Size and range of Integer type on 16-bit machine**

Type	**Size (bytes)**	**Format specifier**	**Range**
char or signed char	1	%c	-128 to 127
unsigned char	1	%c	0 to 255

Derived data types and User defined data types are the combination of primitive data types. They are used to represent a collection of data.

They are:

- Arrays
- Pointers
- Structures
- Unions
- Enumeration

Note: Number of bytes and range given to each data type is platform dependent.

Variables

- Variable is a valid identifier which is used to store the value in the memory location, that value varies during the program execution.

Types of variables:

- Global Variables
- Local Variables

- **Global Variable:** The variables which are declared at the starting of the program are called as global variable. They are visible to all the parts of the program.

- **Local Variable:** The variables which are declared in a function are called local variables to that function. These variables visible only within the function.

Variables (contd…)

Variable Declaration & Definition:

- Each variable in your program must be declared and defined.
- In C, a declaration is used to name an object, such as a variable.
- Definitions are used to create the object.
- When you create variables, the declaration gives them a symbolic name and the definition reserves memory for them.
- A variable's type can be any of the data types, such as character, integer or real except void.
- C allows multiple variables of the same type to be defined in one statement.

  Example: int a, b;

Variable Initialization:

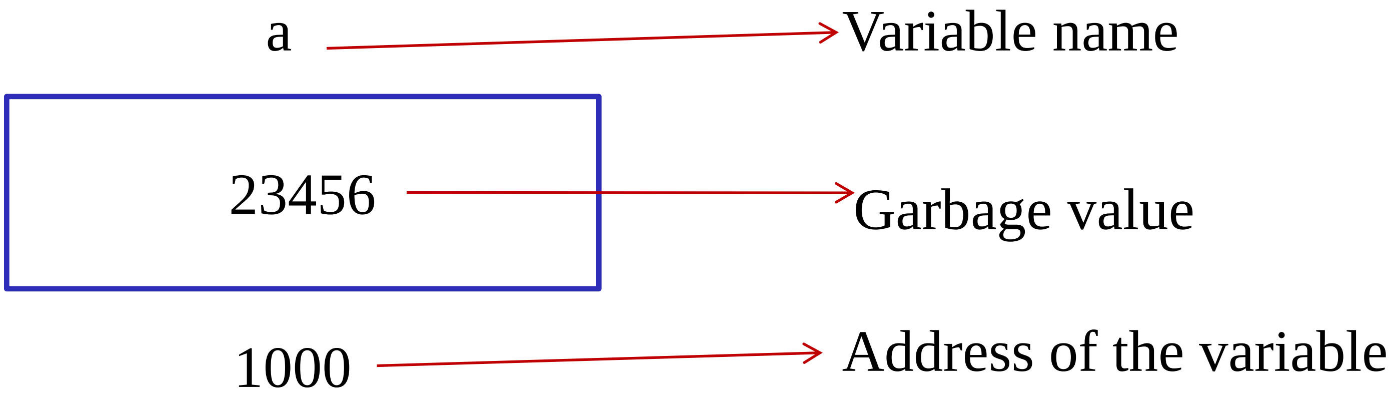
- You can initialize a variable at the same time that you declare it by including an initializer.
- To initialize a variable when it is defined, the identifier is followed by the assignment operator and then the initializer.

  Example: int count = 0;

Variables (contd…)

Variable declaration and definition:

Example: int a;



Variable initialization:

datatype identifier = initial value;

Examples:

int a=10;
float b=2.1;
float pi=3.14;
char ch=‘A’;

Variables (contd…)

- When you want to process some information, you can save the values temporarily in variables.

```
#include<stdio.h>
main()
{
       int a=10,b=20,c;
       c=a+b;
       printf(“sum of a and b=%d\n”,c);
       return 0;
}
```

- **Note** : Variables must be declared before they are used, usually at the beginning of the function.

Variables (contd…)

There are some restrictions on the variable names (same as identifiers):

- First character must be alphabetic character or underscore.
- Must consist only of alphabetic characters, digits, or underscore.
- Should not contain any special character, or white spaces.
- Should not be C keywords.
- Case matters (that is, upper and lowercase letters). Thus, the names **count** and **Count** refer to two different identifiers.

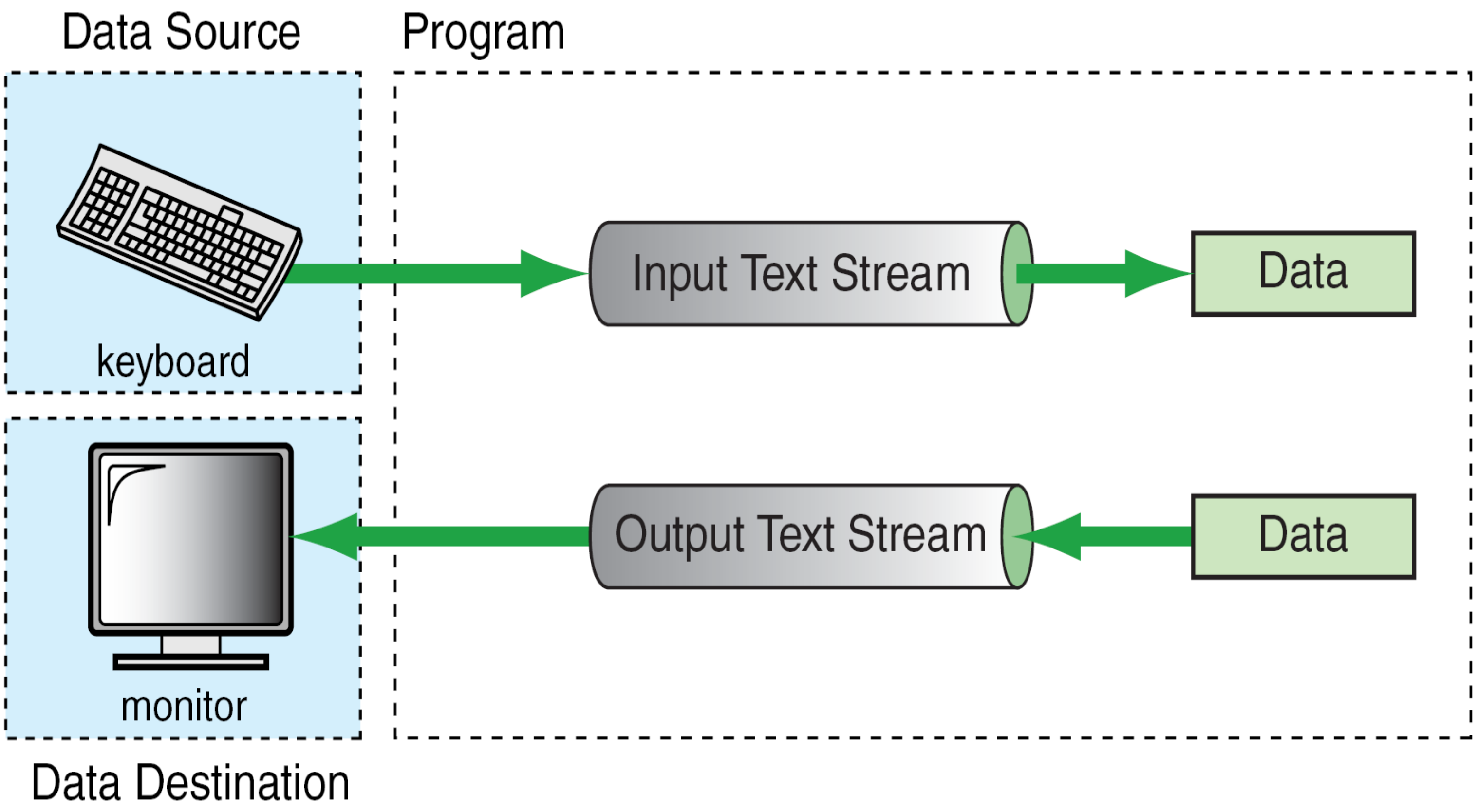
- The following list contains some examples of legal and illegal C variable names:

Variable Name	**Legality**
Percent	Legal
y2x5__fg7h	Legal
annual_profit	Legal
_1990_tax	Legal but not advised
savings#account	Illegal: Contains the illegal character #
double	Illegal: It is a C keyword
9winter	Illegal: First character is a digit

Formatted input/output

- The data is to be arranged in a particular format.
- The data is input to and output from a stream.
- A stream is a source or destination of the data, it is associated with a physical device such as terminals (keyboard, monitor).
- C has two forms of streams: **Text Stream** and **Binary Stream**.
- **Text Stream** consists of sequence of characters.
- **Binary Stream** consists of a sequence of data in 0’s and 1’s.

- A terminal **keyboard** and **monitor** can be associated only with a **text stream**.
- A keyboard is a **source** for a text stream; a monitor is a **destination** for a text stream.



Stream Physical Devices

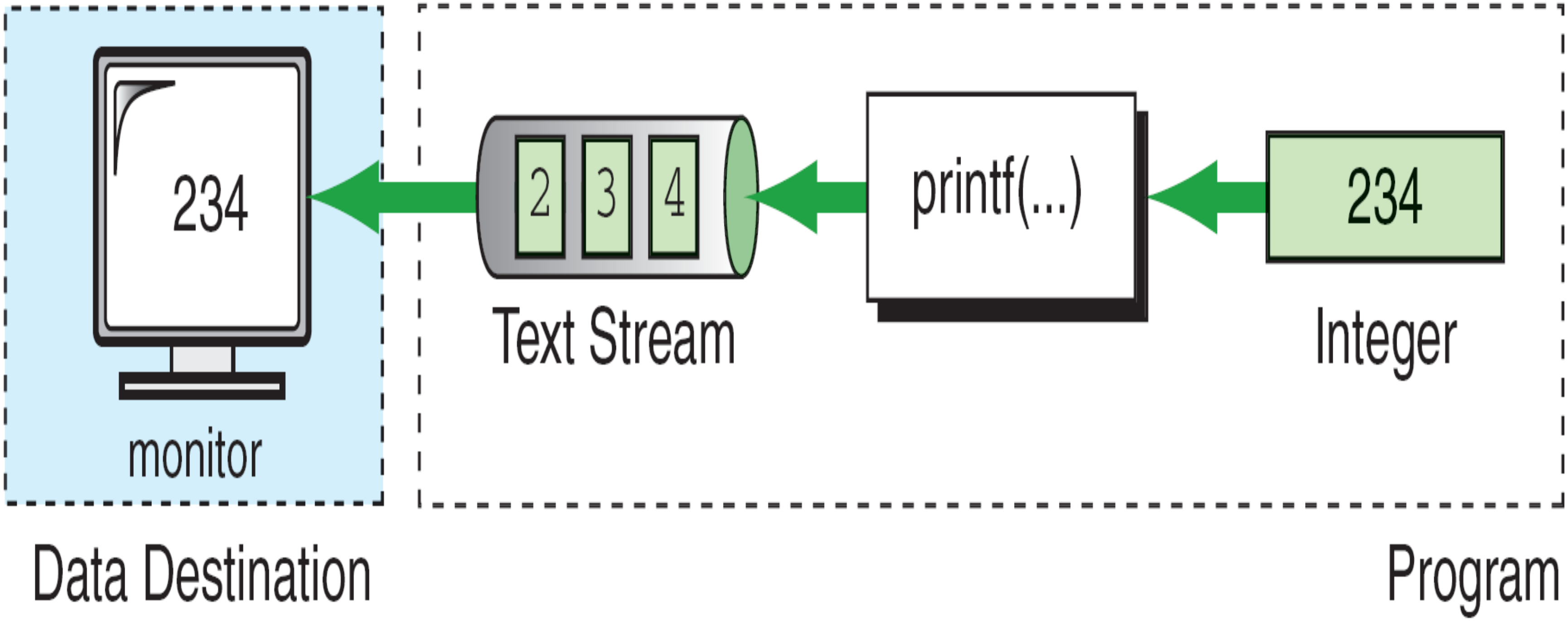
- The data is formatted using the printf and scanf functions.

- scanf() converts the text stream coming from the keyboard to data values (char, int, etc.,) and stores them in the program variables.

- printf() converts the data stored in variables into the text stream for output the keyboard.

- **printf()** takes the set of data values and converts them to text stream using formatting instructions contained in a **format control string**.

- **Format control specifiers** are used to format the text stream.



Output Formatting Concept

- Format specifier specifies the data values type, size and display position.

- Printf statement takes two arguments

  1. Control String and

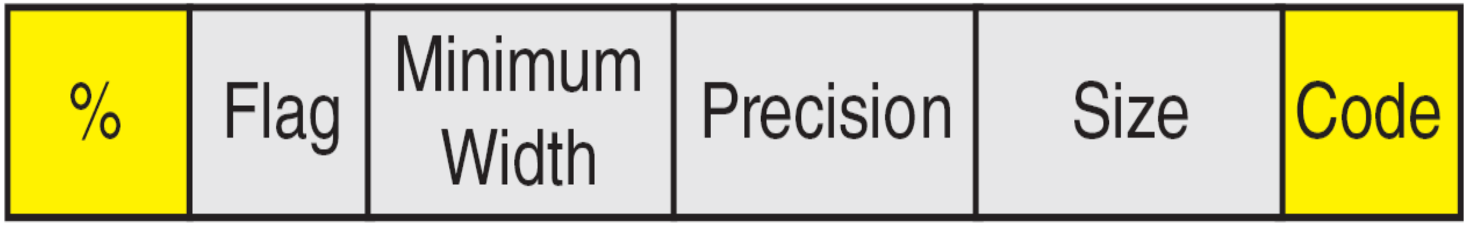
  2. Data Values(Variables)

- Control string contains the format specifiers and some text.

  Syntax: printf(“control string”,var1,var2,…,varn);

  Example:

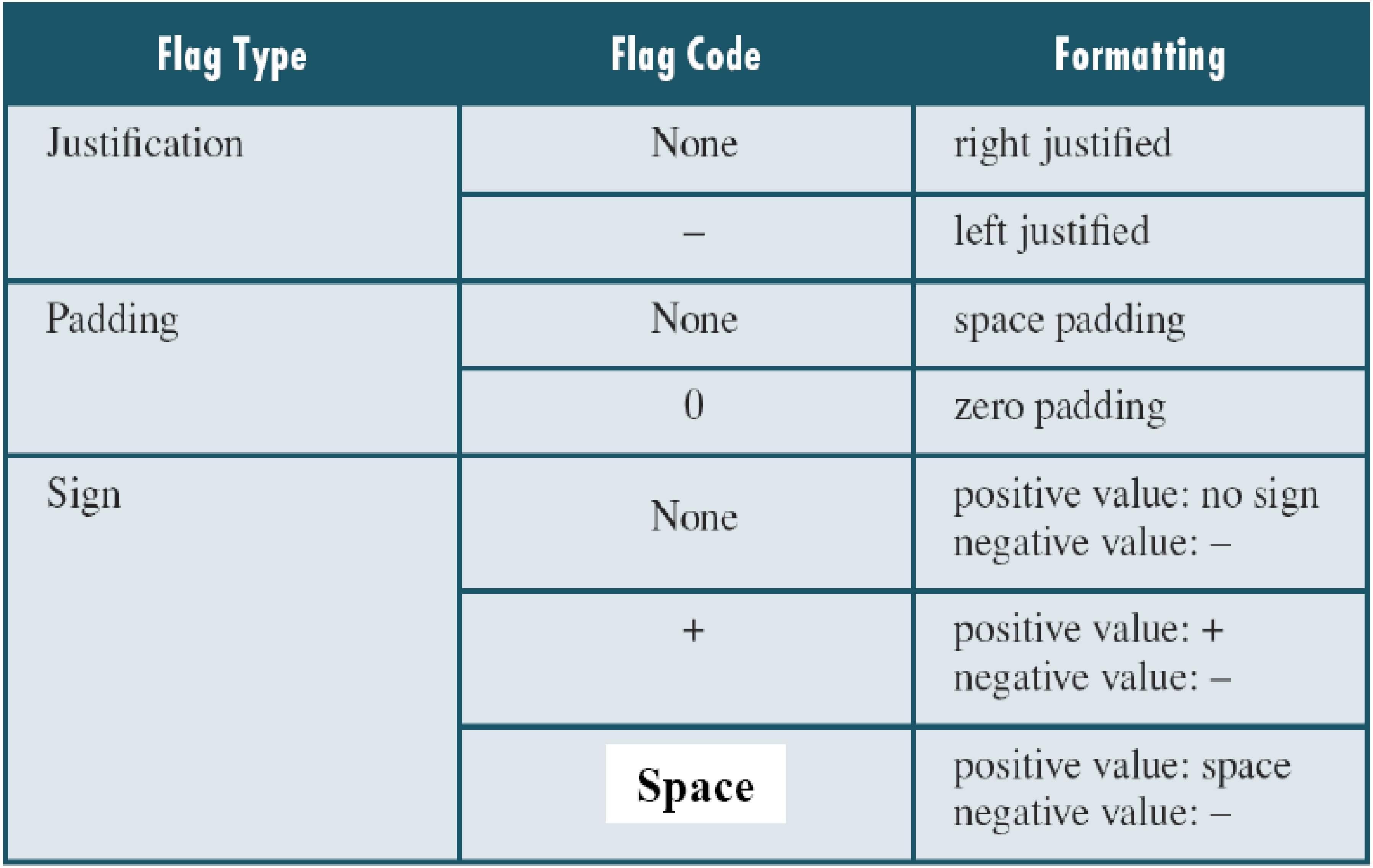
  ```
  int a=10, b=20;
  printf(“a=%d  b=%d”, a,b);
  ```



%	Flag	Minimum Width	Precision	Size	Code

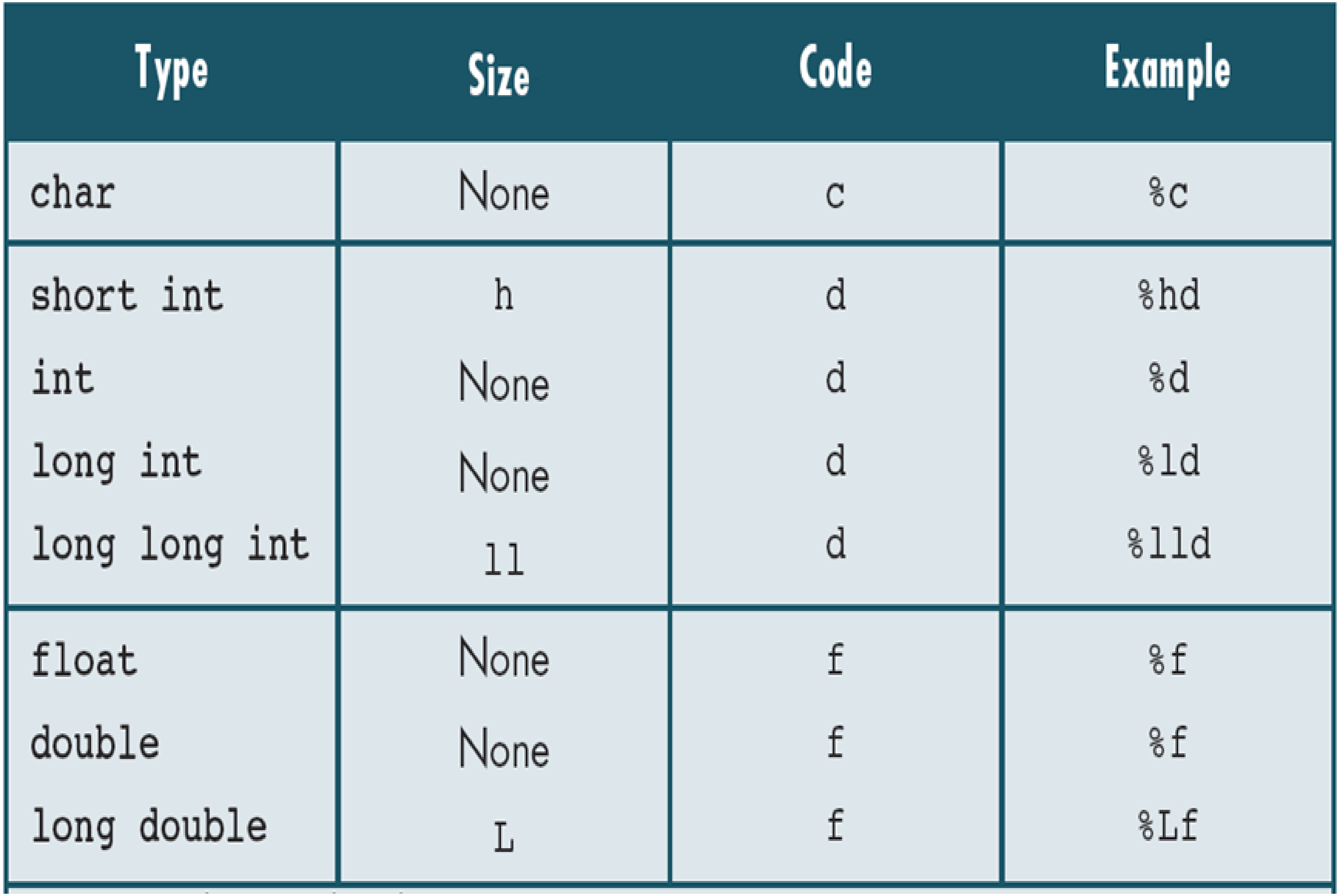
- ➢The percent sign and conversion code are required but other modifiers such as width and precision are optional.
- ➢The width modifier specifies the total number of positions used to display the value.
- ➢The precision modifier indicates the number of characters should be printed after decimal point.The precision option is only used with floats or strings.

Conversion Specification



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: –
	Space	positive value: space negative value: –

Flag Formatting Options



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

Format Codes for Output

Format

printf(“%d”,9876);

9	8	7	6

printf(“%6d”,9876);

		9	8	7	6

printf(“%2d”,9876);

9	8	7	6

printf(“%-6d”,9876);

9	8	7	6		

printf(“%06d”,9876);

0	0	9	8	7	6

int y=98.7654;

printf(“%7.4f”,y);

9	8	.	7	6	5	4

printf(“%7.2f”,y);

		9	8	.	7	7

printf(“%f”,y);

9	8	.	7	6	5	4

Output Examples

```
1. printf("%d%c%f", 23, 'z', 4.1);
```

```
23z4.100000
```

```
2. printf("%d %c %f", 23, 'z', 4.1);
```

```
23 z 4.100000
```

```
3. int num1 = 23;
   char zee = 'z';
   float num2 = 4.1;
   printf("%d %c %f", num1, zee, num2);
```

```
23 z 4.100000
```

```
4. printf("%d\t%c\t%5.1f\n", 23, 'z', 14.2);
   printf("%d\t%c\t%5.1f\n", 107, 'A', 53.6);
   printf("%d\t%c\t%5.1f\n", 1754, 'F', 122.0);
   printf("%d\t%c\t%5.1f\n", 3, 'P', 0.1);
```

```
23      z        14.2
107     A        53.6
1754    F       122.0
3       P         0.1
```

```
5. printf("The number%dis my favorite.", 23);
```

```
The number23is my favorite.
```

```
6. printf("The number is %6d", 23);
```

```
The number is     23
^^^^^^^^^^^^^^^^^^^^
```

7. `printf("The tax is %6.2f this year.", 233.12);`

```
The tax is 233.12 this year.
```

8. `printf("The tax is %8.2f this year.", 233.12);`

```
The tax is   233.12 this year.
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

9. `printf("The tax is %08.2f this year.", 233.12);`

```
The tax is 00233.12 this year.
```

10. `printf("\"%8c   %d\"", 'h', 23);`

```
"       h   23"
^^^^^^^^^^^^^^^
```

```
11. printf("This line disappears.\r...A new
    line\n");
    printf("This is the bell character \a\n");
    printf("A null character\0kills the rest of the
    line\n");
    printf("\nThis is \'it\' in single quotes\n");
    printf("This is \"it\" in double quotes\n");
    printf("This is \\ the escape character it
    self\n");
```

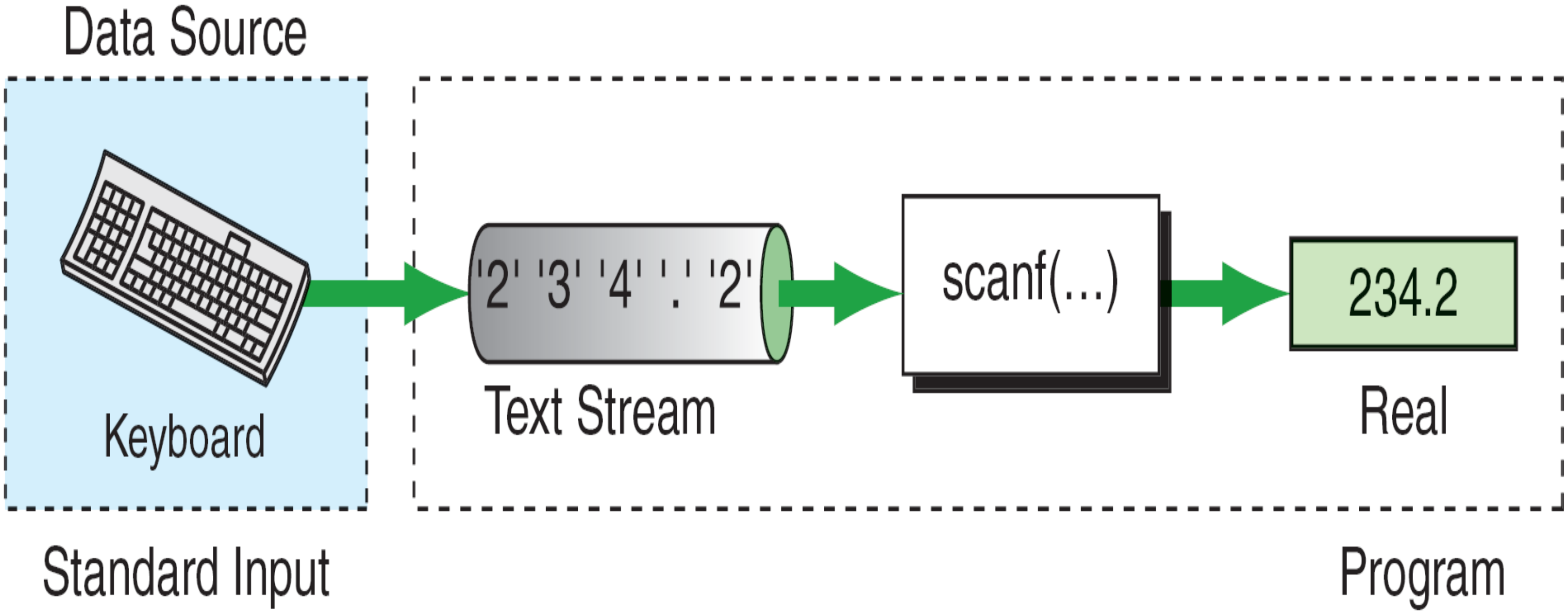
```
...A new line
This is the bell character
A null character
This is 'it' in single quotes
This is "it" in double quotes
This is \ the escape character it self
```

```
12. printf("|%-+8.2f| |%0+8.2f| | %-0+8.2f|", 1.2,
    2.3, 3.4);
```

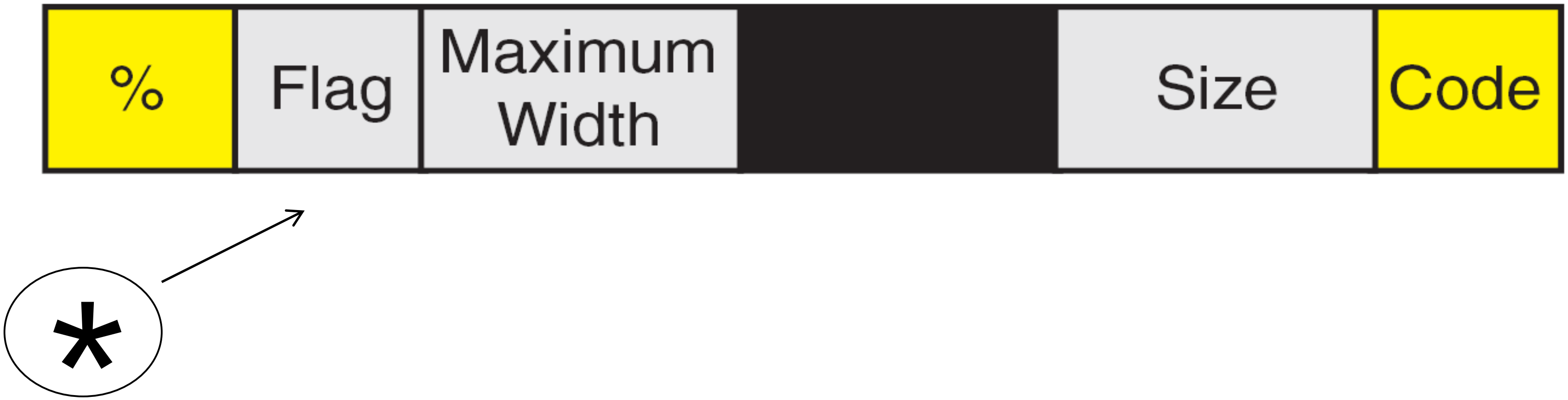
```
|+1.20   | |+0002.30| | +3.40   |
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

scanf("control string",&var1,&var2.. &varn);

- control string includes format specifiers and specifies the field width.
- scanf requires variable addresses in the address list.



Formatting Text from an Input Stream



Flag(*):An optional modifier * ,which state that the value being will not be assigned to an variable, but will be dropped

Width: An optional width specifier ,which designates the maximum number of characters to be read that compose the value for the variable.

Conversion Specification

Input Examples

1. 214 156 14z

```
scanf("%d%d%d%c", &a, &b, &c, &d);
```

2. 214 156 14 z

```
scanf("%d%d%d %c", &a, &b, &c, &d);
```

3. 2314 15 2.14

```
scanf("%d %d %f", &a, &b, &c);
```

4. 14/26 25/66

```
scanf("%2d/%2d %2d/%2d", &num1, &den1, &num2,
&den2);
```

5. 11-25-56

```
scanf("%d-%d-%d", &a, &b, &c);
```

Exercise:

1. Sum of the digits of a number.
2. Nature of quadratic equations.
3. Income tax problem.
4. Standard deviation.