Programming Using C
Week 01

Introduction
Constants , Variables & Datatypes

Prepared By: REC Faculty 4.0 Team

TOPICS

- INTRODUCTION TO C LANGUAGE
- CHARACTER SET
- CONSTANTS
- VARIABLES
- DATATYPES(PRIMITIVE)
- SAMPLE PROGRAMS & MCQ

INTRODUCTION- C LANGUAGE

- C is a programming language developed at AT & T's Bell Laboratories of USA in 1972.
- It was designed and written by Dennis Ritchie.
- C was limited to use within Bell Laboratories until 1978, when Brian Kernighan and Ritchie published a definitive description of the language.
- C is a general purpose programming language
- C is a Procedural Language.

WHY C IS POPULAR?

- C is a popular language since it is simple, reliable and easy to use. It is a language which has survived for more than 3 decades even when new languages, tools and technologies have evolved.
- Major parts of popular operating systems like Windows, UNIX, Linux are written in C.
- Common consumer devices like microwave oven, washing machines and digital cameras are getting smarter day by day. This smartness comes from a microprocessor, an operating system and a program embedded in this devices. Such programs are written in Embedded C.
- Many popular gaming frameworks have been built using C language.
- To closely interact with hardware devices C language provides features and also make these interactions feasible without compromising performance.

WHY C?

- Most Companies expect the candidate to be aware of this language.
- Forms the Base for learning other languages.

STRUCTURE OF A C PROGRAM

STRUCTURE	SAMPLE C PROGRAM
Documentation Section	//Sample Program

Link Section	#include<stdio.h>
	#include<conio.h>

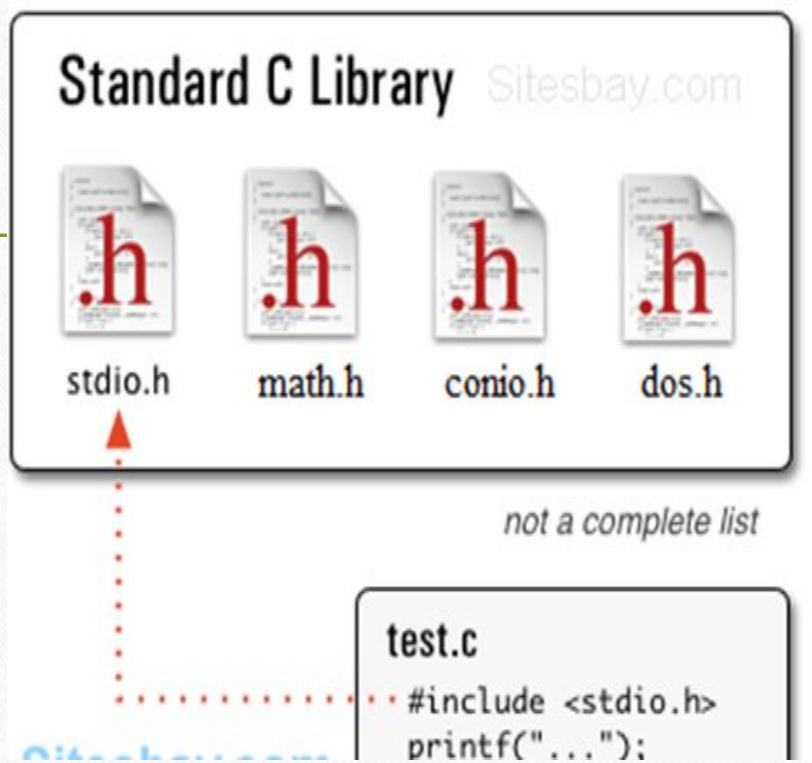
Function Declaration Section	int func1();
Global Declaration Section	int a=10;

Main Section	int main ()
	{
	clrscr();
	printf (“the value of a is inside main%d”,a);
	int x=fun();
	}

Subroutine Section	int fun ()
	{
	printf (“the value of a is inside func1%d”, a);
	}

HEADER FILES

- A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.
- There are two types of header files:
  - Files that comes with the compiler(predefined)
  - Files that are written by the Programmer



```
#include <stdio.h>
printf("...");
```

COMMENTS

- In C, comments can be placed anywhere in the program which are not executed as a part of the program.
- Comments are used for **the better understandability of the program** by others and also helps in better debugging of the program.
- Adding Comments to a program is a highly recommended practice.

SYNTAX OF COMMENT

Single line Commenting can be done in 2 ways,

```
/* Comments here */
```

```
//Comment goes here
```

Multiple Line Commenting is done as follows,

```
/*Comment in line 1
Comment in line 2
Comment in line n */
```

```
/* Sample program-find area of circle
Programming Language-C */
```

CODE INDENDATION

- Indentation is one of the most important aspects in any programming domain. Indentation is a **way to organize and document your source code.**
- Proper code indentation will make it:
  - Easier to read
  - Easier to understand
  - Easier to modify
  - Easier to maintain
  - Easier to enhance

CODE INDENDATION

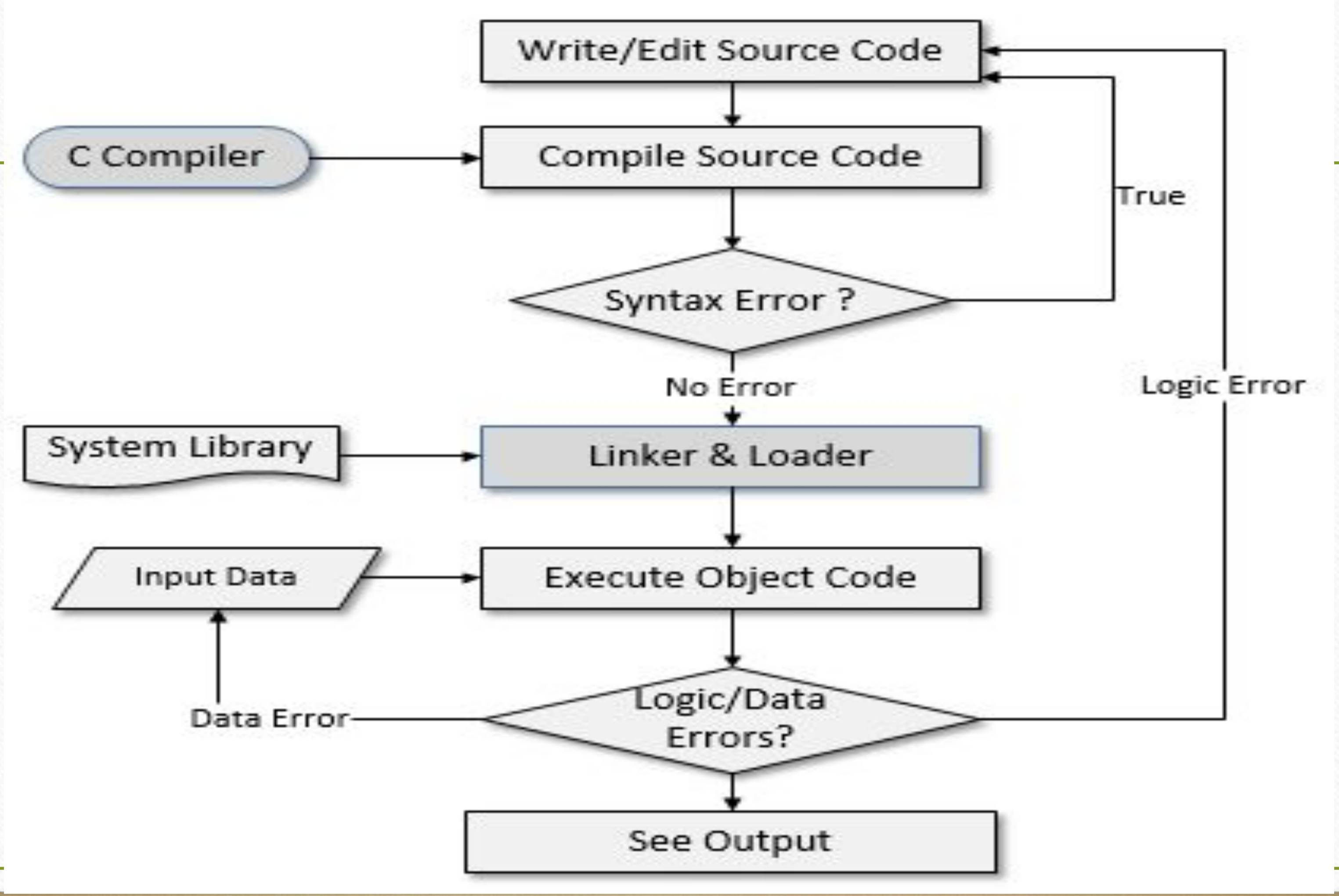
Example 1:

```
int main (int arg c, char *argv[])
{
        int   a,
        float  b;
        printf("Give me an 'a' : ");
        scanf("%d",&a);
        printf(" Give me a 'b' : ");
        scanf("%f",&b);
        . . .
}
```

Example 2:

```
    if (victor(human))
    {       human_wins++;
    printf("humble servant.\n");
    }
    else
    {       computer_wins++;
            printf("Your destiny \n");
    }
```

COMPILATION AND EXECUTION



C COMPILERS

- A compiler is system software that converts source code written in a programming language (source language, usually HLL) into computer language (machine language/assembly code).

- **ANSI C**, **ISO C**, and **Standard C** are successive standards for the C programming language published by the American National Standards Institute (ANSI) and (ISO) and the International Electrotechnical Commission (IEC).

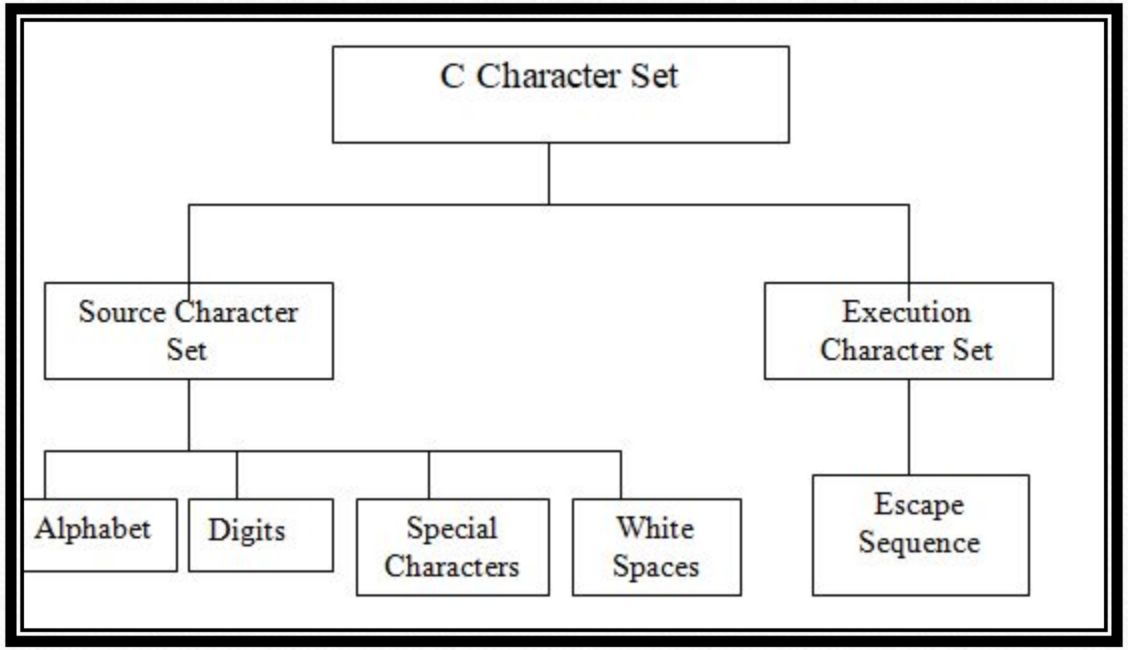
- Historically, the names referred specifically to the original and best-supported version of the standard (known as **C89** or **C90**). Software developers writing in C are encouraged to conform to the standards, as doing so helps portability between compilers.

Example of C Compilers: Turbo GCC Tiny C Clang

COMPILER VS INTEPRETER

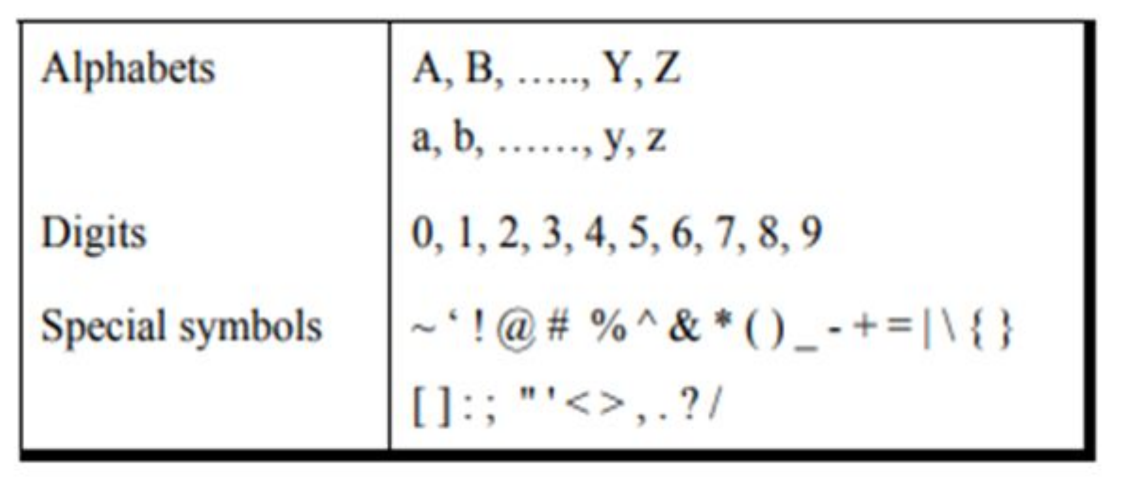
INTERPRETER	COMPILER
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

CHARACTER SET



SOURCE CHARACTER SET

- A character denotes any alphabet, digit or special symbol used to represent information. The following Figure shows the valid alphabets, numbers and special symbols allowed in C.
- The alphabets, numbers and special symbols when properly combined form **tokens**.



<table>
<tr><th></th><th></th></tr>
<tr><td>Alphabets</td><td>A, B, ….., Y, Z
a, b, ……, y, z</td></tr>
<tr><td>Digits</td><td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td></tr>
<tr><td>Special symbols</td><td>~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { }
[ ] : ; " ' < > , . ? /</td></tr>
</table>

EXECUTION CHARACTER SET

- It is also called as “**Escape sequences**”.
- This set of characters are also called as non graphic characters because, these characters are invisible and cannot be printed directly.
- These characters will have effect only when the **program is executed**.

CODE	MEANING
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\0	Null
\\	Backslash
\v	Vertical Tab
\a	Alert

EXECUTION CHARACTER SET-Example Program

```
#include <stdio.h>
int main()
{
    printf("\"Hello World");
    return 0;
}
```

OUTPUT:

```
"Hello World
```

```
#include <stdio.h>
int main()
{
    printf("\”Wel\tcome to \nREC\'");
    return 0;
}
```

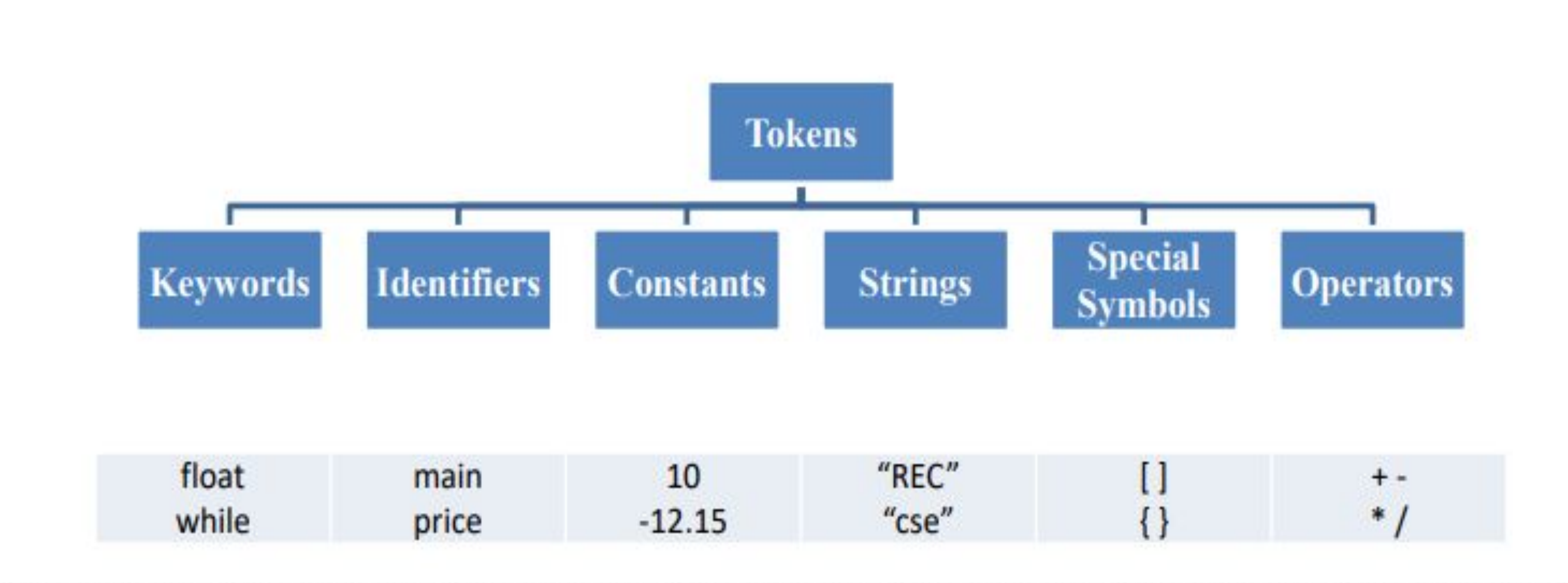
OUTPUT:

```
"Wel    come to
REC'
```

C TOKENS

- In C program, the smallest individual units are known as C tokens.

- **Types of Tokens:**



Tokens

Keywords

Identifiers

Constants

Strings

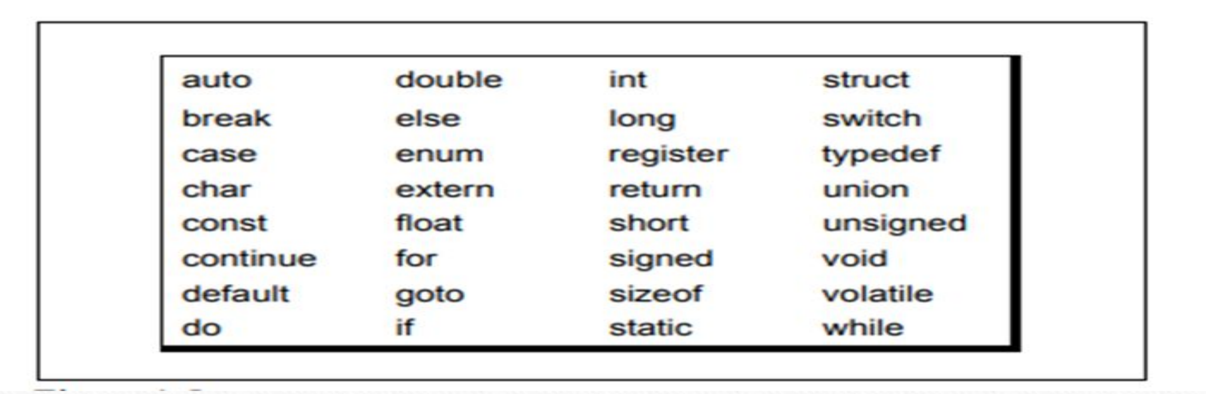
Special Symbols

Operators

float while	main price	10 -12.15	“REC” “cse”	[ ] { }	+ - * /

C KEYWORDS

- Keywords are the words whose meaning has already been explained to the C compiler.
- The keywords cannot be used as variable names .The keywords are also called ‘Reserved words’.
- There are only 32 keywords available in C. The following Figure gives a list of these keywords for your ready reference.



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers in C

- Used for naming variables, functions, arrays, structures, etc.
- Identifiers in C are the user-defined words.
- **Rules** for constructing identifiers in C
  - The first character of an identifier should be either an **alphabet** or an **underscore**, and then it can be followed by any of the character, digit, or underscore.
  - It should **not** begin with any **numerical** digit.
  - In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
  - **Commas or blank spaces cannot** be specified within an identifier.
  - **Keywords cannot** be represented as an identifier.

Strings in C

- Sequence of characters.
- Strings in C are enclosed within **double quotes**, while characters are enclosed within **single characters**.

Example:

“Hello”

‘A’

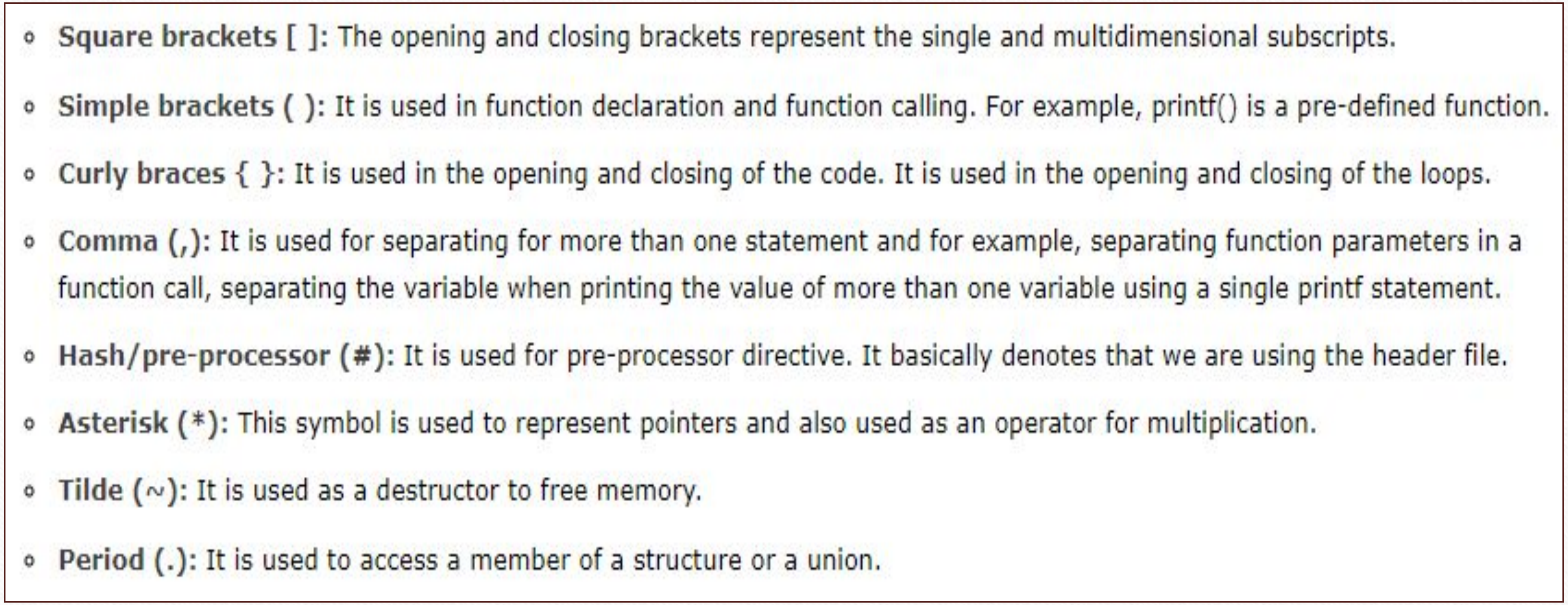
Operators in C

- Symbol used to perform the operation

- Example:

  +,-,/,*,%,++,--,<<,>>,&&,||,!

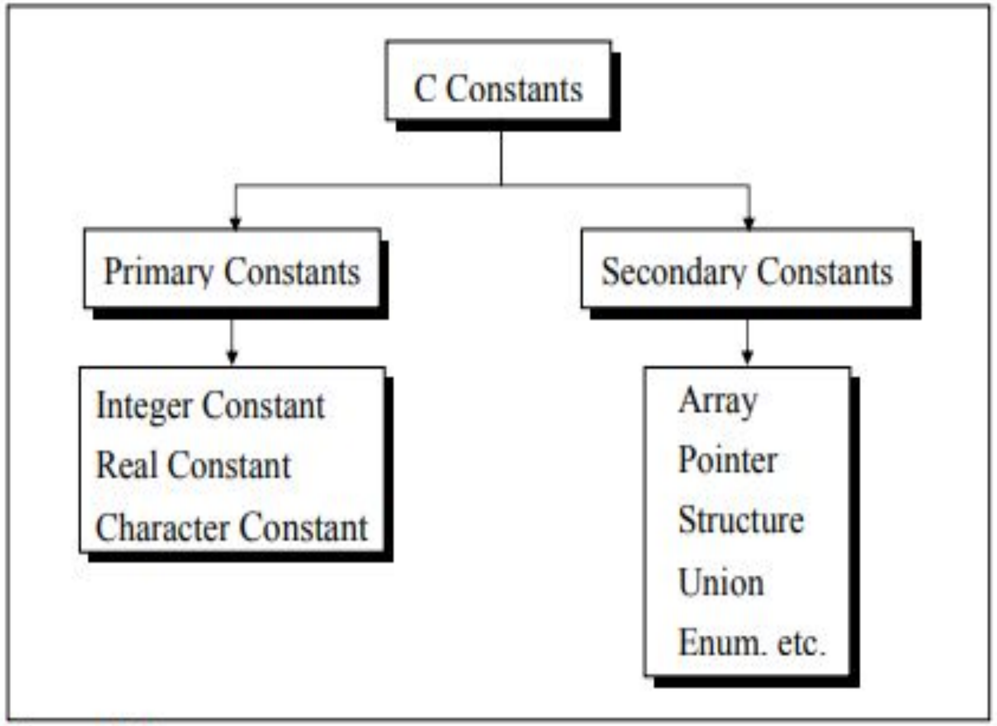
Special Symbols



- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

CONSTANTS

- A constant is an entity that doesn’t change its value.
- C constants can be divided into two major categories:
  - Primary Constants
  - Secondary Constants



C Constants

Primary Constants

Secondary Constants

Integer Constant
Real Constant
Character Constant

Array
Pointer
Structure
Union
Enum. etc.

VARIABLES

- Variable is a name used for storing a data value.
- Its value may be changed during the program execution.
- Variable names are names given to locations in memory.
- These locations can contain integer, real or character constants.
- Rules for defining variables is same as Identifier.

VARIABLES

DECLARING A VARIABLE:

- The declaration of variables must be done before they are used in the program.
- It tells the compiler what the variable name is and it specifies what type of data the variable will hold.
- A variable declaration consists of a data type name followed by a list of one or more variables of that type separated by commas.
- **Syntax:**
- datatype var1, var2, . . . varN;// here var1,var2..varN are names of the variable
- **Eg: int sum;**
- The variables will contain some garbage value when they are declared.

VARIABLES

ASSIGNING VALUES TO VARIABLE:

Values can be assigned to variables using the assignment operator =.

Syntax:

```
variablename = value;
int x;
x=100;
```

VARIABLES

INITIALIZING VALUE TO VARIABLE:

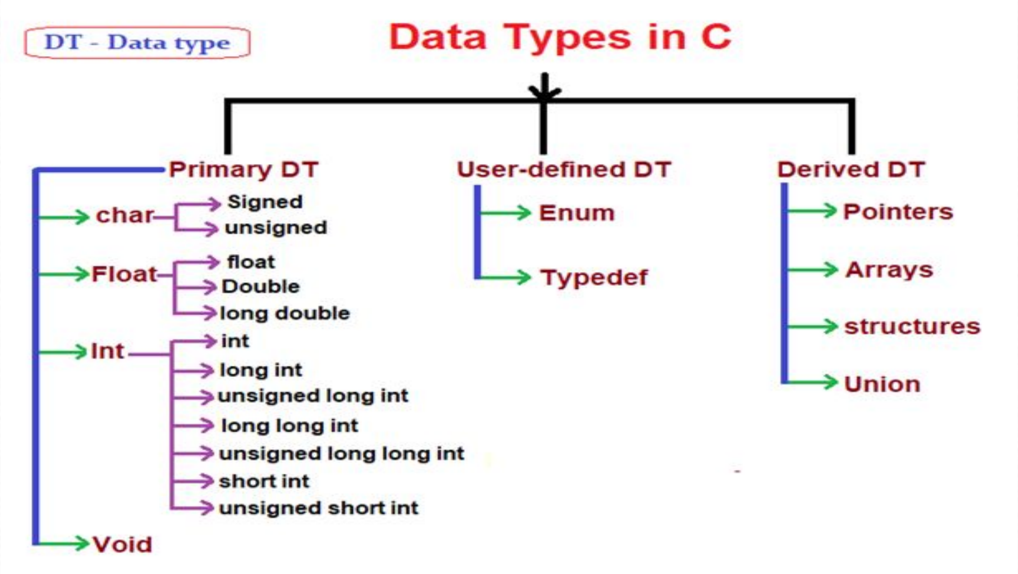
- The process of giving an initial value to variables is called initialization.
- C permits the initialization of more than one variable in one statement using multiple assignment operators.

Example:

```
int a = 0;
int x,y,z;
 x = y = z = 1;
```

DATA TYPES IN C

- A data type specifies the type of data that a variable can store such as integer, floating, character, etc.. ANSI C provides three types of data types:



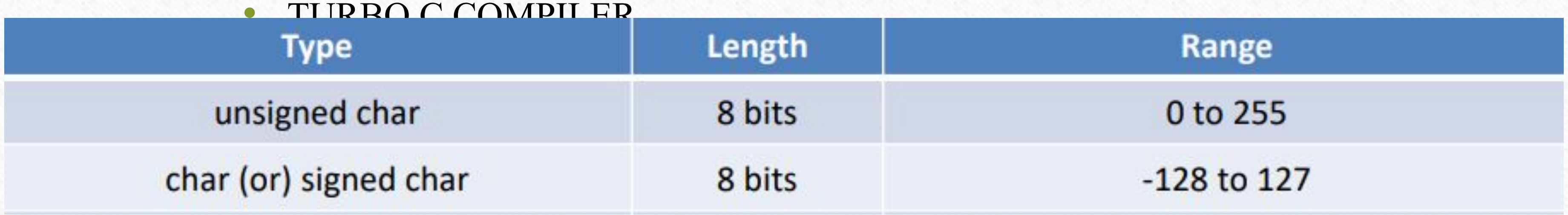
DT - Data type

Data Types in C

- Primary DT
  - char
    - Signed
    - unsigned
  - Float
    - float
    - Double
    - long double
  - Int
    - int
    - long int
    - unsigned long int
    - long long int
    - unsigned long long int
    - short int
    - unsigned short int
  - Void
- User-defined DT
  - Enum
  - Typedef
- Derived DT
  - Pointers
  - Arrays
  - structures
  - Union

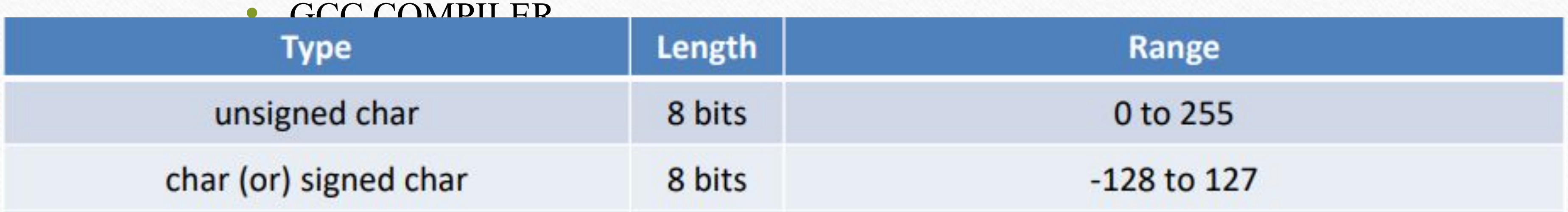
PRIMARY (BUILT-IN) DATA TYPES:

- **char:**
- The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
- Range indicates the values that can be stored inside the variable of a given type.
  - TURBO C COMPILER



Type	Length	Range
unsigned char	8 bits	0 to 255
char (or) signed char	8 bits	-128 to 127

  - GCC COMPILER

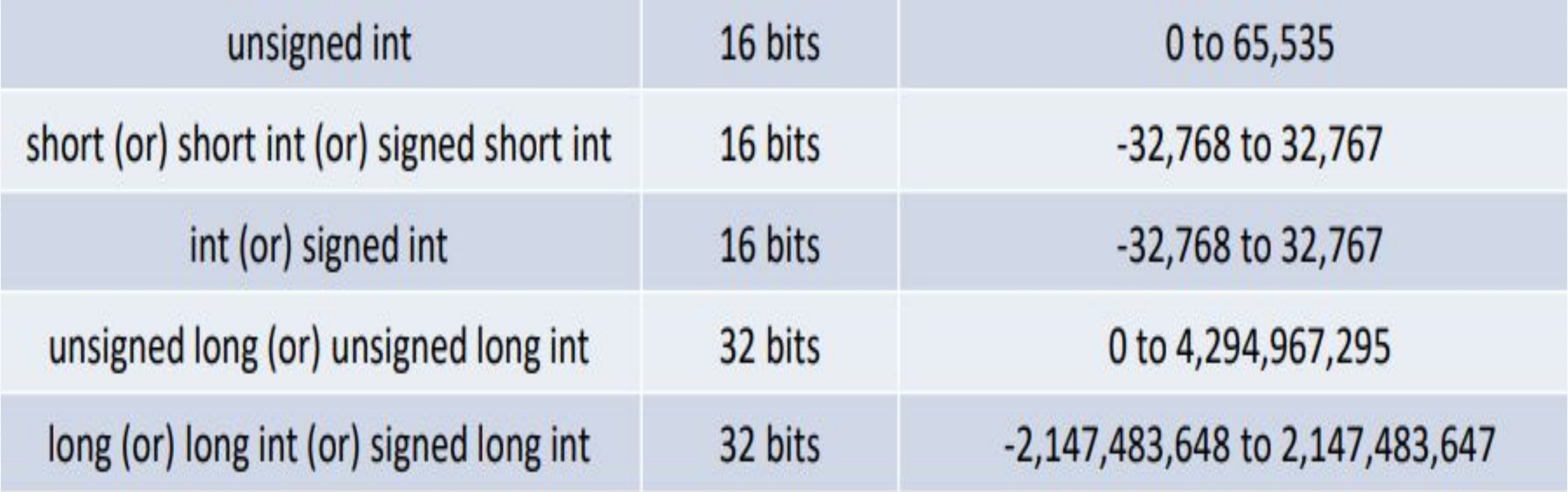


Type	Length	Range
unsigned char	8 bits	0 to 255
char (or) signed char	8 bits	-128 to 127

PRIMARY (BUILT-IN) DATA TYPES:

int:

- int datatype is used to store whole numbers.
- TURBO C COMPILER

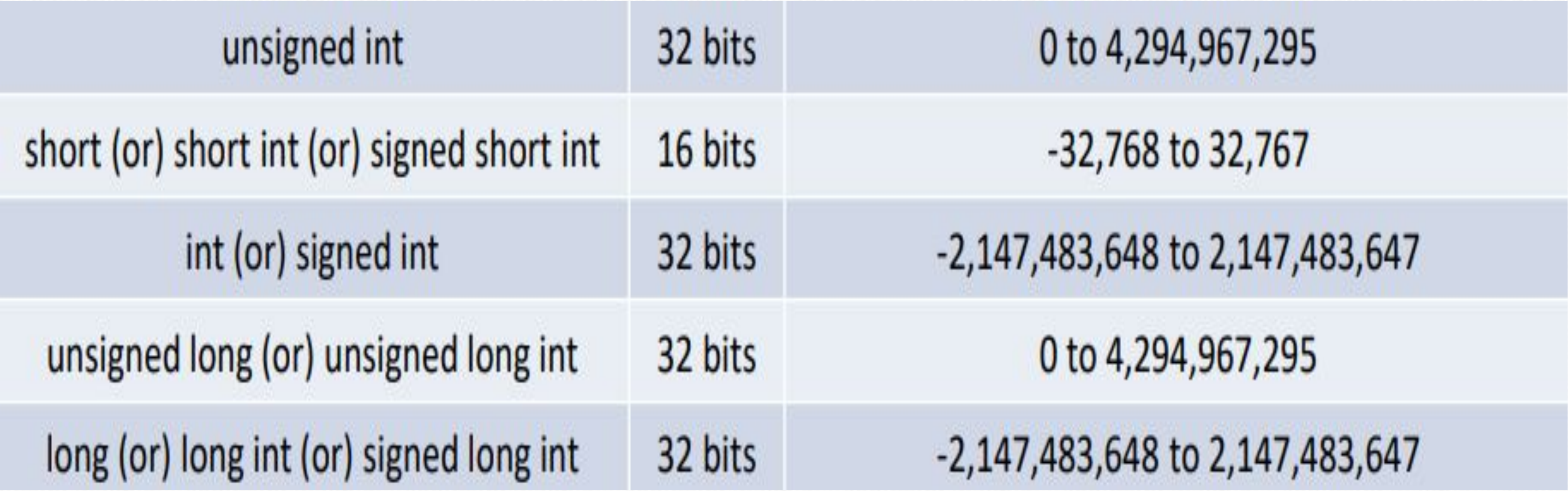


unsigned int	16 bits	0 to 65,535
short (or) short int (or) signed short int	16 bits	-32,768 to 32,767
int (or) signed int	16 bits	-32,768 to 32,767
unsigned long (or) unsigned long int	32 bits	0 to 4,294,967,295
long (or) long int (or) signed long int	32 bits	-2,147,483,648 to 2,147,483,647

PRIMARY (BUILT-IN) DATA TYPES:

int:

- int datatype is used to store whole numbers.
- GCC COMPILER



unsigned int	32 bits	0 to 4,294,967,295
short (or) short int (or) signed short int	16 bits	-32,768 to 32,767
int (or) signed int	32 bits	-2,147,483,648 to 2,147,483,647
unsigned long (or) unsigned long int	32 bits	0 to 4,294,967,295
long (or) long int (or) signed long int	32 bits	-2,147,483,648 to 2,147,483,647

PRIMARY (BUILT-IN) DATA TYPES:

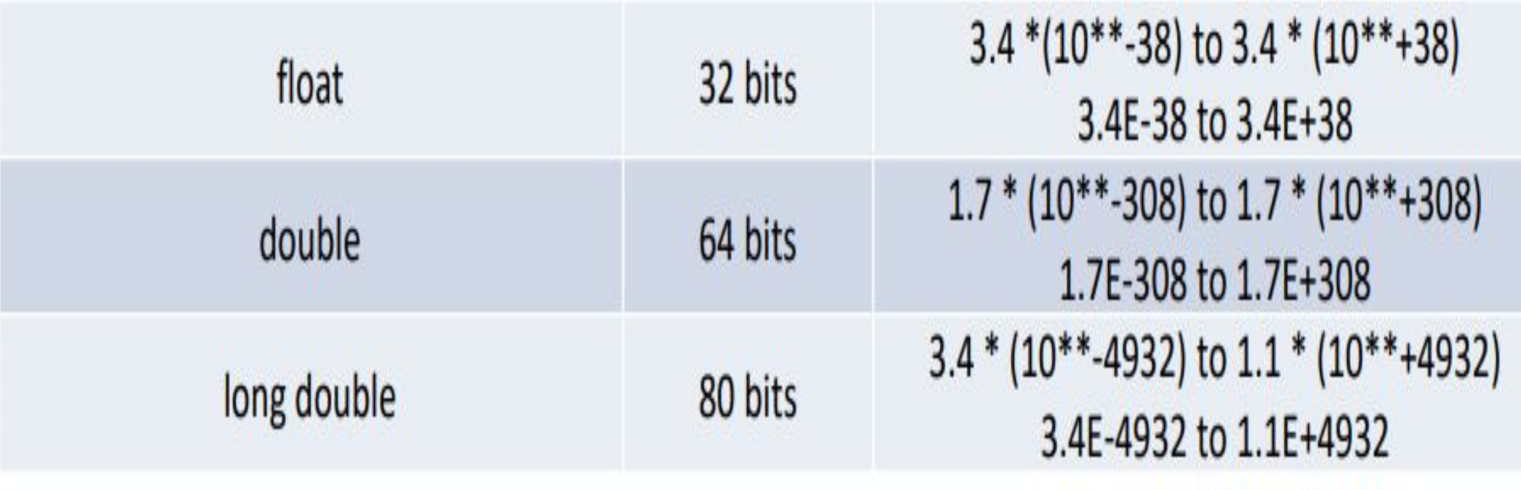
float and double:

Floating types are used to store real numbers.

float: A single-precision floating point value.

Double: A double-precision floating point value.

- TURBO C COMPILER

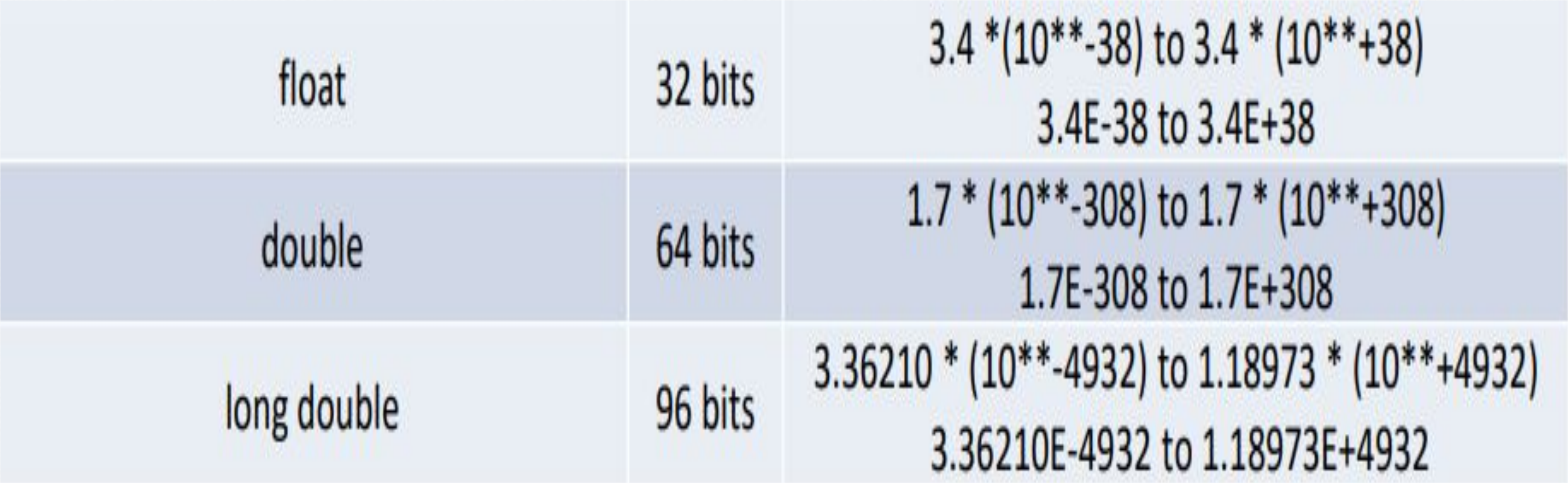


float	32 bits	3.4 *(10**-38) to 3.4 * (10**+38) 3.4E-38 to 3.4E+38
double	64 bits	1.7 * (10**-308) to 1.7 * (10**+308) 1.7E-308 to 1.7E+308
long double	80 bits	3.4 * (10**-4932) to 1.1 * (10**+4932) 3.4E-4932 to 1.1E+4932

PRIMARY (BUILT-IN) DATA TYPES:

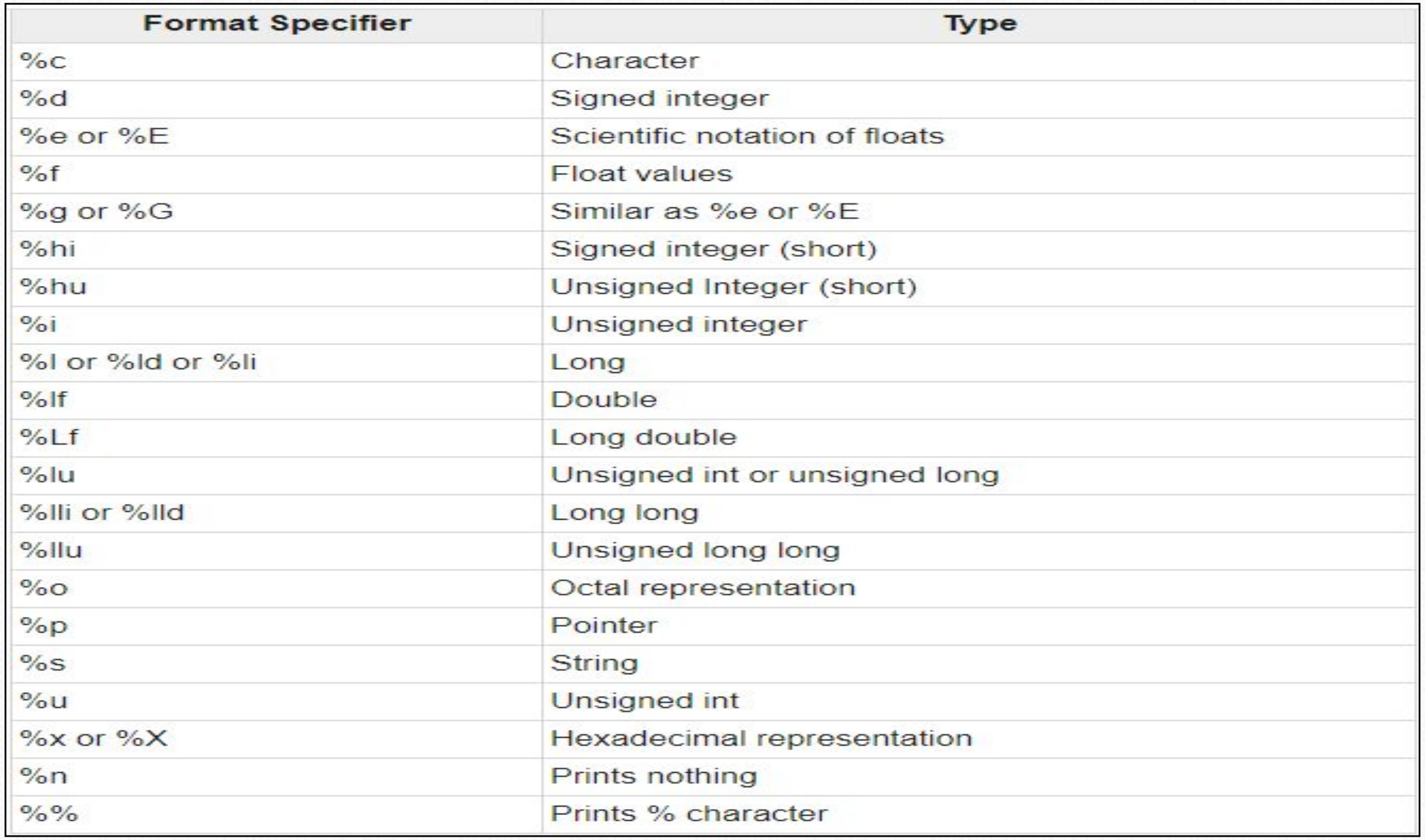
float and double:

- **GCC COMPILER**



float	32 bits	3.4 *(10**-38) to 3.4 * (10**+38) 3.4E-38 to 3.4E+38
double	64 bits	1.7 * (10**-308) to 1.7 * (10**+308) 1.7E-308 to 1.7E+308
long double	96 bits	3.36210 * (10**-4932) to 1.18973 * (10**+4932) 3.36210E-4932 to 1.18973E+4932

FORMAT SPECIFIERS



Format Specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	Unsigned integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

INPUT/OUTPUT FUNCTIONS

- In C Language a library of functions (predefined functions) is provided to perform I/O operations. The I/O library functions are listed the “header” file <stdio.h>.
- All input and output is performed with streams. A "stream" is a sequence of characters organized into lines. Each line consists of zero or more characters and ends with the "newline" character.
- Standard input stream is called "stdin" and is normally connected to the keyboard.
- Standard output stream is called "stdout" and is normally connected to the display screen.
- Standard error stream is called "stderr" and is also normally connected to the screen.

INPUT/OUTPUT FUNCTIONS

- **FORMATTED OUTPUT FUNCTION**: printf ()
- This function is a build-in function used for displaying formatted output to the screen.
- **Syntax:**
- **printf (“format specifier/quotes/escape sequences”, var1, var2, …var n);**
- The “format specifier” includes a listing of the data types of the variables to be output and, optionally, some text and control character(s).
- **Example:**
- int b=10;
- printf (“The value of b is: %d \n”,b);// %d –format specifier of integer

INPUT/OUTPUT FUNCTIONS

FORMATTED INPUT FUNCTION: scanf ()

This function is a build-in function used for getting formatted input from the keyboard.

Syntax:

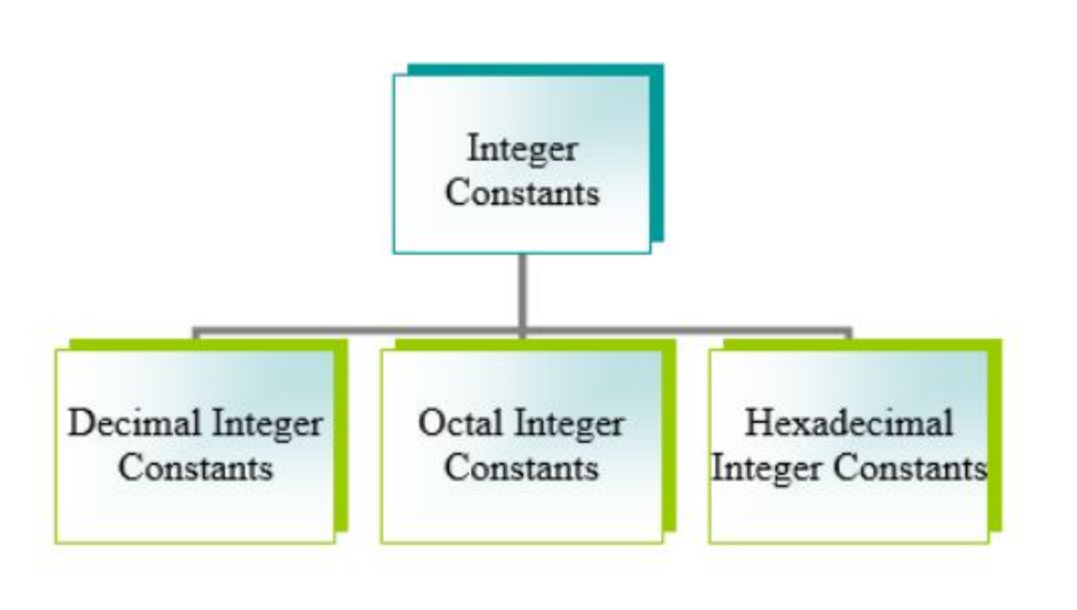
```
scanf ( “format specifier” , &var1, &var2, …) ;
```

The “format specifier” is a listing of the data types of the variables to be input and the & in front of each variable name tells the system WHERE(address) to store   the value that is given as input. It provides the address for the variable.

Example:

```
float a;  int b;
scanf (“%f%d”, &a, &b);
```

INTEGER CONSTANTS



INTEGER CONSTANTS

- C accepts integer constants in three numbering systems - decimal, octal and hexadecimal.
- **Decimal Integer Constant**-Decimal integer constants can consist any combinations of digits from 0 to 9, preceded by an optional + or – sign.(Eg: 98, +47, -87)
- **Octal Integer Constant**-Octal integer constants can consist of any combinations of digits from 0 to 7. C specifies that the octal integer must be preceded by a 0.(Eg:010, 07,0240)

INTEGER CONSTANTS

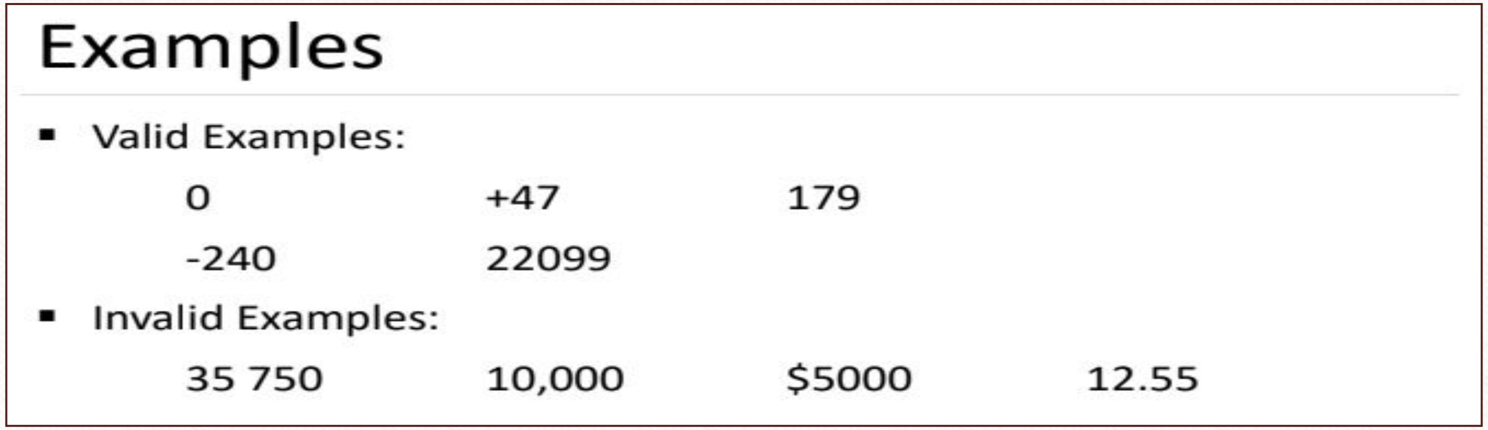
- **Hexadecimal Integer Constant**-Hexadecimal integer constants can consist of any combination of digits from 0 to 9 and letters from A to F (either uppercase or lowercase represents 10 to15)
- A hexadecimal integer constant must begin with either 0x or 0X(Eg:0x2,0X2A,0Xab)

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.
- The allowable range for integer constants depends upon the compiler.

Example:
Ex.:
426
+782
-8000
-7605



Examples

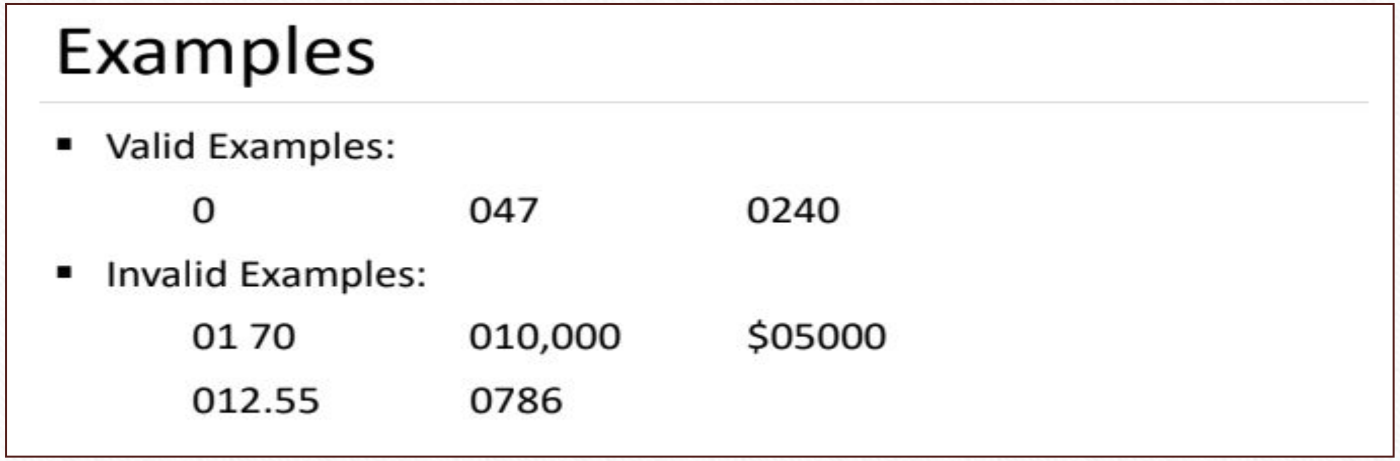
- Valid Examples:

0 +47 179

-240 22099

- Invalid Examples:

35 750 10,000 $5000 12.55



Examples

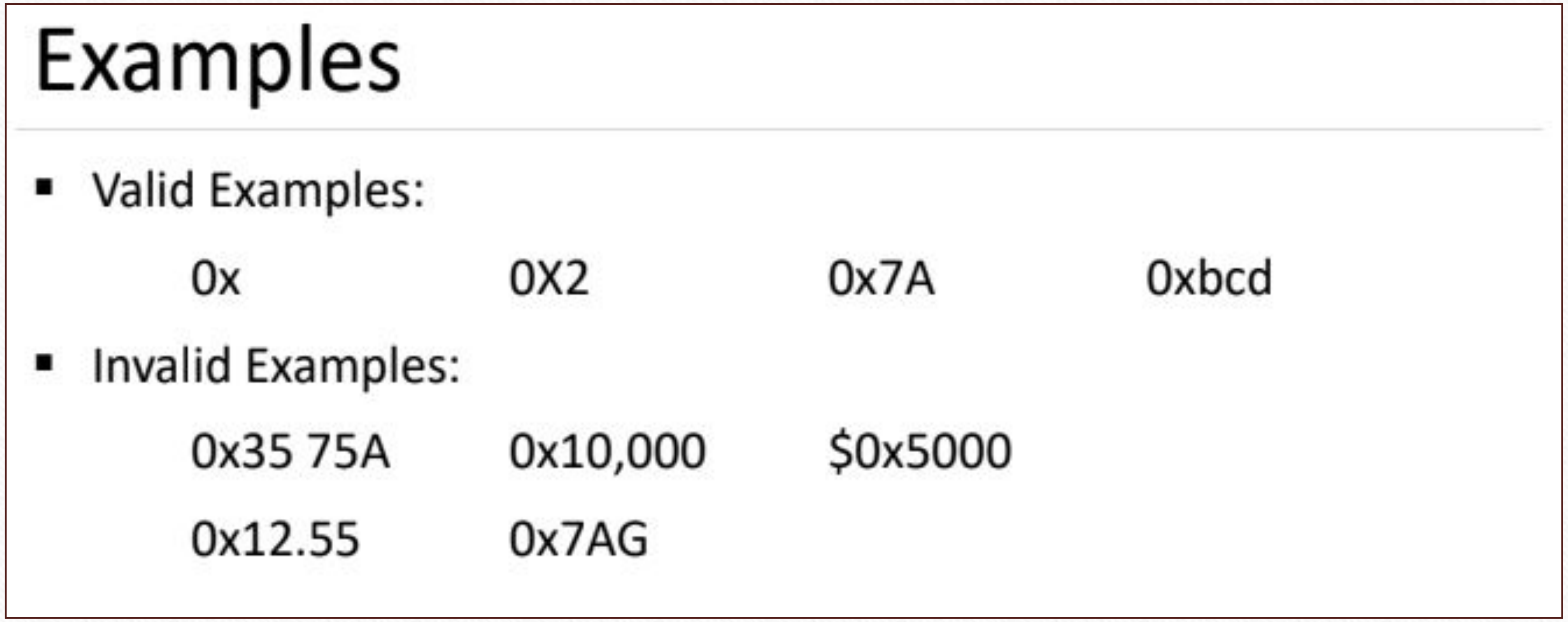
- Valid Examples:

0 047 0240

- Invalid Examples:

01 70 010,000 $05000

012.55 0786



Examples

- Valid Examples:

  0x   0X2   0x7A   0xbcd

- Invalid Examples:

  0x35 75A   0x10,000 $0x5000

  0x12.55   0x7AG

INTEGER CONSTANTS-Example

Example: 1

```
printf (“%d %d %d %d”, 45, 045, 0x45, 0X45);
```

What it will print?

The decimal equivalent of all the constants will be printed.

45 37 69 69

Example: 2

```
printf (“ %o %x”, 45, 45);
```

What it will print?

The octal and hexadecimal equivalent of 45 will be printed.

55 2D

INTEGER CONSTANTS-MCQ

An integer constant in C must have:

(1) At least one digit

(2) At-least one decimal point

(3) A comma along with digits

(4) Digits separated by commas

ANSWER:1

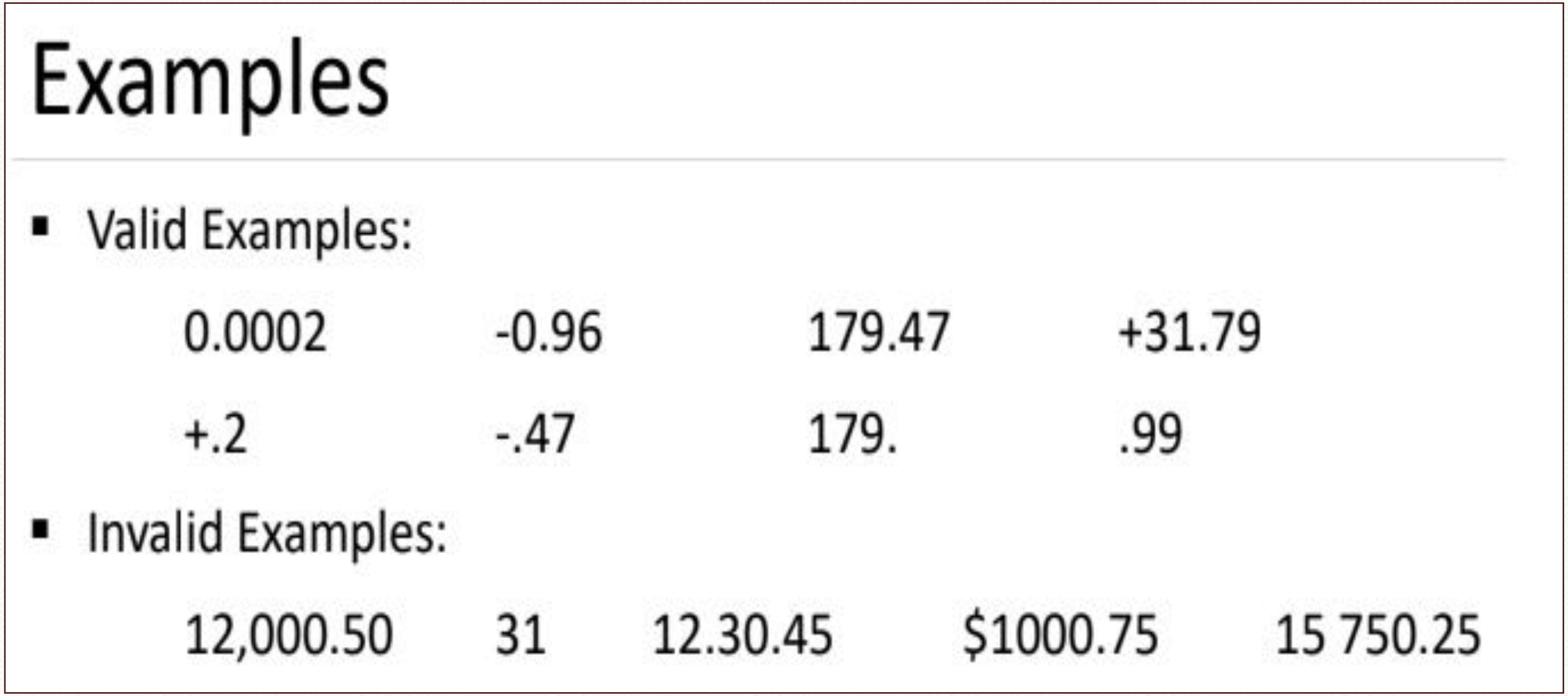
REAL CONSTANTS (FLOATING POINT CONSTANTS)

- Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form

Real constants expressed 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.

Ex.: +325.34
426.0
32.76
-48.5792



Examples

- Valid Examples:

  0.0002 -0.96 179.47 +31.79

  +.2 -.47 179. .99

- Invalid Examples:

  12,000.50 31 12.30.45 $1000.75 15 750.25

REAL CONSTANTS (FLOATING POINT CONSTANTS)

Example: 1

```
printf("%f %f",+456.67,-67.89);
```

What it will print?

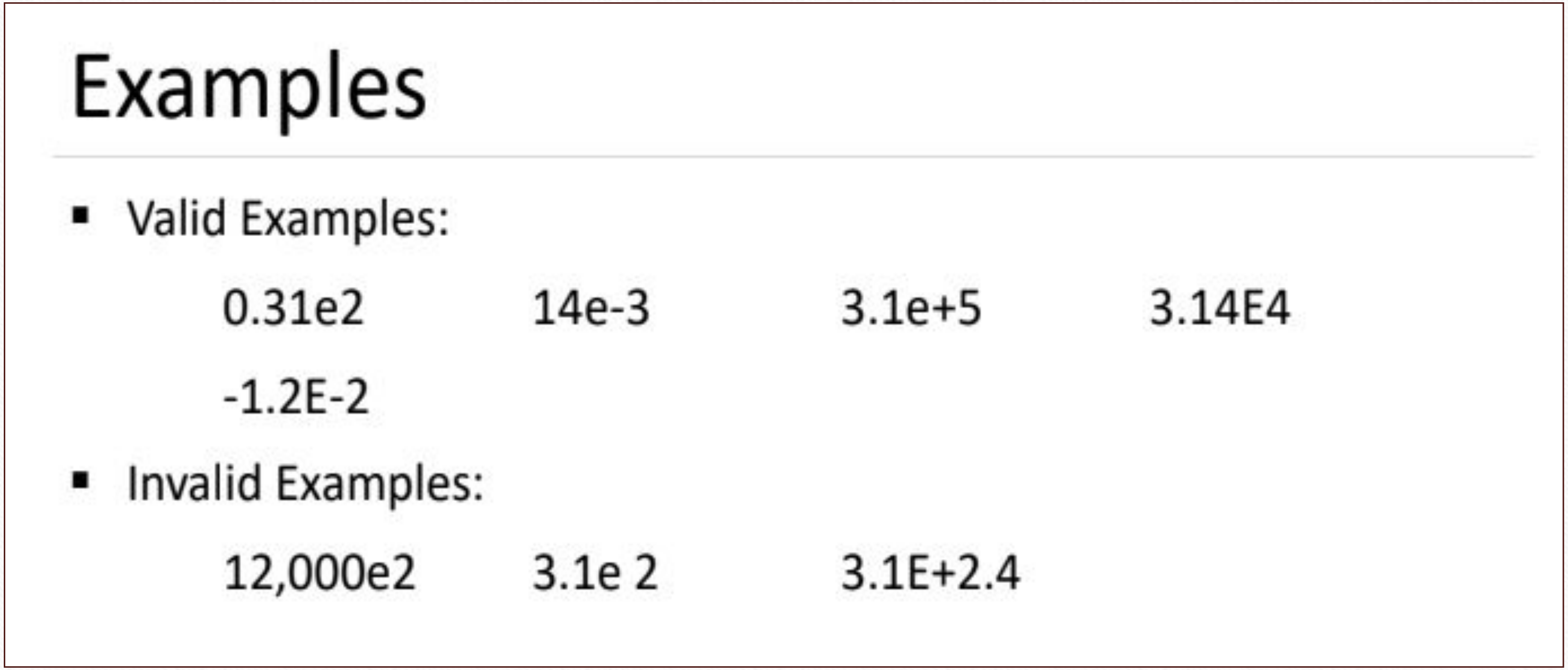
```
456.670000 -67.890000
```

REAL CONSTANTS (FLOATING POINT CONSTANTS)

Real constants expressed in exponential form

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa 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.
- Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex.:
+3.2e-5
4.1e8
-0.2e+3
-3.2e-5



Examples

- Valid Examples:

  0.31e2   14e-3   3.1e+5   3.14E4

  -1.2E-2

- Invalid Examples:

  12,000e2   3.1e 2   3.1E+2.4

REAL CONSTANTS (FLOATINT POINT CONSTANTS)

Example: 1

```
printf ("%f %f",4.1e8,-3.2e-5);
```

What it will print?

```
410000000.000000 -.000032
```

CHARACTER CONSTANTS

Rules for Constructing Character Constants

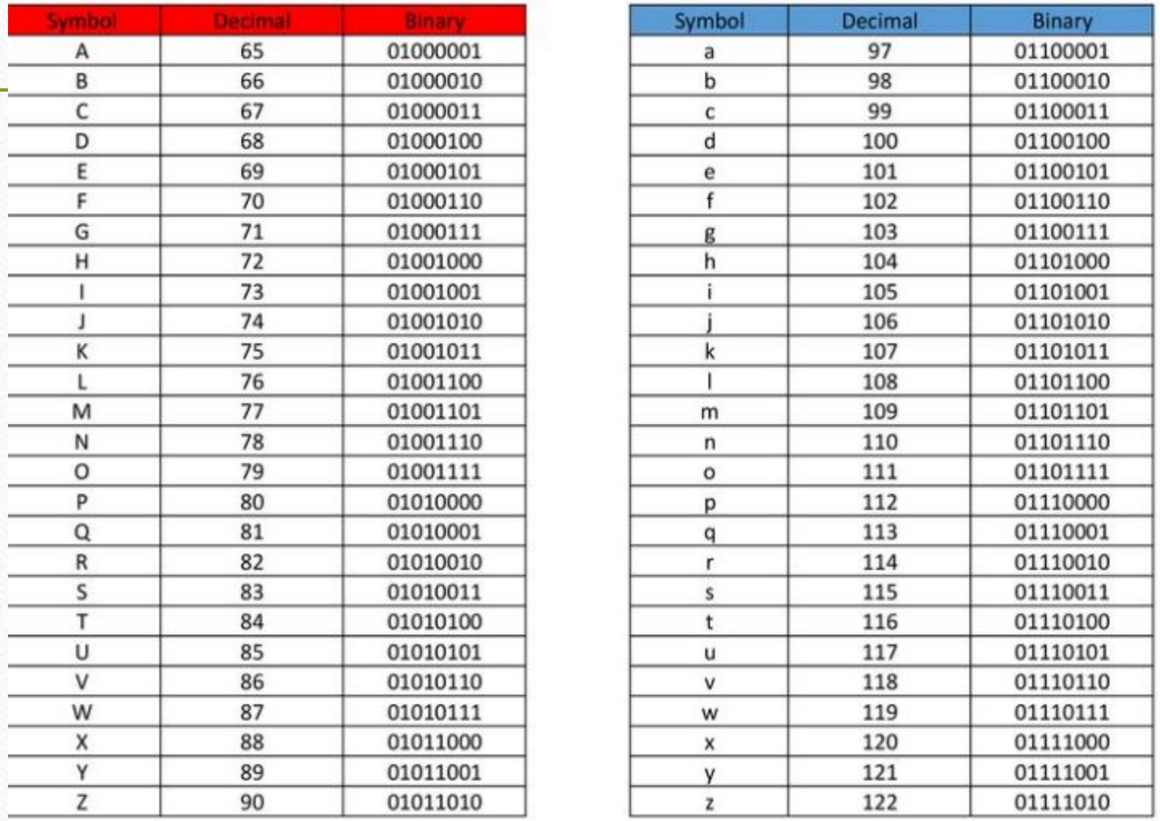
A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

The maximum length of a character constant can be 1 character.

Ex.: 'A'
'I'
'5'
'='

CHARACTER CONSTANTS-ASCII VALUES

In C language, all character constants are associated to an ASCII value(a number) ranging from 0 to 127.



Symbol	Decimal	Binary
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

Symbol	Decimal	Binary
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110
g	103	01100111
h	104	01101000
i	105	01101001
j	106	01101010
k	107	01101011
l	108	01101100
m	109	01101101
n	110	01101110
o	111	01101111
p	112	01110000
q	113	01110001
r	114	01110010
s	115	01110011
t	116	01110100
u	117	01110101
v	118	01110110
w	119	01110111
x	120	01111000
y	121	01111001
z	122	01111010

CHARACTER CONSTANTS-Example

Example: 1

```
    printf ("%c %c",'a','5');
    printf ("%d %d",'a','5');
```

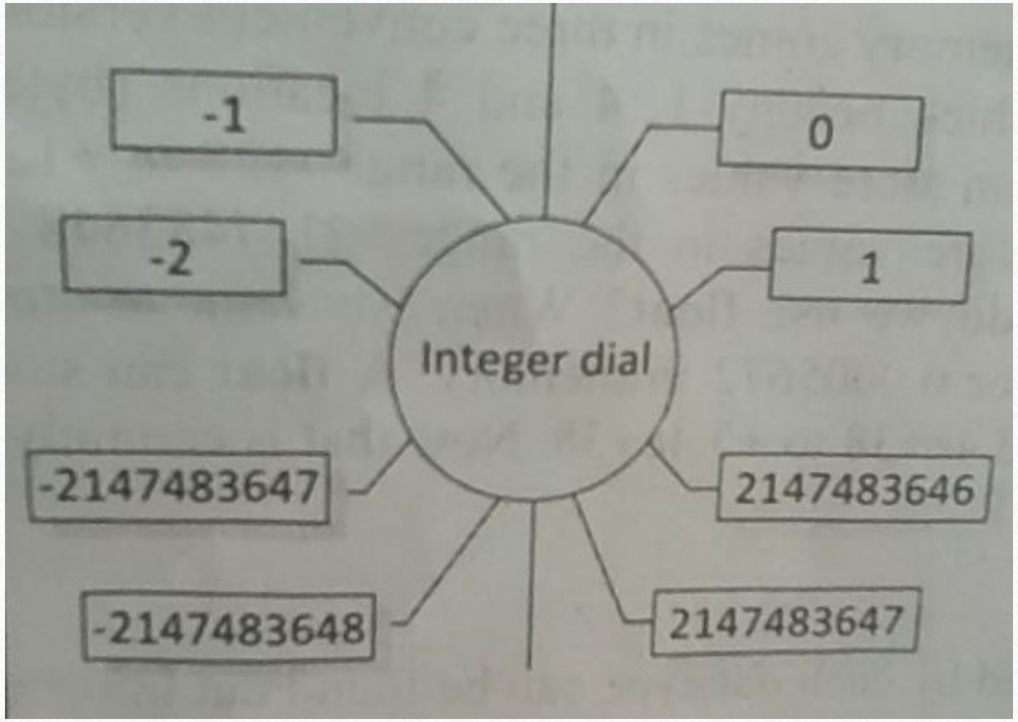
What it will print?

a 5 (%c will print the actual character as such)

97 53   (%d will print the ASCII value of the character)

INTEGER DIAL

- The range of the signed INT datatype is -2147483648 to 2147483647.The range values is logically viewed in the form of a dial.

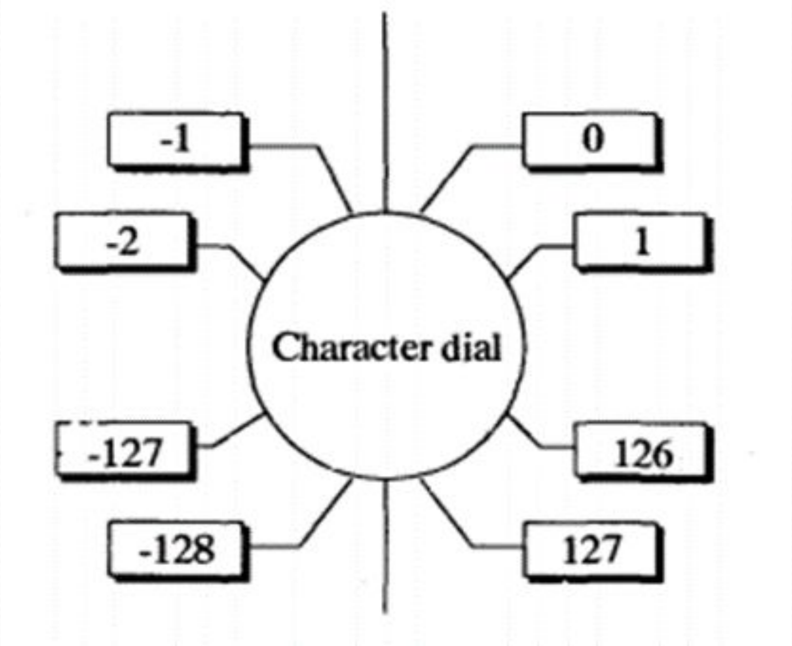


```
#include <stdio.h>
int main()
{
    int a=2147483648;
    printf("%d",a );
    return 0;
}
```

OUTPUT:-2147483648

CHARACTER DIAL

- The range of the signed char datatype is -128 to 127.The range values is logically viewed in the form of a dial.



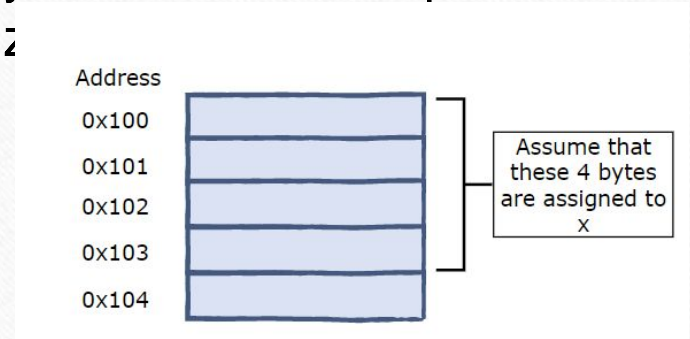
```
#include <stdio.h>
int main()
{
      char a = 128;
      printf("%d",a );
      return 0;
}
```

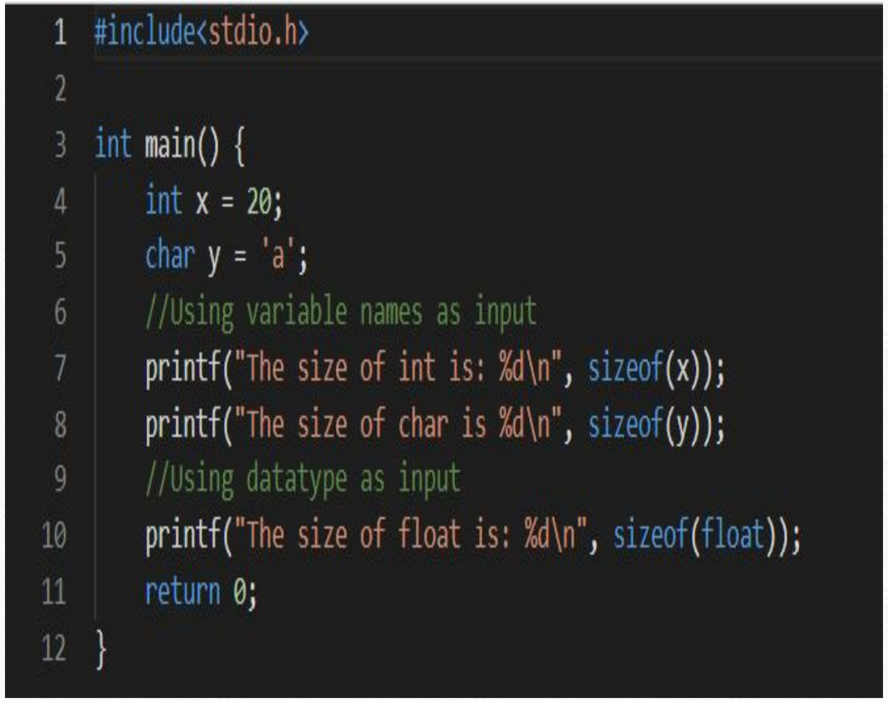
OUTPUT:-128

sizeof() built-in function

The **sizeof()function** in C is a built-in function that is used to calculate the size (in bytes)that a data type/variable/structure/expression occupies in the computer’s memory.

A computer’s memory is a collection of byte-addressable chunks. Suppose that a variable x is of type integer and takes four bytes of the computer’s memory, then siz





```c
#include<stdio.h>

int main() {
    int x = 20;
    char y = 'a';
    //Using variable names as input
    printf("The size of int is: %d\n", sizeof(x));
    printf("The size of char is %d\n", sizeof(y));
    //Using datatype as input
    printf("The size of float is: %d\n", sizeof(float));
    return 0;
}
```

POINT OUT THE ERRORS, IF ANY, IN THE FOLLOWING C STATEMENTS:

- 1. int = 314.562 * 150 ;
  - Error: variable name is missing/identifier is expected
- 2. name = ‘Ajay’ ;
  - Error: datatype is missing/ name variable is undeclared
- 3. varchar = ‘3’;
  - Error: datatype is missing/ name variable is undeclared

MULTIPLE CHOICE QUESTIONS

What is the output of this C code?

```
#include <stdio.h>
int main()
{
printf("Hello World! %d \n", x);
return 0;
}
```

A. Hello World! x;
B. Hello World! followed by a junk value
C. Compile time error
D. Hello World!

- **ANSWER:C(Since x is not declared as a variable)**

MULTIPLE CHOICE QUESTIONS

```
#include <stdio.h>
int main()
{
int main = 10;
printf("%d", main);
return 0;
}
```

A. It will cause a compilation error
B. It will cause a run-time error
C. It will run without any error and prints 10
D. It will experience infinite looping

- **ANSWER: C (It will run without any error and prints 10 as ‘main’ is not a keyword and can be used as a variable name)**

MULTIPLE CHOICE QUESTIONS

What will be the output of the following C code? [Assume it’s a 32-bit system]

```
#include <stdio.h>
int main()
{
int x = 70;
printf("%c", x);
return 0
}
```

A. 70

B. F

C. 70.0

D. Compilation error

ANSWER:B(Since ascii value of ‘F’ is 70)

MULTIPLE CHOICE QUESTIONS

The statement char ch = ‘Z’ would store what in ch

i. The character Z
ii. ASCII value of Z
iii. Z along with the single inverted commas
iv. Both (1) and (2)

ANSWER: B(for character variables the ASCII value will get stored in the memory)

WHAT WILL BE THE OUTPUT?

```
#include <stdio.h>
int main()
{
char a;
float b;
int c;
printf("%d %d %d\n", sizeof(char), sizeof(int), sizeof(float));
printf("%d %d %d\n", sizeof(a), sizeof(b), sizeof(c));
printf("%d %d %d ", sizeof('7'), sizeof(7), sizeof(7.0));
return 0;
}
```

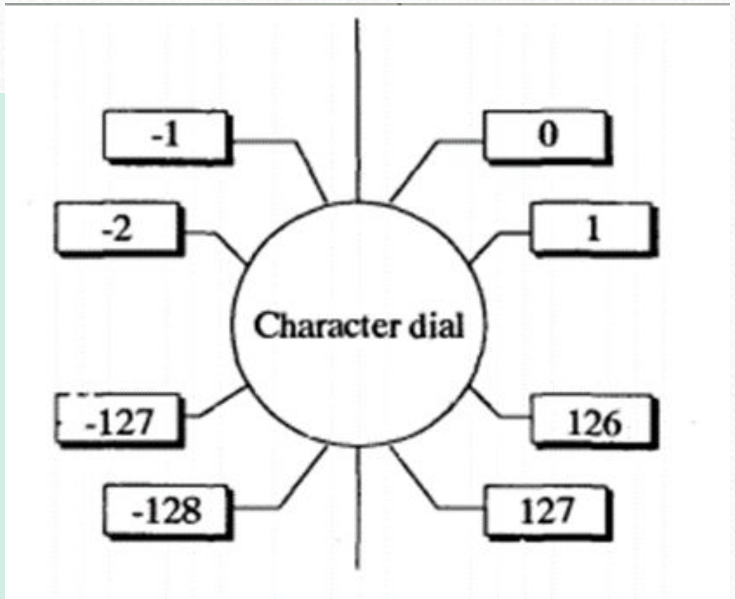
OUTPUT:

```
1 4 4
1 4 4
4 4 8
```

(Though ‘7’ is a character constant, its ASCII value is taken into account and the size of value gets calculated, for all characters it’s the same,if they are not stored into a specific variable)

WHAT WILL BE THE OUTPUT?

```
#include <stdio.h>
int main()
{
char ch = 291;
printf("%d %d %c", 2147483648, ch, ch);
return 0;
}
```



OUTPUT:

-2147483648 35 #(based on integer dial, based on char dial,35 is an ascii value,# is the corresponding char)

WHAT WILL BE THE OUTPUT?

```
#include <stdio.h>
int main()
{
unsigned char c = 257;
char a='1270';
char b='123a';
printf("%c %d\n",a,a);
printf("%c %d\n",b,b);
printf("%d %d\n", c, c);
return 0;
}
```

OUTPUT

```
0 48
a 97
1 1
```

(when we try to store more than 1 char into an char variable the last char will get assigned ignoring others)

WHAT WILL BE THE OUTPUT?

```
#include<stdio.h>
int main()
{
 int a = 123456789999;
 float b = 3.4;
 printf("a = %d b = %.2f\n", a, b);
 printf("%d %d", sizeof(a), sizeof(b));
 return 0;
}
```

OUTPUT

```
a = -1097261585 b = 3.40
4 4
```

(to get the output of a integer dial to be followed)

WHAT WILL BE THE OUTPUT?

```
#include<stdio.h>
int main()
{
 char str[10];
scanf(“%s”,str);
printf("%s",str);
}
```

Input:

```
Welcome to rec
```

OUTPUT:

```
Welcome
```

WHAT WILL BE THE OUTPUT?

```
#include<stdio.h>
int main()
{
 char str[10];
scanf(“%[^\n]s”,str);
printf("%s",str);
}
Input:
Welcome to rec
OUTPUT:
Welcome to rec
```

THANK
YOU