Fundamentals of C Language

Identifiers:

Identifiers: An identifiers is the name given to program elements such as variables, functions and arrays, etc. Identifiers are also the names of objects, which can take different values but only one value at a time.

Example: The following names are valid identifiers:

x	y12	sum_1	_temperature
names	area	tax_rate	TABLE

Keywords or Reserved Words:

C language used the following keywards which are not available to users to use them as variables/function names.

auto	**float**	**int**	**register**	**do**	**for**
return	**void**	**switch -------**			

Tokens:

Cprogram has punctuation marks, keywords and operators as the smallest individual units and are referred to as C tockens Following six types of tockens are used in C language.

	C tokens	Exs
1)	keywords	auto, float etc
2)	constants	-23, 4,16 etc
3)	identifiers	sum, average, etc
4)	string literals	“Total marks”, “sum=” etc
5)	operators	+, -, * etc
6)	separators	; , : “

Ex: What is identifier?

Solution: An identifier is a name used to identify a variable, function, symbolic constant and so on.

Example: sigma, calcu_avg

Ex: List the rules of naming an identifier in C.

Solution:

(i) An identifier must start with a letter or underscore followed by any number of digits and/or letters.

(ii) No reserved word or standard identifier should be used.

(iii) No special character (other than underscore) should be included in the identifier.

Ex: What is the difference between a keyword and an identifier?

Solution: *Keyword* is a special word that has a special meaning and purpose. Keywords are reserved and are a few. For example, goto, switch, else etc., are keywords in C.

Identifier is the user-defined name given to a part of a program viz., variable, object, function etc. Identifiers are not reserved. These are defined by the user and they can have letters, digits and a symbol underscore. They must begin with either a letter or underscore. For instance, _chk, chess, trial etc., are valid identifiers in C.

constant

The constant is a value, written into a program instruction that does not change during the execution of a program.

There are four basic types of constants in C. They are integer constants, floating-point constants, character constants and string constants.

Integer Constants: An *integer* constant is an integer–valued number. Thus, it consists of a sequence of digits. Integer constants can be written in three different systems: decimal (base 10), octal (base 8) and hexadecimal (base 16).

A *decimal* integer constant can consist of any combination of digits taken from the set 0 through 9. If the constant contains two or more digits, the first digit must be something other than 0.

Example: Several valid decimal constants are shown below:

0 1 745 5280 9999

An *octal* integer constant can consist of any combination of digits taken from the set 0 through 7. However the first digit must be 0, in order to identify the constant as an octal number.

Example: Several valid octal integer constants are shown below:

0 01 0743 077777

A *hexadecimal* integer constant must begin with either 0x or 0X. It can then be followed by any combination of digits taken from the sets 0 through 9 and **a** through **f** (either upper or lowercase). Note that the letters **a** through **f** (or A through F) represent the (decimal) quantities 10 through 15, respectively.

Example: Several valid hexadecimal integer constants are shown below:

0x 0X1 0X7FFF 0xabcd

Floating-Point Constants: A *floating-point* constant is a base-10 number that contains either a decimal point or an exponent (or both).

Example: Several valid floating-point constants are shown below:

0.	1.	0.2	827.602	50000.
0.000743	12.3	2E-8	0.006e-3	1.6667E+8

Character Constants: A *character* constant is a single character, enclosed in single quotation marks.

Example: Several character constants are shown below:

‘A’ ‘b’ ‘3’

String Constants: A string constant consists of any number of consecutive characters, enclosed in double quotation marks.

Example: Several string constants are shown below:

“green” “Washington, D.C. 20005”, “$19.95”

data-types:

There are various types of data that a variable may contain in a programming language. All such types are named as ***data type***. The basic data types are: int, char, float and double. The basic data types can be augmented by the use of the data type qualifiers short, long, signed and unsigned.

Variables:

A variable is a name that C language compiler associates with a storage location in the main memory of the computer. A variable is an identfifier that is used to represent some specified type of information within a designated portion of a program.Variable holds data that can be modified during program execution.

Every variable needs to be declared before it is used in a program. A variable declaration consists of a data type followed by one or more variable names, ending with a semicolon.

The general format of the variable declaration is:

```
data_type identifier 1,2, …, n
```

For example to declare j as an integer and x as a floating point variable, we should write:

```
int j;
float x;
```

As a shorthand, we can declare j and k in a single line as:

```
int j, k;
```

which is the same as the declaration of j and k as:

```
int j;
int k;
```

Q. How can the values of an expression be converted to a different data type?

Solution: The value of an expression can be converted to a different data type if desired. To do so, the expression must be preceded by the name of the desired data type, enclosed in parentheses, i.e.,

(data type) expression

This type of construction is known as a cast.

For example, suppose that i is an integer variable whose value is 7, and f is a floating-point variable whose value is 8.5. The expression

    (i + f) % 4

is invalid, because the first operand (i + f) is floating-point rather than integer. However, the expression

    ((int) (i + f)) % 4

forces the first operand to be an integer and is therefore valid, resulting in the integer remainder 3.

Q. What is a variable? How are the variables declared in C?

Solution:

Q. What is a data types?

data-types: There are various types of data that a variable may contain in a programming language. All such types are named as ***data type***. The basic data types are: int, char, float and double. The basic data types can be augmented by the use of the data type qualifiers short, long, signed and unsigned.

Q. Distinguish between data-types, variables, constants and identifiers giving suitable examples.

data-types: There are various types of data that a variable may contain in a programming language. All such types are named as ***data type***. The basic data types are: int, char, float and double. The basic data types can be augmented by the use of the data type qualifiers short, long, signed and unsigned.

variables: A variable is a name that C language compiler associates with a storage location in the main memory of the computer. Variable holds data that can be modified during program execution. After declare a variable in a program, we can assign it a value.

For example:

```
int a,b,c;
```

which state that a, b and c are integer variables.

constants: constant is a value, written into a program instruction, that does not change during the execution of a program.

There are three types of constant:

- String constant
- Numeric constant
- Character constant

For example: the following are the examples of valid string constants:
"The Total Marks="
"Rs 2000.00"
"34.557"

identifiers: identifiers are the names given to program elements such as variables, functions and arrays, etc. Identifiers are also the names of objects, which can take different values but only one value at a time.

Q. Name and describe the four basic data types in C with their memory requirements.

Solution: There are various types of data that a variable may contain in a programming language. All such types are named as Data Type.

C supports several different types of data, each of which may be represented differently within computer's memory. The basic data types are listed below. Typical memory requirements are also given.

Data Type	**Description**	**Typical Memory Requirements**	**Range**	
			Form	To
int	Integer quantity	2 bytes or one	-32,768	32,767

		word		
char	Single character	1 byte	-128	127
float	Floating-point number (i.e. a number containing a decimal point and /or an exponent	1 word (4 bytes)	3.4 X 10^{-38}	3.4 X 10^{38}
double	Double-precision floating-point number (i.e., more significant figures, and an exponent which may be larger in magnitude)	2 words (8 bytes)	1.7 X 10^{-308}	1.7 X 10^{308}
long	Whole numbers	1 word (4 bytes)	-2,147,438,648	2,147,438,647
long double	Fractional numbers	10 bytes	3.4 X 10^{-4932}	3.4 X 10^{4932}
unsigned int	Whole numbers	2 bytes	0	65,535
unsigned long	Whole numbers	4 bytes	0	4,294,967,295

Ex: What is string constant? Give an example.

Solution: String of characters enclosed in single or double quote is called string constant.
Example:
‘m’ - single character string constant
“COMPUTER” - string of characters string constant.

Ex: What is the difference between C character and C string?

Solution: A C character can hold a single character which is enclosed in single quote(‘ ’).

Example:

```
char ch = ‘m’;
```

A C character string can hold a string of characters which is enclosed in double quotes (“ ”).

Example:

```
char st[20] = “COMPUTER WORLD”;
```

Q. What is a character constant? How do these character constants differ from an integer constant?

Solution:

Character constant is either a single alphabet or a single digit, or a single special symbol enclosed within a pair of single quotation mark. Here are some examples:

‘A’
‘a’
‘*’
‘?’

Integer constants are whole numbers (0, 1, -1, 2, -2, 3, -3 etc.) and should not have any fractional or decimal part. The following are examples of valid integer constants:

3
-8000
162

Ex: What is the difference between ‘a’ and “a” in C?

Solution: ‘a’ is a single character constant but “a” is a string of characters constant. The computer assigns a null character “\0” to the end of this constant.

Operators and Expressions:

An expression consists of variables and constants separated by operators. C language uses many types of operators as listed below:

i) Arithmetic operators
ii) Relational operators
iii) Logical operators
iv) Increment and decrement operators
v) Assignment operators
vi) Conditional operators
vii) Bitwise operators

Arithmetic operators:

Arithematic operators are used to perform arithematic operations. Following are the arithematic operations used in C language:
+, −, *X* , /, % (modulus operator).

Ex: What is hierarchy of operation (operator precedence)? Mention the operator precedence for arithmetic operators.

Solution: The computer scans an expression from left to right many times and performs only one operation at a time. The order in which various expressions are performed is called hierarchy of operation or operator precedence. The following are the operator precedence for arithmetic operators.

1. * / %
2. + -

Q. What is a modulus operator? What are the restrictions of a modulus operator?

Solution: A Modulus operator gives the remainder value. The result of x%y is obtained by (x-(x/y)*y). This operator is applied only to integral operands and cannot be applied to float or double.

Ex: What is a Unary operators?

When only one operand is used with + or – operator, the operation is called unary plus or unary minus, which does not actually refer to addition or subtraction.
For eg. +25 (it is simply number 25), -7 (it is negative number 7)

Ex. What is the Relational operators?

Relational operators are used to compare the values of operands (expression) to produce a logical value. A logical value is either true or false. Following are the relational operators in C.

Operator	**Meaning**
<	Less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

Note that in C language the logical value true is represented by integer 1 and false by 0.

Q.3: What are the logical operators available in C?

Solution: The logical operators available in C are:

&&	-	logical AND
||	-	logical OR
!	-	logical NOT (negation)

Q. Write down the logical operators available in C and show how they work giving examples.

Solution: A logical operator is used to compare or evaluate logical and relational expressions. There are three logical operators in C language. They are:

Operator	**Meaning**
&&	Logical AND
| |	Logical OR
!	Logical NOT

Logical AND: For example, consider the following expression:

a > b && x = = 10

The expression on the left is a > b and that on the right is x = = 10. The whole expression evaluates to true (1) only if both expressions are true (if **a** is greater than **b** and the value of x is equal to 10).

Logical OR: Consider the following example involving the | | operator:

a < m | | a < n

The expression is ***true*** if one of them is ***true***, or if both of them are ***true***. That is, if the value of **a** is less thyan that of **m** or **n**. Needless to say, it evaluates to ***true*** in case **a** is less than both **m** and **n**.

Logical NOT: The ! (NOT) operator takes single expression and evaluates to ***true*** (1) if the expression is false (*zero*), and evaluates to ***false*** (0) if the expression is ***true***. In other words, it just reverses the value of the expression. For example, consider:

!(x >= y)

The expression after the ! operator, in this case is x >= y. The not expression evaluates to true only if the value of x is neither greater than nor equal to y (i.e., only if x is less than y). Therefore, the whole expression including the ! operator is the same as:

x < y

So what is the use of this operator? The ! operator is convenient to use when we want to test whether the value of a variable is *zero*. For example,

```
if( !i )
        printf("The value of i is zero.\n");
```

The expression inside the braces of the if statement is:

!i

This is true only if the value of i is *zero.*

Increment and Decrement operators.

Ex: Discuss the increment operator in C.

Solution: The increment operator (++) is used to increase the value of a variable by 1. It can be written as post-fix or prefix to a variable, e,g, a++ or ++a.

Ex: Discuss the decrement operator in C.

Solution: The decrement operator (--) is used to reduce the value of a variable by 1. It can be written as post-fix or prefix to the variable, e.g. a-- or --a.

Q. Describe two ways to utilize the increment and decrement operators. How do the two methods differ?

Solution: The increment and decrement operators (there are two, a **pre-** and **a post-** operator for each operation) are unary operators.
The **post-increment** operator ++ is written to the right of its operand:

```
n++;
```

The effect of the statement above is to add one to the current value of n; its action is equivalent to:

```
n = n+1;
```

Besides being more concise, the post-increment operation on some computers may be executed faster than this latter form of assignment.

The post-decrement operation is represented by -- and is written:

n--;

it decrements **n** by unity, producing a result identical to that of:

n = n-1;

The pre-increment and pre-decrement operators are placed to the left of the operand on which they are to act, and the result of their action is, as before, to increment or decrement the operand:

```
--n;    /* assign n-1 to n */
++n;   /*assigns n+1 to n*/
```

Insofar as their operand **n** is concerned, the **pre-** and **post-** increment or decrement operators are equivalent. Each adds or subtracts one to its operand.

Where the pre- and post- operators differ is in the value used for the operand **n** when it is embedded inside expressions.

If it’s a “pre” operator the value of the operand is incremented (or decremented) before it is fetched for the computation. The altered value is used for the computation of the expression in which it occurs.

To clarify, suppose that an **int** variable **a** has the value 5. Consider the assignment:

b = ++a;

Pre-incrementation implies:

```
Step 1: increment a;                  /* a becomes 6 */
Step 2: assign this value to b;       /* b becomes 6 */
```

If it’s a “post” operator the value of the operand is altered after it is fetched for the computation. The unaltered value is used in the computation of the expression in which it occurs.

Suppose again that **a** has the value 5 and consider the assignment:

b = a++;

Post-incrementation implies:

```
Step 1: assign the unincremented a to b;     /* b becomes 5 */
Step 2: increment a;                         /* a becomes 6 */
Result: a is 6, b is 5.
```

The placement of the operator, before or after the operand, directly affects the value of the operand that is used in the computation. When the operator is positioned before the operand, the value of the operand is altered before it is used. When the operator is placed after the operand, the value of the operand is changed after it is used.

Q. How does x++ differ from ++x?

Solution: x++ is a post-fix increment operator. For example, when m =15; k = m++; it assigns the value 15 to k and thereafter increase the value of m to 16.
++x is a prefix increment operator. For example, when m =15; k = ++m; it increases the value m to 16 and thereafter assigns 16 to k.

Ex: Write four different C statements each adding 1 to integer variable x.

Solution: The four different C statements each adding 1 to integer variable x are:

```
x = x + 1;
x += 1;
x++;
++x;
```

Assignment operators:

Assignment operators are used to perform arithematic operations while assigning a value to a variable. . Following are the arithematic assignment operators used in C.

Operator	Ex	Equivalance Exp(m=15)	Result
+=	m+=10	m=m+10	25
-=	m-=10	m=m-10	5
*=	m*=10	m=m*10	150
/=	m/=10	m=m/10	1
%=	m%=10	m=m%10	5

Conditional operator(?:) or Ternary operator:

Conditional operator is used to check a condition and select a value depending on the value of the condition. Normally the selected value will be assigned to a variable which has the following form.

Variable=(condition) ? value 1 : value 2;

When this operator is executed by the computer, the value of the condition is evaluated. If it is trued then the value 1 is assigned to the variable, otherwise value 2 is assigned to the variable.

For eg. Big= (a > b) ? a: b;

Bitwise operators.

Bitwise operators are used to perform operations at binary digit level. The bit operators used in C are:

<<	left shift operator
>>	right shift operator
~	bitwise inversion (one’s complement)
&	bitwise logical and
|	bitwise logical or
^	bitwise exclusive or.

Ex: What are the bit operators used in C.

Additional operators:

There are two operators can be used in a C program.

(i) Sizeof operator
(ii) Comma operator

Sizeof operator:

The sizeof( ) operator is used to find the number of bytes occupied by a data item in the memory. For eg.,

sizeof(float); returns the value 4.

Int m, x[50]; declares an array with 50 elements and variable m.

Comma (,) operator:

The comma operator is used to link related expressions to make the program more compact. Related expressions are evaluated from left to right and the value of the right most expression is returned as the value of the expression. For eg.   Temp = x, x = y, y = temp;

Casts or explicit conversion:

A variable declared in a specific data type can be converted to another data type. This process is called casting. It has the following form:

(type) expression

For eg.,

```
int m=5;
Float y;
y = (float) m/2;
```

This expression, the **int** variable m is converted into **float** to get the result 2.5 , otherwise the result will be 2.

Ex: What do you mean by casting? Give an example.

Q. Differentiate the following:

Unary operator and binary operator

Solution:

Unary operator:

1. Operators that act on one operand are referred to as unary operators.
2. Associativity of unary operators are from right to left.
3. Examples of unary operators are ++ and – called increment and decrement operators, respectively, -(unary minus) one’s complement (~), Negation (!), Address of (&), value at address (*), size in bytes (size of) etc.

Binary operator:

1. Operators that act upon two operands are referred to as binary operators.
2. Associativity of binary operators are from left to right.
3. Examples of binary operators are addition (+), subtraction (-), division (/), (*) multiplication, modulus (%), etc.

Ex: How does the type float differ from double in C language?

Solution: float data type refers real number in single precision and has 6 decimal digits. It takes 4 bytes in memory to refer values ranging from 3.4e-38 to 3.4e+38.
double data type also refers to real number but in double precision and has 12 decimal digits. It takes 8 bytes of memory to refer values ranging from 1.7e-308 to 1.7e+308.

Q. How does the type ‘float’ differ from ‘double’ in C language?

Solution:

To store a fraction, we need a floating point type of data variable, called data type float. It occupies 4 bytes in memory. It can hold numbers in the range from –3.4e38 to +3.4e38 with six and seven digits of precision.

A double data type can store a floating point value with a greater accuracy than a float type. Double data type occupies 8 bytes in memory and can hold numbers in the range from –1.7e308 to +1.7e308 with 15 digits of precision.

A variable of type **double** can be declared as:

```
double a, population;
```

A variable of type **float** can be declared as:

```
float a, population;
```

Ex: Explain with an example the usage of shorthand assignment operator.

Solution: Assignment operators are used to perform arithmetic operation while assigning a value to a variable. The assignment operators are more concise and more efficient.
For example, m+=10; represents the shorthand of the expression m = m + 10;

Q. List the bitwise operators and describe their working.

Solution: A bitwise operator operates on each bit of data. These operators are used for testing, complementing or shifting bits to the right or left. Usually bitwise operators are not useful in cases of float and double variables. A list of bitwise operators are given below:

Operator	**Meaning**
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	shift left
>>	shift right
~	bitwise complement

To experiment with these operators, assume that a, b and c are integers declared by the statement

```
int a = 13, b = 7, c;
```

Also, for convenience, let us assume that an integer occupies 16 bits (2 bytes).

The binary representation of a is 0000 0000 0000 1101
The binary representation of b is 0000 0000 0000 0111.

Bitwise AND: The statement,

```
c = a & b;
```

makes use of the bitwise AND operator. After this statement is executed, each bit in c will be 1 only if the corresponding bits in both a and b are 1. For example, the right most bit of both integers is 1, and hence the rightmost bit in c is 1. The next bit is 0 in a and 1 in b. Hence the second bit (from the right) in c is 0. Applying this reasoning to all the bits in each integer, the value of c after the above statement is executed, will be 0000 0000 0000 0101 which, in decimal is 5 and is illustrated below:

Bitwise AND operator: a & b

```
    a  0000 0000 0000 1101
    b  0000 0000 0000 0111
a & b  0000 0000 0000 0101
```

Bitwise OR: The statement,

c = a | b;

makes use of the bitwise OR operator. After this statement is executed, a bit in C will be 1 whenever at least one of the corresponding bits in a or b is 1. In the above example, the value of c will be 0000 0000 0000 1111, i.e., decimal 15 and is illustrated below:

Bitwise OR operator: a | b

```
    a  0000 0000 0000 1101
    b  0000 0000 0000 0111
a | b  0000 0000 0000 1111
```

Bitwise XOR: The statement,

c = a ^ b;

makes use of the bitwise XOR operator. After this statement is executed, a bit in c will be 1 whenever the corresponding bits in a and b differ. So in the above example, the value of c will be 0000 0000 0000 1010 which, in decimal is 10 and is illustrated below:

Bitwise EX-OR operator: a ^ b

```
    a  0000 0000 0000 1101
    b  0000 0000 0000 0111
a ^ b  0000 0000 0000 1010
```

Left shift operator: The left bit-shift operator, “<<” is a binary operator. For example consider the statement,

c = a << 3;

The value in the integer a is shifted to the left by three bit positions. The result is assigned to the integer c. Since the value of a is 0000 0000 0000

1101 the value of c after the execution of the above statement is 0000 0000 0110 1000 (104 in decimal) and is illustrated below:

Left-Shift <<

$$\begin{tikzcd}\text{drop off} & \boxed{\text{0000 0000 0000 1101}} \arrow[l] & \text{insert 0's} \arrow[l]\end{tikzcd}$$

after Left-bit shift by 3 places i.e., a << 3

$\boxed{\text{0000 0000 0110 1000}}$

The three left-most bits drop off due to the left shift (i.e., they are not present in the result). Three zeros are inserted in the right. The effect of shifting a variable to the left by one bit position is to multiply it by 2. In the above example, a is shifted left by 3 positions, which is equivalent to multiplying a by 2*2*2, i.e., 2^3. Since the initial value of a is 13, the value assigned to c is 13 * 8 = 104.

While multiplying a number by a power of 2, considerable saving in execution time can be achieved by using the left bit-shift operator instead of the multiplication operator, since shifting is faster than multiplication.

Right shift operator: The right bit-shift operator is also a binary operator. For Example, consider

c = a >> 2;

The value of a is shifted to the right by 2 positions. Since the value of a is 0000 0000 0000 1101, the value of c after the execution of the above statement is 0000 0000 0000 0011 (3 in decimal) and is illustrated below:

Right-Shift >>

$$\begin{tikzcd}\text{Insert 0's} \arrow[r] & \boxed{\text{0000 0000 0000 1101}} \arrow[r] & \text{drop off}\end{tikzcd}$$

after right shift by 2 places i.e., a >> 2

$\boxed{\text{0000 0000 0000 0011}}$

The 2 right-most bits drop off (are not present in the result), and zeros are inserted in the left. The effect of shifting a variable to the right by one bit position is to perform integer division by 2 (i.e., divide by 2 and truncate the result). Hence, shifting to the right by 2 bit positions has the effect of integer division by 2*2 = 4. Since the initial value of a is 13, shifting to the right by 2 bit positions yields the value 3 (the result of dividing 13 by 4 and truncating the result).

Bitwise complement operator: The one's complement operator (~) is a unary operator that causes the bits of its operand to be inverted (i.e., reversed) so that 1s become 0s and 0s become 1s. This operator always precedes it operand. The operand must be an integer-type quantity. Generally, the operand will be an unsigned octal or an unsigned hexadecimal quantity.
Consider the hexadecimal number 0x7ff. The corresponding bit pattern, expressed in terms of a 16-bit word, is 0000 0111 1111 1111. The one's complement of this bit pattern is 1111 1000 0000 0000, which corresponds to the hexadecimal number f800. Thus, we see that the value of the expression ~0x7ff is 0xf800.

Q.5: Discuss the difference between assignment and equality.

Solution: Assignment means the process of assigning the value of a variable/expression from the right side of assignment operator to the left side variable. The assignment operator in C is =.
Equality means the process of checking the value of the left side variable/expression with the right side variable. The equality operator in C is = =.

Q. A C program contains the following declarations:

```
int i,j;
```

```
float x;
char c;
```

Determine the data type of each of the following expressions.

i) i+c
ii) x+c
iii) i+x
iv) j+x

Solution: The data type for the expression will be:

i) Integer
ii) Float
iii) Float
iv) Float

Q. If a is an integer variable, then what value a=5/2 will return?

Solution: The value 2 will return.

Q. How do you write x^y in C language? **1**
Solution: We write x^y as

```
pow(x,y);
```

Q. How can you determine the maximum value that a numeric variable can hold?

Solution: For integral types, on a machine that uses two's complement arithmetic (which is just about any machine you're likely to use), a signed type can hold numbers from 2(number of bits 1) to +2(number of bits 1) 1. An unsigned type can hold values from 0 to +2(number of bits) 1. For instance, a 16-bit signed integer can hold numbers from (32768) to +(32767).

Q.9: What will be the output for 9%4

Solution: The output will be 1.

Q. What will be the effect of executing the following statement?

```
printf("%d\n", 'A');
```

Solution: The ASCII value 65 for character 'A' will be printed.

&&&&&&&&&&