1. Explain operators available in C.

- An operator is a symbol that tells to perform certain mathematical or logical operation.
- C programming language provides several operators to perform different kind to operations.
- **Unary** operators operate on single operand, e.g. -8 (Minus 8)
- **Binary** operators operate on two operands, e.g. a + b (Addition of a and b) where + is operator and a & b are operands.
- C has rich set of operators as below,

Arithmetic Operators

- Arithmetic operators are used for mathematical calculation.
- Suppose a=10, b=5 then

Operator	Description	Answer
+ (Addition)	c = a + b;	15
– (Subtraction)	c = a – b;	5
* (Multiplication)	c = a * b.	50
/ (Division)	c = a / b;	2
% (Modulo)	c = a % b	0

Relational Operators

- Relational operators are used to compare two numbers and taking decisions based on their relation. Relational expressions are used in decision statements such as if, for, while, etc…
- Suppose a=10, b=5

Operator		Description	Answer
<	(less than)	a < b returns true if a is smaller than b.	FALSE
<=	(less than or equal to)	a <= b returns true if a is smaller or equal to b.	FALSE
>	(greater than)	a > b returns true if a is larger than b.	TRUE
>=	(greater than or equal to)	a >= b returns true if a is larger or equal to b.	TRUE
==	(is equal to)	a == b returns true if a and b have same value.	FALSE
!=	(is not equal to)	a != b returns true if a and b have different value.	TRUE

Logical Operators

- Logical operators are used to check more than one conditions at a time and make decisions
- Suppose a=10,b=5 then,

Operator	Description	Example
&& (Logical AND)	If both conditions are true then true otherwise false	• if(a>7 && b<4) returns TRUE • if(a>7 && b>4) return FALSE
|| (Logical	If either condition is true	• if(a>7 || b<4) returns TRUE

OR)	then true otherwise false	• if(a>7 || b>4) return TRUE
! (Logical NOT)	It converts false in to true and true in to false	• if(a!=7) returns TRUE • if(a==10) returns TRUE • if(a!=10) returns FALSE

Assignment Operators

- Assignment operators are used to assign the result of an expression to a variable.
- C supports shorthand assignment operators which simplify operation with assignment

Operator	Description	Example
=	Assigns value of right side to left side	`a=b`
+=	a += 1 is same as a = a + 1	`a+=5` → `a=a+5` → 15
–=	a -= 1 is same as a = a – 1	`a-=5` → `a=a-5` → 5
*=	a *= 1 is same as a = a * 1	`a*=5` → `a=a*5` → 50
/=	a /= 1 is same as a = a / 1	`a/=5` → `a=a/5` → 2
%=	a %= 1 is same as a = a % 1	`a%=5` → `a=a%5` → 0

Increment and Decrement Operators

- These are special operators in C which are generally not found in other languages.
- They simplify most commonly used operations of + by 1 or – by 1.
- ++ Increments value by 1.
- **a++** is **postfix**, the expression is evaluated first and then the value is incremented. Ex. a=10; b=a++; after this statement, a= 11, b = 10.
- **++a** is **prefix**, the value is incremented first and then the expression is evaluated. Ex. a=10; b=++a; after this statement, a= 11, b = 11.
- – – Decrements value by 1.
- **a--** is **postfix**, the expression is evaluated first and then the value is decremented. Ex. a=10; b=a--; after this statement, a= 9, b = 10.
- **--a** is **prefix**, the value is decremented first and then the expression is evaluated. Ex. a=10; b=--a; after this statement, a= 9, b = 9.

Conditional Operator

- It is special short form of if … else … We can write simple if condition in single line with the help of conditional operator.
- C supports special conditional operator (? :) to evaluate conditional expression in single line
- It is also known as a ternary operator because it is the only operator in C which requires three operands.
- First of all expr1 is evaluated, if it is true then the expression expr2 is evaluated otherwise expr3 is evaluated. Only one of expr2 or expr3 is evaluated
- `exp1? exp2 : exp3` if exp1 is true then execute exp2 otherwise exp3
- `Ex: x = a > b? a : b;`

Bitwise Operators

- Bitwise operators are used to perform operation on bit.
- Bitwise operators can be applied only on integer data type.
- Suppose a = 11 and b = 5 then, in bit-wise operators it will convert the number in to binary then it will perform the task,
- a = 11 = 1011
- b = 5 = 0101

Operator	Description	Output
&	bitwise AND (Will Perform AND Gate)	a & b = 0001 = 1
|	bitwise OR (Will Perform OR Gate)	a | b = 1111 = 15
^	bitwise exclusive OR (Will Perform XOR Gate)	a ^b = 1110 = 14
<<	shift left (Shift the bits to left side)	a<<1 = 10110 = 22
>>	shift right (Shift the bits to right side)	a>>1 = 0101 = 5

Special Operators

Operator	Description
& (Address operator)	It is used to determine address of the variable.
* (Pointer operator)	It is used to declare pointer variable and to get value from it.
, (Comma operator)	It is used to link the related expressions together.
sizeof()	It returns how many bytes are required to store the variable in memory.
. (Member selection operator)	It is used in structure to access a member.
-> (Member selection operator)	It is used in structure to access a member via pointer.

2. Explain operator precedence and associativity.

- When two operators share an operand the operator with the higher precedence goes first.
- Precedence of an operator is its priority in an expression for evaluation.
- We evaluate 5 + 3 * 2 as 5 + (3 * 2), giving 11 and not as (5 + 3) * 2 , giving 16. Because we know that * (multiplication) and / (division) has higher priority compared to + (addition) and – (subtraction).
- The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.
- Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence.
- Following table provides a complete list of operator, their precedence level, and their rule of association. Rank 1 indicates highest precedence level and 15 is the lowest

Precedence	Associativity	Operator	Description
1	Left to Right	() []	Function Call Array Element Reference
2	**Right to Left**	+, - ++, -- ! ~ * & (type)	Unary Plus, Unary Minus Increment, Decrement Logical Negation Ones Complement Pointer Reference (Indirection) Address Type Cast (Conversion)
3	Left to Right	* / %	Multiplication Division Modulus
4	Left to Right	+ -	Addition Subtraction
5	Left to Right	<< >>	Left Shift Right Shift
6	Left to Right	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to
7	Left to Right	== !=	Equality Inequality
8	Left to Right	&	Bitwise AND
9	Left to Right	^	Bitwise XOR
10	Left to Right	|	Bitwise OR
11	Left to Right	&&	Logical AND
12	Left to Right	||	Logical OR
13	**Right to Left**	?:	Conditional Expression
14	**Right to Left**	==, *=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=	Assignment Operators
15	Left to Right	,	Comma Operator

3. Explain Expression and Expression Evaluation in C.

- An arithmetic expression is a combination of variables, constants and operators arranged as per the syntax of the language.
- For example, a = b + c; denote an expression in which there are 3 operands a,b,c and two operator + and =.
- Expressions are evaluated using an assignment statement of the form, variable = expression
- When the statement is encountered, the expression is evaluated first and the result is assigned to the variable on the left hand side of equal to operator.
- All variables used in expression must be assigned values before evaluation starts.

Rules for evaluation of expression

1. First sub expressions inside the parenthesis (bracket) are evaluated.
2. The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions.
3. The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
4. Arithmetic expressions are evaluated from left to right using the rules of precedence.

EXAMPLE:

Solve the following two expression where take a=5, b=3, c=4, d=2;

1.)a+b*c/d*d+a-b

= 5+3*4/2*2+5-3

= 5+12/2*2+5-3 (Same Priority of *, / but if we go left to right then first * comes so we had perform that)

= 5+6*2+5-3 (Same Priority is of *, / but if we go left to right the first / comes so we had perform that)

= 5+12+5-3 (Highest Priority is of * so we had perform that)

= 17+5-3 (Same Priority is of +, - but if we go left to right the first + comes so we had perform that)

= 22-3 (Same Priority is of +, - but if we go left to right the first + comes so we had perform that)

= 19 ←ANS

2.)(a+b*(c/d)*(d+a-b))

= (5+3*(4/2)*(2+5-3))

= (5+3*(2)*(7-3)) (First of all Whatever are in brackets it will execute first then other things will be execute, if more than one operators are there, with same priority then go to the Left to Right and which comes first it will performs first)

= (5+3*(2)*(4))

= (5+3*8) (Brackets execute first)

= (5+24) (Highest priority is of '*' from *, +)

= 29 ← ANS

4. Formatted Input and Output:

- Formatted Input & Output refers to an input data that has been arranged in a particular format.

1. Formatted Input & Output for Integer Numbers:

- The field specification for reading an integer number is: %w sd
- *w* is an integer number that specifies the field width of the number to be read and *d* known as data type character, indicates that the number to be read is in integer mode.
- Example of Formatted Input and Output for Integer No. is as below;

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int no1,no2;
   printf("\nEnter Two Integer Numbers Here:");
   scanf("%2d %5d",&no1,&no2);
   printf("\nValue of No1 is: %2d",no1);
   printf("\nValue of No2 is: %5d",no2);
   printf("%d\n",no1);
   printf("%6d\n",no1);
   printf("%2d\n",no1);
   printf("%-6d\n",no1);
   printf("%06d\n",no1);
   getch();
}
```

2. Formatted Input and Output for Real Numbers:

- Unlike integer numbers, the field width of real numbers is not to be specified and therefore scanf reads real numbers using the simple specification %f for both the notations, namely, decimal point notation and exponential notation.
- Example of Formatted Input and Output of Real No. is as below;

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float fno1,fno2;
    printf("\nEnter  Two Real Numbers Here:");
    scanf("%f %f",&fno1,&fno2);
    printf("\nValue of No1 is: %.2f",fno1);
    printf("\nValue of No2 is: %.5e",fno2);
    printf("%7.4f\n",fno1);
    printf("%7.2f\n",fno1);
    printf("%-7.2f\n",fno1);
    printf("%f\n",fno1);
    printf("%10.2e\n",fno1);
    printf("%e\n",fno1);
    getch();
}
```

- Here First Real No.'s output, will contain 2 digits after points.
- Here Second Real No.'s output, will print in to the exponential format.
- After that different types of output is shown for real numbers.

3. Inputting Character and Writing Character:

- Reading a single character can be done by using the function getchar(), with the help of scanf function also.
- The getchar() takes the following form; `Variable_name = getchar();`
- Variable_name is a valid C name that has been declared as char type.
- Like getchar(), there is another function putchar() for writing characters one at a time to the terminal.
- It takes following form; `putchar(variable_name);`
- Where variable_name is a type char variable containing a character. This statement displays the character contained in the variable_name at the terminal.
- Example of Inputting character using getchar() and putchar() function, is as below;

```
#include<stdio.h>
#include<conio.h>
void main()
{
   char ch;
   printf("Enter Your Character Here:");
   ch = getchar();
   printf("You had entered: ");
   putchar(ch);
   getch();
}
```

4. Input and Output Character string :

- We had already seen how a single character can be read from the terminal using the getchar function.
- The same can be achieved using the scanf function. In addition, a scanf function can input strings containing, more than one character.
- Following are the specifications for reading character strings; *%ws or %wc*
- To read character a-z we can implement following code,

```
#include<stdio.h>
#include<conio.h>
void main()
{
char str1[50];
printf("\nEnter String wth Numbers Here:");
   scanf("%[a-z]",str1);
   printf("%s\n\n",str1);
  getch();
}
```

- To read character with space, then we can implement following code,

```
#include<stdio.h>
#include<conio.h>
void main()
{
   char str1[50];
   printf("\nEnter String wth Numbers Here:");
   scanf("%[^\n]",str1);
   printf("%s\n\n",str1);
  getch();
}
```

- The format specification for outputting strings is similar to that of real numbers. It is of the form,

  %w.ps

- where *w* specifies the field width for display and *p* instructs that only the first *p* characters of the string are to be displayed. The display is right-shifted.

```
#include<stdio.h>
#include<conio.h>
   void main()
   {
        char ch[30]="RAJKOT 360022";
        printf("Output of the String is Here:");
        printf("\n%s",ch);
        printf("\n%20s",ch);
        printf("\n%20.10s",ch);
        printf("\n%.5s",ch);
        printf("\n%-20.10s",ch);
        printf("\n%5s",ch);
        getch();
   }
```

Sr. No.	Guess the output (Question)	Output

1.	#include<stdio.h>
	int main()
	{
	int k, n=40;
	k = (n>5 ? (n <=10 ? 50 : 60): 70);
	printf("%d\n",k);
	return 0;
	}
		60

2.	#include<stdio.h>
	int main()
	{
	int r=5,p;
		1

	p= r - sizeof(float);
	printf("%d", p);
	return 0;
	}	

3.	#include<stdio.h>
	int main()
	{
	int n ;
	n = -32768;
	printf(“%d” , n-1);
	return 0;
	}
		-32769

4.	#include<stdio.h>
	int main()
	{
	char ch;
	ch = ‘A’ ;
	printf(“%c” , ch+25);
	return 0;
	}
		Z

5.	#include<stdio.h>
	int main()
	{
	printf(“%d , %d” , 10/3 , 10%3);
	return 0;
	}
		3, 1

6.	#include<stdio.h>
	int main()
	{
	int n1,n2;
	n1=81;
	n2=181;
	printf ("%d" , n1 > n2 ? n1 : n2);
	return 0;
	}
		18

7.	(10+10)/10*10-10	10

Sr. No.	**Evaluate the following expression**
1.	If b=4, c=2, d=2 → a=b*c/d-2+b/c a = 4*2/4-2+4/2 a = 8/4-2+4/2 a = 2-2+4/2 a = 2-2+2 a = 0+2 a = 2

2. If b=4, c=2, d=2 → a=b*(c/d)-(2+b)/c
a = 4*(2/2)-(2+4) /2
a = 4*1 – 6/2
a = 4 – 6/2
a = 4 - 3
a = 1

3. (10 + 10) / 10 * 10 – 10
20 / 10 * 10 – 10
2 * 10 – 10
20 – 10
10

4. 5 % 4 + 5 / (1+ 2)
5 % 4 + 5 / 3
1 + 5 / 3
1 + 1
2

5. 8/4/2
2/2
1

6. (125*(-3)-42)(19+432%66)
(-375-42) * (19+36)
- 417 * 55
-22935

7. 20.0 – (5.5/(0.5*10.0))+15.0
20.0 – (5.5/5)+15.0
20.0 – 1.1 + 15.0
18.9 + 15.0
33.9

8. c=a-- + ++b, where a=8 and b=7.
c = 8 + 8
c = 16

Sr. No. **Convert the following expression in to C**

1. **$X^2 + y^2+4xySin(y)$**
(x*x) + (y*y) + (4 *x*y*Sin(y))

2. **9xy/p+1-1/3(m+n)**
9*x*y / p+1-1 /3*(m+n)