While Loop	Do-While Loop
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
If there is a single statement, brackets are not required.	Brackets are always required.
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);

Sr.no	If……else	Switch
1.	In this we can test only one condition.	In this we can test multiple condition.
2.	It can have values based on constraints.	It can have values based on user choice.
3.	In this we cannot have different conditions.	In this case we can have only one expression but various value of the same expression.
4.	If else statement is used to evaluate a condition to be true or false	A switch case statement is used to test for multiple values of the same variable or expression like 1,2,3 etc.

Ans:

Call by Value	Call by Reference
In this case the value of the parameters is passed to the caller function.	In this case the reference of the variable is passed to the function by passing the address of parameters.
In this case the actual parameters are not accessible by the called function.	In this case, since the address of the variables are available, the called function can access the actual parameters.
This is implemented by using simple variable names.	This is implemented by the use of pointer variables.
Hence the actual parameters remain unchanged in case of the call by value.	Hence the actual parameters can be altered if required in case of the call by reference method.

Break	Continue
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements.
A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered.	A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue.
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops. You will get an error if this appears in switch statement.

When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.

No.	Structure	Union
1.	Keyword struct defines a structure.	Keyword union defines a union.
2.	The members of a structure all begin at different offsets from the base of the structure.	In a union all members have offset zero from the base
3.	Within a structure all members gets memory allocated; therefore any member can be retrieved at any time.	While retrieving data from a union the type that is being retrieved must be the type most recently stored.
4.	One or more members of a structure can be initialized at once.	A union may only be initialized with currently required type.
5.	Data is more secured in structure.	Data can be corrupted in a union.
6.	Memory allotted to structure is equal to the space require collectively by all the members of the structure.	Memory allotted for a union is equal to the space required by the largest memory of that union
7.	Structure provides ease of programming.	Unions are difficult for programming.
8.	Structures require more memory.	Unions require less memory.
9.	Structure must be used when information of all the member elements of a structure are to be stored.	Unions must be used when only one of the member elements of the union is to be stored.
10.	struct s_name { int ival; float fval; }s;	union u_name { int ival; float fval; }u;

Basis	ENTRY CONTROLLED LOOP	EXIT CONTROLLED LOOP
Execution Flow	A loop in which the given condition is checked first, before entering the loop body	A loop in which the loop body is executed first and then after the given condition is checked
Condition Evaluation	The loop body would be executed, only if the given condition is true	The loop body would be executed at least once, even if the given condition is evaluated as false
Example	For Loop and While Loop are examples of this type of loop	Do While Loop is an example of exit controlled loop
Use	It is used when condition evaluation is mandatory before executing loop body	It is used when one requires to iterate through loop body at least once before condition evaluation

SR. NO.	KEYWORD	IDENTIFIER
1	Keywords are predefined word that gets reserved for working program that have special meaning and cannot get used anywhere else.	Identifiers are the values used to define different programming items such as variables, integers, structures, unions and others and mostly have an alphabetic character.
2	Specify the type/kind of entity.	Identify the name of a particular entity.
3	It always starts with a lowercase letter.	First character can be a uppercase, lowercase letter or underscore.
4	A keyword should be in lower case.	An identifier can be in upper case or lower case.
5	A keyword contains only alphabetical characters.	An identifier can consist of alphabetical characters, digits and underscores.
6	They help to identify a specific property that exists within a computer language.	They help to locate the name of the entity that gets defined along with a keyword.
7	No special symbol, punctuation is used.	No punctuation or special symbol except 'underscore' is used.
8	Examples of keywords are: int, char, if, while, do, class etc.	Examples of identifiers are: Test, count1, high_speed, etc.

13.1Distinguish between structures and unions:

	Structure	Union
1	A structure is defined with 'struct' keyword	A union is defined with 'union' keyword
2	All members of a structure can be manipulated simultaneously	The members of a union can be manipulated only one at a time
3	Memory is allocated for all variables.	Allocates memory for variable which variable require more memory.
4	All members of structure can be initialized	Only the first member of a union can be initialized.
5	Structure member are allocated distinct memory location	Union members share common space for their exclusive use
6	Syntax: struct struct_name { structure element 1; --------- structure element n; }struct_var_nm;	Syntax: union union_name { union element 1; --------- union element n; }union_var_nm;
7	Example: struct item_mst { int rno; char nm[50]; }it;	Example: union item_mst { int rno; char nm[50]; }it;

---

2. **Algorithm:**
   What do you mean by algorithm? Which points you should consider while developing the algorithm?

Ans.: An algorithm is a finite set of statements, each of which has a clear meaning and can be executed in a finite amount of time and with a finite amount of effort. Thus whatever is the size of input data, the algorithm can solve the given problem in finite amount of time.

Points to consider while developing the algorithm:

i) **Non – ambiguity:** This property of an algorithm indicates that each of the statement in the algorithm must be clear and precise. There must be no ambiguity in any of the statement.

ii) **Range of Input:** The range of the input for which the algorithm works is also to be compulsorily mentioned in algorithm. There should be clear indication for the range of input for which the algorithm may fail.

iii) **Multiplicity:** The algorithm can be represented in multiple ways. An algorithm can be written English language or by the graphical representation called as flowchart or pseudo code.

iv) **Speed:** One of the important properties of an algorithm is that it should produce the result efficiently and at faster speed.

v) **Finiteness:** The algorithm should be finite i.e. there should be no infinite condition leading to never ending procedure and hence never completing the task.

Q1. b) What is an error ? Explain different types of errors occurred in program.

Ans.

Error :-

While writing c programs, errors also known as bugs in the world of programming may occur unwillingly which may prevent the program to compile and run correctly as per the expectation of the programmer.

Types of errors :-

Basically there are three types of errors in c programming:

1. Runtime Errors :

   C runtime errors are those errors that occur during the execution of a c program and generally occur due to some illegal operation performed in the program. For example,
   - Dividing a number by zero
   - Trying to open a file which is not created
   - Lack of free memory space

2. Compile Errors :-

   Compile errors are those errors that occur at the time of compilation of the program. C compile errors may be further classified as:

   2.1 Syntax Errors :

   When the rules of the c programming language are not followed, the compiler will show syntax errors.

   2.2 Semantic Errors :

   Semantic errors are reported by the compiler when the statements written in the c program are not meaningful to the compiler.

3. Logical Errors :-

   Logical errors are the errors in the output of the program. The presence of logical errors leads to undesired or incorrect output and are caused due to error in the logic applied in the program to produce the desired output. Also, logical errors could not be detected by the compiler, and thus, programmers have to check the entire coding of a c program line by line.

Differentiate between system function and user-defined function:

	System function	User Defined function
1	They are predefined	They are created by user
2	Part of header file	Part of program which compiles runtime
3	Name is given by developer	Name of function is decided by user
4	Cannot be changed	Can be changed

Differentiate between gets() and scanf():

gets()	scanf()
Used to read only string data	Used to read all types of data including string
Just reads a line and stores into one variable	Passes input and reads data into one or more variable
It stores it into a string until a newline character (\n) or end of line is reached	Stores according to parameter format into locations pointed by additional argument.

Q1. d) Explain any four standard library functions from sting.h ?
Ans.

Standard Library Functions from string.h :-

In the C Programming Language, the Standard Library Functions are divided into several header files.

The following is a list of functions found within the <string.h> header file:

I. Comparison functions
   1. memcmp - Compare Memory Blocks
   2. strcmp - String Compare
   3. strcoll - String Compare Using Locale-Specific Collating sequence.
   4. strncmp - Bounded String Compare
   5. strxfrm - Transform Locale-Specific String

II. Concatenation functions
   1. strcat - String Concatenation
   2. strncat - Bounded String Concatenation

III. Copying functions
   1. memcpy - Copy Memory Block
   2. memmove - Copy Memory Block
   3. strcpy - String Copy
   4. strncpy - Bounded String Copy

IV. Search functions
   1. memchr - Search Memory Block for Character
   2. strchr - Search String for Character
   3. strcspn - Search String for Intial Span of Characters Not in Set

---

   4. strpbrk - Search String for One of a Set of Characters
   5. strrchr - Search String in Reverse for Character
   6. strspn - Search String for Initial Span of Characters in Set
   7. strstr - Search String for Substring
   8. strtok - Search String for Token

V. Miscellaneous functions
   1. memset - Initialize Memory Block
   2. strerror - Convert Error Number to String
   3. strlen - String Length

Q1. e) Explain break and continue statement with example.

Ans.

It is sometimes desirable to skip some statements inside the loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used.

break Statement :-

The break statement terminates the loop (for, while and do...while loop) immediately when it is encountered. The break statement is used with decision making statement such as if...else. In C programming, break statement is also used with switch...case statement.

Syntax of break statement :

```
break;
```

Example :-

Code :

```
// Program to calculate the sum of maximum of 10 numbers
// Calculates sum until user enters positive number
# include <stdio.h>
# include <conio.h>
int main()
{
int i;
double number, sum = 0.0;
for (i=1; i <= 10; ++i)
        {
printf("Enter a n%d: ",i);
```

1.

```
scanf("%lf",&number);
// If user enters negative number, loop is terminated
if(number < 0.0)
        {
            break;
        }
        sum += number; // sum = sum + number;
 }
 printf("Sum = %.2lf",sum);
 return 0;
}
```

Output :

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

continue Statement :-

The continue statement skips some statements inside the loop. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement :

```
continue;
```

Example :-

Code :

```
// Program to calculate sum of maximum of 10 numbers
// Negative numbers are skipped from calculation
# include <stdio.h>
# include <conio.h>
int main()
{
   int i;
   double number, sum = 0.0;

  for(i=1; i <= 10; ++i)
  {
     printf("Enter a n%d: ",i);
     scanf("%lf",&number);
     // If user enters negative number, loop is terminated
     if(number < 0.0)
     {
        continue;
     }
     sum += number; // sum = sum + number;
  }
  printf("Sum = %.2lf",sum);
  return 0;
}
```

Output :

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

Q3. b) Define pointer and its use. Explain array of pointer with example. Write program to swap the values by using call by reference concept.

Ans.

Pointer :-

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is −

type *var-name;

Uses of pointer :-

1. Pointers reduce the length and complexity of a program.
2. They increase execution speed.
3. A pointer enables us to access a variable that is defined outside the function.
4. Pointers are more efficient in handling the data tables.
5. The use of a pointer array of character strings results in saving of data storage space in memory.

Array of Pointers :-

Just like we can declare an array of int, float or char etc, we can also declare an array of pointers, here is the syntax to do the same.

Syntax :

datatype *array_name[size];

Example :

int *arrop[5];

Here arrop is an array of 5 integer pointers. It means that this array can hold the address of 5 integer variables, or in other words, you can assign 5 pointer variables of type pointer to int to the elements of this array. arrop[i] gives the address of i th element of the array. So arrop[0] returns address of variable at position 0, arrop[1] returns address of variable at position 1 and so on. To get the value at address use indirection operator (*). So *arrop[0] gives value at address[0], Similarly *arrop[1] gives the value at address arrop[1] and so on.

not true for the first statement.

7.1.4 Difference between for, while and do-while loop:

SN	For Loop	While Loop	Do – while loop

1	for(initializations; condition; increment / decrement / updating)
	{
	statements;
	}
		while(condition)
		{
		statement;
		updating;
		}
			do
			{
			statement;
			updating;
			}while(condition);

2	▯ This is used when certain statements are to be executed for a fixed number of times. ▯ This is achieved by initializing a loop counter to a value and increasing or decreasing the value for certain no of times until a condition is satisfied. ▯ The loop statements are executed once for every iteration of the 'for' loop.	▯ This is called as An Entry controlled loop, as the entry inside the loop is possible only if the condition is true.	▯ This is called as An Exit controlled loop, as the entry inside the loop is sure i.e. no condition is checked to enter inside the loop. ▯ But the exit is possible only if the condition is false.
3	▯ The initialization statement is executed only once at the beginning of the 'for' loop. ▯ Then the test expression is checked by the program. If the test expression is false, for loop is terminated. ▯ But if the test expression is true, then the code inside the body of loop is executed and then update expression is updated.	▯ If the condition is not true for the first time, the control will never enter into the loop. ▯ Hence there is a possibility that the control never enters into loop and the statements inside the loop are never executed.	▯ Even if the condition is not true for the first time the control will enter into the loop. ▯ Hence the statements inside the loop will be executed at least once even if the condition is
	This process repeats until test expression is false.		not true for the first time.
4	▯ Conditions are separated by semicolons. You are not required to put a statement here, as long as a semicolon appears.	▯ There is no semicolon (;) after the condition in the syntax of the while loop	▯ There is semicolon (;) after the condition in the syntax of the do - while loop

Explain the advantages of using Function

Code reusability: Functions allow you to write code once and reuse it multiple times throughout your program. This can save time and reduce the amount of code you need to write.

2. Improved readability: Functions can make your code more readable by breaking it up into smaller, more manageable pieces. This can make it easier to understand what your code is doing and to find and fix bugs.
3. Reduced complexity: Functions can help reduce the complexity of your code by encapsulating complex logic into a single function. This can make your code easier to understand and maintain.
4. Improved modularity: Functions can help improve the modularity of your code by allowing you to group related code together. This can make it easier to test and debug your code.
5. Improved maintainability: Functions can make your code easier to maintain by allowing you to make changes to a single function without affecting the rest of your code.

6. Improved performance: Functions can improve the performance of your code by allowing you to optimize specific sections of your code. This can help reduce the overall runtime of your program.

Arrays:

- An array is a collection of **homogeneous** elements, meaning all elements have the same data type (e.g., integers, characters, etc.).
- Elements in an array are stored at **contiguous memory locations**.
- Accessing array elements is done using **subscripts** (square brackets) with an index.
- Arrays are useful for storing a fixed number of elements of the same type.
- The size of an array is fixed and determined by the number of elements multiplied by the size of each element.
- Arrays are not objects and cannot be instantiated directly.

```
int myArray[5]; // Declares an array of 5 integers
```

Structures:

- A structure is a **user-defined data type** that can group items of **different data types** into a single type.
- Elements in a structure are called **members** and can have different data types.
- Structure members may or may not be stored in contiguous memory locations.
- Accessing structure members is done using the **dot (.) operator** followed by the member name.
- Structures allow you to create custom data structures with various fields.
- The size of a structure is not fixed because each member can have a different type and size.

Example of a simple structure:

```
struct Person {
    char name[50];
    int age;
    float salary;
};
```

S N	Call by value	Call by reference
1.	Calling by value makes a copy of the variable so the function can use it.	Calling by reference sends the memory address of the variable to the function so the function works with the original value.
2.	In the call by value method, the called function creates the new set of variables in stack and copies the values in the arguments into them.	In the call by reference method, instead of passing values to the function being called, references / pointers to the original variables are passed
3.	In this technique, the changes made in the function definition are not reflected in main program	In this technique, the changes made in the function definition are reflected in main program
4.	Pointer is not used in call by value	Pointer is used in call by reference

16.1.3 Difference between call by value and call by reference:

What are the different types of variables?

- Automatic variables:
  - Automatic variables are declared inside a function in which they are to be utilized.
  - They are created when the function is called and destroyed automatically when the function is exited
  - The keyword 'auto' is used to declare automatic variables.

```
void main()
{
    auto int number;
}
```

- External variables:
  - Variables that are both alive and active throughout the entire program are known as external variables.
  - They are also known as global variables.
  - External variables are declared outside the function.

```
int count;
void main()
{
    count=0;
}
function()
{
    int count=0;        count=count+1;
}
```

- Static variables:
  - The value of static variables persists until the end of the program.
  - A variable can be declared static using the keyword 'static'
  - A static variable may be either an internal type or an external type depending on the place of declaration.
  - Internal static variables can be used to retain values between function calls.

```
void stat(void);
void main()
```

```
{
    int i;
    for(i=0;i<=3;i++)
    stat();
}
void stat(void)
{
    static int x=0;
    x=x+1;
    printf("x=%d\n",x);
}
```

- Register variables:
  - We can tell the compilers that a variable should be kept in one of the machines register, instead of keeping in the memory.
  - Since the register access is much faster than the memory access, keeping the frequently accessed variables in the register will lead to faster execution of programs.
  - The keyword 'register is used to declare register variables. E.g.: register int count;
  - C can automatically convert register variables into non-register variables once the limit is reached.

13. Unions:

- Union is a collection of multiple data elements that can be of different data types.
- The memory space required to store one variable of union is equal to the memory space required by the largest element in the union.
- Syntax of union declaration:

```
union union _name
{
    data_type variable_name;
    data_type variable_name;
    -
};
```

- Syntax of declaration of a variable of the union:
  union union_name union_variable_name;

12.2Nested Structures:

- If one of the members of the structure is also a structure, then such a structure is called as a nested structure.
- Syntax of structure declaration:

```
struct structure_name
{
        data_type variable_name;
        struct
        {
                data_type variable_name;
                -
        }internal_structure_name;
        -
};
```

- Syntax of accessing an internal structure element:
  structure_variable name.internal_structure_name.element_name;

12. Structures:

- Structure is a collection of multiple data elements that can be of different data types.
- The memory space required to store one variable of structure is equal to the memory space required by all the elements independently to be stored in memory.
- Syntax of structure declaration:

```
struct structure_name
{
    data_type variable_name;
    data_type variable_name;
    -
    -
};
```


- Syntax of declaring a structure variable:
  struct structure_name structure_variable_name;
- Syntax of accessing a structure element:
  structure_variable_name.variable_name;

11.1 String functions:

11.1.1 strlen() function:

- Returns an integer value that is the length of the string passed to the function
- char a[] = {'J', 'A', 'Y'}
- printf("%d", strlen(a)); //3
- When returning the length of the string it does not consider the null character.
- Syntax : int strlen(char a[])

11.1.2 strcpy() function:

- Copies the second string into the first string passed to the function.
- char a[] = {'J', 'A', 'Y'}
- char b[10];
- strcpy(newString, oldString);

11.1.3 strcmp() function:

- Compare the two string variables passed to the function.
- It returns an integer value equal to
  - Zero – if two strings are equals.
  - Negative value – if first string is smaller than the second string
  - Positive value – if first string is greater than the second string
- The comparison of strings is done based on the alphabetical order.

11.1.4 strcat() function:

- Concatenates the two string variables passed to the function.
- The concatenated string is stored in the first string which is passed to the function.

11.1.5 strrev() function:

- Reverse the string passed to the function.

11.1.6 strlwr() function:

- Convert all the uppercase characters in the string, which is provided as parameter, into lower case.

11.1.7 strupr() function:

- Convert all the lowercase characters in the string, which is provided as parameter, into uppercase.

11.1.8 strncpy() function:

- strncpy(str 1, str 2, n) copies up to n characters from the string pointed by str 2 to the string pointed to by str 1 and terminates str 1 with a null. The str 2 must be a pointer to a null terminated string.

11.1.9 strchr() function:

- strchr(str,ch) finds a character in string. Returns a pointer to the first occurrence of the character in string, if ch does not occur in str , strchr returns NULL.

11.1.10 strstr() function:

- strstr(str 1, str 2) finds the first occurrence of substring str 2 in str 1. Returns a pointer to the elements in str 1 that contain str 2 , or NULL if str 2 does not occur in str 1.

10. Arrays:

10.1 Single Dimensional Arrays:

- Collection of multiple data of same datatypes
- All elements of array have to be of same type only.
- The starting index i.e. index of the first element of an array is always zero.
- The index of last element is n-1, where n is the size of an array.
- An array has static memory location i.e. memory size once allocated for an array cannot be changed.
- Syntax: data_type array_name[array_size];
  where, data_type is the datatype like int, float, double. array_name is identifier i.e. name of the variable. array_size determines size of the array.

9.1 Recursive Functions:

- A function that calls itself is called as recursive function.
- A recursive function must definitely have a condition that exits from calling the function again.
- A recursive function must have a condition which calls the function again if it true and exits the loop if the condition is false; or vice-versa.
- C does not allow the main function to call itself. Recursion is a powerful technique using which some problems can be expressed in a form which is very close to their natural statement.
- Syntax:

```
return_Type Function_Name ( argument list)
{
        Function_name( );
}
```

Difference between function declaration and function definition.

Function declaration	Function definition
It provides information about the function definition to the compiler, so	It contains following information: 1.storage class(optional) 2. return type

that compiler knows that such function exit.	3. name of function 4.argument list 5.function body

Syntax :
Return_Type Function_Name(type Parm_name 1,type parm_name 2);
	Syntax :
	Return_Type Function_name(Formal argument)
	{
	Function Body
	}

There is no semicolon at the end of the declaration.	There is no semicolon at the end of the header.
There is no function body.	The function body follows header.

Example-
float square(float x);
	Example-
	float volume (int x, float y, float z)
	{
	Float v = a*b*c;
	.....
	}

9. Functions:

- A function is a self-contained block of statements that can be called and used whenever required.
- A number of statements grouped into a single logical unit are called a function.
- The use of function makes programming easier. Since repeated statements can be grouped into functions, splitting the program into separate function make the program more readable and maintainable.
- A function definition has two principal components: the function header and body of the function. The function header is the data type of return value followed by function name and (Optionally) a set of arguments separated by commas and enclosed in parenthesis.

- Syntax –

```
return_type function_name(argument list)
{
        -
        Statements;
        -
}
```

- Important points:
  - Return_type – value the function returns to the caller function
  - Void function – does not return any data
  - Actual parameters – parameters passed by the caller function
  - Formal parameters – parameters received by the called function
  - Number of actual and formal parameters will always be same
  - If a function is called before its definition, then a prototype declaration of that function must be written before calling the function.

7.2.3 Difference between switch statement and nested if else:

SN	The nested if…else	The switch-case
1.	Is a two – way branching.	Is a multi – way branching.
2.	Is complicated.	Is easier to write.
3.	It can take many levels and it becomes difficult to match the else part to its corresponding if.	No such problem occurs in switch – case.
4.	Debugging becomes difficult.	Tracing of error and debugging is easier.
5.	The test expression can be constant value or may involve any type of operators.	Only constant integer expressions and values are allowed.

7.2 Conditional Statement:

7.2.1 If-else Statements:

- Used to check the condition and accordingly execute a set of statements, based on whether the condition is true or false.
- Syntax – if(condition)

```
{
  Statements1;
  -
}
else
{
  Statements2;
  -
}
```

- if-else Ladder – used to check multiple cases of a particular condition.

```
if(condition)
{
  Statements1;
  -
}
else if(condition)
{
  Statements2;
  -
}
  else{
                Statements3;
        }
```

7.2.2 Switch-case Statements:

- Syntax – switch(expression/variable)

```
{
  case label1: Statements;
                break;
  case label2: Statements;
                break;
  -
  -
  case labeln: Statements;
                break;
```

default: Statements; }

=	**Simple assignment operator**, assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	**Add AND assignment operator**, it adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	**Subtract AND assignment operator**, it subtracts right operand from the left operand and assign the result to left operand	C -= A equivalent to C = C – A
*=	**Multiply AND assignment operator**, it multiplies right operand with the left operand and assign the result to left operand	C *= A equivalent to C = C * A
/=	**Divide AND assignment operator**, it divides left operand with the right operand and assign the result to left operand	C /= A equivalent to C = C / A
%=	**Modulus AND assignment operator**, it takes modulus using two operands and assign the result to left operand	C %= A equivalent to C = C % A
<<=	**Left shift AND assignment operator**	C<<= 2 equivalent to C = C << 2
>>=	**Right shift AND assignment operator**	C>>= 2 equivalent to C = C >> 2
&=	**Bitwise AND assignment operator**	C &= 2 equivalent to C = C & 2
^=	**Bitwise exclusive OR and assignment operator**	C ^= 2 equivalent to C = C ^ 2
|=	**Bitwise inclusive OR and assignment operator**	C |= 2 equivalent to C = C | 2

6.2.5 Ternary operator:

- Used to check a condition and accordingly do one thing based on the condition being true or false.
- Syntax: variable x = (condition)? value if true : value if false
- exp 1 ? exp 2 : exp 3
- if exp 1 is true,
  evaluate exp 2
  else
  evaluate exp 3
- E.g.: int a , b;
  a = 10;
  b = (a == 1)? 20: 30;
  Output: b=30

3. Assignment operators:

&	**Bitwise AND Operator** copies a bit to the result if it exists in both operands.	(A & B) = 12 i.e. 0000 1100
|	**Bitwise OR Operator** copies a bit if it exists in either operand.	(A | B) = 61 i.e. 0011 1101
^	**Bitwise XOR Operator** copies the bit if it is set in one operand but not both.	(A ^ B) = 49 i.e. 0011 0001
~	**Bitwise Not Operator** is unary and has the effect of 'flipping' bits.	(~A) = -60 i.e. 1100 0011
<<	**Bitwise Left Shift Operator**. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e. 1111 0000
>>	**Bitwise Right Shift Operator.** The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e. 1111

Examples:

5 & 3 = 1
$(5)_{10} = (0\ 1\ 0\ 1)_2$
$(3)_{10} = (0\ 0\ 1\ 1)_2$
$(1)10 = (0\ 0\ 0\ 1)_2$

12 | 9 = 13
$(12)_{10} = (1\ 1\ 0\ 0)_2$
$(9)_{10} = (1\ 0\ 0\ 1)_2$
$(13)10 = (1\ 1\ 0\ 1)_2$

8 ^ 10 = 2
$(8)_{10} = (1\ 0\ 0\ 0)_2$
$(10)_{10} = (1\ 0\ 1\ 0)_2$
$(2)10 = (0\ 0\ 1\ 0)_2$

~7 = -8
$(7)_{10} = (0\ 1\ 1\ 1)_2$
$(-8)_{10} = (1\ 0\ 0\ 0)_2$
10 << 2 = 40
Assuming that data to be char i.e. 8 bit data
$(10)_{10} = (0\ 0\ 0\ 0\ 1\ 0\ 1\ 0)_2$
After shifting left once $(0\ 0\ 0\ 1\ 0\ 1\ 0\ 0)_2$
After shifting left for the second time $(0\ 0\ 1\ 0\ 1\ 0\ 0\ 0)2 = (40)_{10}$

11 >> 3 = 1
Assuming that data to be char ie 8 bit data
$(11)_{10} = (0\ 0\ 0\ 0\ 1\ 0\ 1\ 1)_2$
After shifting left once $(0\ 0\ 0\ 0\ 0\ 1\ 0\ 1)_2$
After shifting left for the second time $(0\ 0\ 0\ 0\ 0\ 0\ 1\ 0)_2$
After shifting left for the third time $(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)_2 = (1)_{10}$