UNIT-3 Multiple

Choice Questions:

1. To perform a set of instructions repeatedly which of the following can be used?

A.for

B.while

C.if-else-if

D.both i & ii

2.

```
int main() { int i
= 1 ; while(
i<=2)
printf("%d",i);
return 0;
   }
```

What is output of above code

A.12

B.i

C.Compilation error

D.indefinite loop

3. When _____ is encountered inside any loop, Control automatically passes to the first statement after loop.

A.Continue

B.break

C.goto

D.return

4. This Loop tests the condition after having executed the Statements within the Loop.

A.while

B.do-while
C.for Loop
D.if-else-if

Choose a right C Statement.
A) Loops or Repetition block executes a group of statements repeatedly.
B) Loop is usually executed as long as a condition is met. C) Loops usually take advantage of Loop Counter
D) All the above.

5. What is the output of C Program.? int main()
{
```
    while(true)
    {
        printf("RABBIT");
        break;
    }

    return
0; }
```
A) RABBIT
B) RABBIT is printed unlimited number of times.
C) No output **D) Compiler error**.

6. What is the way to suddenly come out of or Quit any Loop in C Language.?
A) continue; statement
B) break; statement
C) leave; statement
D) quit; statement

7. Choose facts about continue; statement is C Language.

A) continue; is used to take the execution control to next iteration or sequence B) continue; statement causes the statements below it to skip for execution

C) continue; is usually accompanied by IF statement.

D) **All the above.**

8. Choose a correct statement about C break; statement.?

A) break; statement can be used inside switch block

B) break; statement can be used with loops like for, while and do while.

C) break; statement causes only the same or inner loop where break; is present to quit suddenly. **D) All the above.**

9. Choose a correct C Statement regarding for loop. for(; ;);

A) for loop works exactly first time

B) for loop works infinite number of times

C) Compiler error

D) None of the above

10. What is an Array in C language.?

A) A group of elements of same data type.

B) An array contains more than one element

C) Array elements are stored in memory in continuous or contiguous locations.

D) **All the above.**

11. Choose a correct statement about C language arrays.

A) An array address is the address of first element of array itself.

B) An array size must be declared if not initialized immediately.

C) Array size is the sum of sizes of all elements of the array.

D) All the above

12. An array Index starts with.?

A) -1
B) 0
C) 1
D) 2

13. What is an array Base Address in C language.?
A) Base address is the address of 0th index element.
B) An array b[] base address is &b[0]
C) An array b[] base address can be printed with printf("%d", b);
D) All the above

14. What is the size of an array in the below C program statement.? int main() { int ary[9]; return 0;
}
A) 8
B) 9
C) 10
D) None of the above

15. What is the minimum and maximum Indexes of this below array.?
int main()
{ int
ary[9];
return 0; }
A) -1, 8
B) 0, 8
C) 1,9
D) None of the above

16. What is the dimension of the C array int ary[10][5].?
A) 1
B) 2
C) 5
D) 10

17. What is the dimension of the below C Array.?
int ary[]={1,3,5,7};
A) 1
B) 2
C) 3
D) 5

18. Array of Arrays is also called.?
A) Multi Data Array
B) Multi Size Array
C) Multi Dimensional Array
D) Multi Byte Array

19. What is a String in C Language.?
A) String is a new Data Type in C
B) String is an array of Characters with null character as the last element of array. C) String is an array of Characters with null character as the first element of array
D) String is an array of Integers with 0 as the last element of array.

20. Choose a correct statement about C String.
char ary[]="Hello..!";
A) Character array, ary is a string.
B) ary has no Null character at the end
C) String size is not mentioned
D) String can not contain special characters.
What is the Format specifier used to print a String or Character array in C Printf or Scanf function.?
A) %c
B) %C
C) %s
D) %w

21. What is the output of C Program with
Strings.? int main() {

```
    char str[]={'g','l','o','b','e'};
    printf("%s",str);
return 0;
}
```

A) g
B) globe
C) globe\0
D) None of the above

22. How do you accept a Multi Word Input in C Language.?
A) SCANF
B) GETS
C) GETC
D) FINDS

23. A character constant is enclosed by.?
A) Left Single Quotes
B) Right Single Quotes
C) Double Quotes
D) None of the above

24. A C string elements are always stored in.?
A) Random memory locations
B) Alternate memory locations
C) Sequential memory locations
D) None of the above

25. Choose a correct C statement about String functions.? A) int n=strlen("abc") returns 3.
B) strupr("abc") returns ABC
C) strlwr("Abc") returns abc
D) All the above

26. What is the output of C program.? int main() { char

```
str1[]="JAMES,";     char
str2[15]="BOND ";
strcat(str2,str1);
printf("%s",str2);
printf("%s",str1);
}
```

A) JAMES BOND,JAMES,

B) JAMES,JAMES,

C) BOND JAMES,JAMES,

D) None of the above

27. Choose a correct C statement about String functions.?

A) toupper('a') returns

A B) tolower('D')

returns d.

C) strcmp("123","12345") returns a negative number

D) All the above

28. What is the output of C program.?

```
int
main() {
    printf("%c","HUMPTY"[2]);
}
```

A) U

B) M

C) HUMPTY

D) None of the above

29. What is the output of C program with String arrays.? int main() {

```
    char code[3][4]={"IN","USA","K"};
    printf("%s", code[1]);
    return 0;
}
```

A) IN

B) USA

C) K

D) Compiler error

Descriptive Questions:

1. Explain while statement with syntax and example The syntax of a while loop is:

```
while (testExpression)
{
  Statement block
} where, test Expression checks the condition is true or false
```

before each loop.

The while loop evaluates the test expression.If the test expression is true (nonzero), codes inside the body of while loop are executed. The test expression is evaluated again. The process goes on until the test expression is false. When the test expression is false, the while loop is terminated. While loop is entry controlled loop.

Example

```
/* Programm to print numbers from 1 to 9*/:
#include
<stdio.h>
main() {   int
i=1;

  while(i<10)
  {
     printf("%d\n",i);
     i++;
  }

}
```

2. Explain do..while statement with syntax and example do...while loop in C programming checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

Syntax :

```
do {
       statement(s);

     } while (condition );
```

if the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false. Example

```
 main ()
{
    int a = 10;
   /* do loop execution
*/    do {
      printf("  %d\n",
a);          a = a + 1;
}while( a < 20 );
 }
```

When the above code is compiled and executed, it prints values from 10 to 19

3. Explain for statement with syntax and example

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Syntax

The syntax of a for loop in C programming language is – for (init_exp; test_exp; update_exp)

```
{

statement(s);
}
```

- The init_exp step is executed first, and only once. This step allows you to initialize loop control variables.
- Next, the test_exp is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.

- After the body of the 'for' loop executes, the flow of control jumps back up to the update_exp This statement allows you to update any loop control variable
- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself. After the condition becomes false, the 'for' loop terminates.

Example

```
#include
<stdio.h> main
()
 {
   int  i;
   /* for loop execution */
   for( i = 1;i < 10;i++ ){
printf("%d\n",,i);
   }
}
```

When the above code is compiled and executed, it prints values from 1 to 9

4. Explain the following:

i. Nested loops

C programming allows to use one loop inside another loop. The following section shows a few examples to illustrate the concept. The syntax for a nested for loop statement in C is as follows –

```
for ( init_exp; test_exp; update_exp )
 {

  for( init_exp; test_exp; update_exp)
{
     statement(s);
   }
   statement(s);
}
```

The syntax for a nested while loop statement in C is as follows –

```
while(condition) {

   while(condition) {
statement(s);
   }
   statement(s);
}
```

The syntax for a nested do...while loop statement in C is as follows –

```
do {
   statement(s);

   do {
      statement(s);
}while( condition );

}while( condition );
```

Example

C program to print the number pattern.

```
1
1                                                                    4
1 2                                                                  4
1 2 3                                                               17
1 2 3 4                                                             17



#include
<stdio.h>
```

```
main() {     int
i=1,j;
    while (i <= 5)
    {
j=1;
        while (j <= i )
        {
            printf("%d ",j);
            j++;
}
printf("\n");
i++;
    }
    }
```

ii. Infinite loops

A loop becomes an infinite loop if a condition never becomes false. Loop will not terminate

Example

```
int i = 1
while(i<10)
{
    printf("%d\n", i);
}
```

Here we are not updating the value of i. So after each iteration value of i remains same. As a result, the condition (i<10) will always be true. For the loop to work correctly add i++;

5. Differentiate break and continue statements

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. Syntax of break statement

```
                    break;
```

How break statement works?

Example

```
main ()
{

   int a = 1 ;
  /* while loop execution */
  while( a < 10 )
  {
    printf("value of a: %d\n", a);
    a++;
if( a >  5)
      {

        /* terminate the loop using break
statement */          break;
     }
  }


}
```

The continue statement skips some statements inside the loop and goes back to beginning of loop for executing next iteration.. The continue statement is used with decision making statement such as if...else.

Syntax of continue Statement

```
                    continue;
```

How continue statement works?

```
while (test Expression)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

```
for (init, condition, update)
{
    // codes
    if (condition for continue)
    {
        continue;
    }
    // codes
}
```

Example

```
#include
<stdio.h>
main ()  {   int
a = 1 ;
  /* do loop execution */
  do {
if( a ==  5)
```

Entry Control Loop	Exit Control Loop
Entry control loop checks condition first	The exit control loop first executes the body of the l checks condition at last.

{	and then body of the loop will be executed.	

```
a ++;
continue;
      }
      printf("value of a: %d\n", a);
      a++;       }
while( a < 10 );
}
```

6. Differentiate entry-controlled and exit-controlled loops

7. Explain one-dimensional array with the help of suitable code example .
   Like any other variable, arrays must be declared before they are used.
   General form of array declaration is,
   data-type variable-name[size];

   int arr[10];

- Here int is the data type, arr is the name of the array and 10 is the size of array. It means array arr can only contain 10 elements of int type.
- Index of an array starts from 0 to size-1 i.e first element of arr array will be stored at arr[0]address and the last element will occupy arr[9].

Initialization of an Array

Entry Control Loop	Exit Control Loop
The body of the loop may or may not be executed at all.	The body of the loop will be executed at least once condition is checked at last
for, while are an example of an entry control loop	Do...while is an example of an exit control loop.

After an array is declared it must be initialized. Otherwise, it will contain garbage value . An array can be initialized at either compile time or at runtime. Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

data-type array-name[size] = { list of values };

/* Here are a few examples */ int marks[4]={ 67, 87, 56, 77 }; /* integer array initialization*/ float area[5]={ 23.4, 6.8, 5.5 }; /* float array initialization*/

int marks[4]={ 67, 87, 56, 77, 59 }; /* Compile time error*/

If we omit the size of the array, an array size is equal to number elements assigned. int balance[] = {1000 , 200, 300, 700, 500}; in the above example size of array is 5.

char b[]={'C','O','M','P','U','T','E','R'}; /*character Array*/

Example:

```
/*  Program to initialize array at run time.*/
#include<stdio.
h> main() {
int a[4];
   int i, j;
   printf("Enter array element:");
for(i=0; i<4; i++)
   {
```

```
        scanf("%d",&a[i]);    //Run time array initialization
    }
    printf("Value in Array:");
for(j=0; j<4; j++)
    {
        printf("%d \n",a[j]);
    }
getch();
}
```

8. Explain two-dimensional array with the help of suitable code example .

It is an ordered table of homogeneous elements. It is generally referred to as matrix, of some rows and some columns.

The general form of two dimensional array is storage_class data_type array~name[rows][columns] for example int Mat[3][4]; declares that "Mat" is a two dimensional array of type int with 3 rows and 4 columns. The subscript value ranges from 0 to 2 for rows and for columns it range from 0 to 3 (4 columns). Storage class is optional.

In general an array of the order M X N (read as M by N) consists of M rows, N columns and MN elements. It may be depicted as shown below.

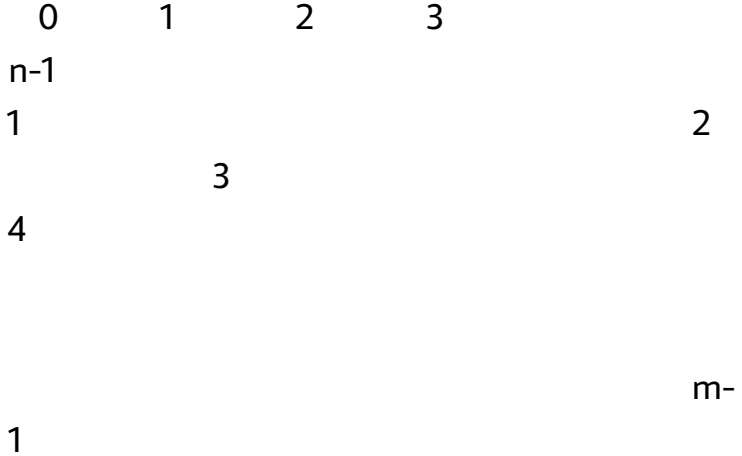

Fig. Two - dimensional array of the order M x N

Initialization of two dimensional array

The general form of two - dimensional array initialization is .
data_type array _name[row][column]={ value 1, value2, ..
valuen} where
storage_class is optional.
Data_type is the basic data type, array_name is the name of a two dimensional array. value 1, value2, ... valuen are the initial values to be assigned to a two dimensional array.
Example:
int mat [2][2] = { 1,2,3,4}; then the elements of matrix will be mat[O][O] = 1 mat[O][1] = 2 mat[l][O] = 3 mat[l][l] = 4 If the number of elements to be assigned are less than the total number of elements, that two dimensional array contained, then all the remaining~ elements are assigned zeros.

The statement

int mat [2][2] = { 1,2,3,4 }; can be equivalently written as int mat [2][2] = { { 1,2}, {3, 4 } }; by surrounding the elements of each row by braces.

If the values are missing in an initialization, then that is automatically set to zero.
For example the statement int mat [2][3] = { { 1, 1 } }, { 2 } }; will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.
Example:

```
#include<stdio.
h>  int
main(){     int
i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
for(j=0;j<3;j++){
  printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
```

```
 }//end of j
}//end of i
return 0;
}
```

9. Explain the different ways available for the initialization of string variable . Character arrays can be initialized in two ways. That is, characterr arrays may be initialized when they are declared.

```
char subject[9] = {'c',' o' ,'m' ,'p','u' ,'t',' e' ,'r' ,'\0' }
```

Even though there are eight characters in the word "computer", the character array
"subject" is initialized to 9. This is because to store the null ('\0') character at the end
of the string. When character array is initialized by listing its elements, null character must be supplied explicitly as shown in the above example. Consider another example char subject[9]={ "computer"};
Here each character of the string "computer" is stored in individual elements of a character array subject, but it automatically adds null character at the end of the string. It is also possible to initialise a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements in the given string. .
For example

static char month[ ] = "April"
Defines the character array month as a 6 element array.

Reading String From Terminal

The input function scanf ( ) with %s format specification can be used to read an array of characters.
Example

```
char Subject[10];
scanf("%s" ,Subject);
```

If we enter subject name as "Computer Science" then only the first word "Computer" will be assigned to the variable Subject This is because, the scanf( ) function terminates its input on the first occurrence of the white space.

In such cases, when line of text has to be read from the terminal one can make use of gets() function or one can make use of getchar() function repeatedly to read successive single characters from the keyboard and place them into a character array.

¬¬

Example Program 1

```
# include <stdio.h>
        # include <string.h>
main()
{
char text[50], ch;
int i=0;
printf("Enter line of text and press enter key at the end\n");
while((ch=getchar())!='\n')
(
text[i]=c
h; i++; }
text[i]='\0' /*put null at the end*/
printf("Entered line of the text:\n%s"
,text); }
```

10. Write a program to sort elements of an integer array.

Program to input n numbers and sort it in ascending order using bubble sort.

```
#include<stdio.h>
main()
 {
   int a[10],n,i,j,temp;
   clrscr();
```

```
    printf("Enter number of    elements:");
   scanf("%d",&n);
   printf("Enter array
elements:");    for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)    for(j=0;j<n-
i-1;j++)   if(a[j]>a[j+1])
  {
temp=a[j];
a[j]=a[j+1];
    a[j+1]=temp;
   }
  printf("Sorted array
is\n");   for(i=0;i<n;i++)
printf("%d\t",a[i]);
getch();
 }
```

11. Write a program to search for an element in an integer array.
    Program to search a number in a list, using linear search technique. If present print its position(s). #include<stdio.h> main()

```
 {
   int a[10],n,x,i,flag=0;
   clrscr();
   printf("Enter number of elements:");
scanf("%d",&n);
   printf("Enter array
elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Enter search
element:");
   scanf("%d",&x);
   for(i=0;i<n;i++)
   {
if(a[i]==x)
```

```
 {
   printf("%d is found at the location
%d\n",x,i);    flag=1;
  }  }
if(flag==0
)
  printf("Given element not found");
  getch();
}
```

12. Explain with example how to declare, initialize and use strings in C.

Declaring String Variables

The general form of declaration of a string variable is char string_name[size];

Where string_name is a valid variable name and the size determines' the number of characters in the string_name. For example char name [20] ; When the compiler assigns a string data to character array, it automatically supplies a null character ('\0') at the end of the string. Therefore the array size should be one more than maximum number of characters in the string.

Initializing Strings

Character arrays can be initialized in two ways. That is, characterr arrays may be initialized when they are declared.

```
char subject[9] = {'c',' o' ,'m' ,'p','u' ,'t',' e' ,'r' ,'\0' }
```

Even though there are eight characters in the word "computer", the character array "subject" is initialized to 9. This is because to store the null ('\0') character at the end of the string. When character array is initialized by listing its elements, null character must be supplied explicitly as shown in the above example. Consider another example char subject[9]={ "computer"};

Here each character of the string "computer" is stored in individual elements of a character array subject, but it automatically adds null character at the end of the string. It is also possible to initialise a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements in the given string. .
For example

static char month[ ] = "April"
Defines the character array month as a 6 element array.

Reading String From Terminal

The input function scanf ( ) with %s format specification can be used to read an array of characters.
Example

```
char Subject[10];
scanf("%s" ,Subject);
```

If we enter subject name as "Computer Science" then only the first word "Computer" will be assigned to the variable Subject This is because, the scanf( ) function terminates its input on the first occurrence of the white space.
In such cases, when line of text has to be read from the terminal one can make use of gets() function or one can make use of getchar() function repeatedly to read successive single characters from the keyboard and place them into a character array.
¬¬
Example Program 1

```
# include <stdio.h>
      # include <string.h>
main()
{
```

```
char text[50], ch;
int i=0;
printf("Enter line of text and press enter key at the end\n");
while((ch=getchar())!='\n')
(
text[i]=c
h; i++; }
text[i]='\0' /*put null at the end*/
printf("Entered line of the text:\n%s" ,text);
}
```

Writing Strings To Screen

The printf function with %s format specifier can be used to display the strings on the screen. For example the following statement

```
printf ("%s", Subject);
```

Can be used to display the contents of the character array Subject.
The %s format conversion character with an optional w.d specification can be used to format the output. In this case 'd' number of characters from the beginning of the string will be printed in a field of width w.

Example Program 2

```
#include <stdio.h>
main()
{ clrscr(); printf ("%20s", "Left justified
printing.\n"); printf ("%-40.24s", "Left
justified printing.\n");   printf ("%-
40.16s", "Left justified printing.\n");
printf ("%-40.12s", "Left justified
printing.\n");   printf ("%-40.8s", " Left
justified printing.\n");   printf ("%-40s",
```

```
"Left justified printing.\n");   printf ("%-
40.0s", " Left justified printing.\n");
printf ("%40.25s", "Right justified
printing.\n"); printf ("%40.20s", "Right
justified printing.\n");   printf ("%40.5s",
"Right justified printing.\n");  printf
("%40.0s", "Right jusdfied printing.\n");
}
```

The output of the program illustrates the following features of the %s specifications.

1. When the field width is less than the length of the string, the entire string is printed.
2. The integer value on the right side of the decimal point specifies the number of characters to be printed.
3. When the number of characters to be printed is specified as zero, nothing is printed.
4. The minus (-) sign in the specification causes the string to be printed left justified.

13. List and explain any four-string handling functions available in C

1. strcat( ) function

This function is used to concatenate two strings, i.e., it appends one string at the end of another string. This function accepts two strings as parameters and adds the contents of the second string at the end of the first string. Its syntax is follows strcat (string!, string2); here, the string string2 is appended to the end of string1.

Example Program 3

```
/* Program to concatenate two given strings */
#include
<stdio.h>
#include
```

```
<string.h>
main() {
char strl [50], str2[25];
clrscr();
printf("Enter the first string\n");
gets( strl);
printf("Enter the second string\n");
gets(str2);
strcat(strl, str2);   1* concatenate strl and str2 */
printf("Concatenated string :\t %s", strl);
getch();
}
```

2. strlen() function

strlen() function returns the number of characters in the given string. Here, the length does not include the terminating character '\0' (NULL). Its syntax is as follows.

Len = strlen(string);

Where Len is an integer type variable and string is an one dimensional array of characters.

Example Program 4

```
/* Program to find the length of the entered string */
#include
<stdio.h>
#include
<string.h>
main() { char
str[50]; clrscr();
printf("Enter the string\n");
gets( str) ;
```

```
printf("Length of the entered string: %d",
strlen(str)); getch();
        }
```

3. strcmp () function

This function is used to compare two strings. The function accepts two strings as parameters and returns an integer, whose value is:
Less than 0, if the first string is less than the second string Equal to 0, if both are identical
Greater than 0, if the first string is greater than the. second string. The general form of strcmp() function is as follows strcmp (stringl, string2);
The strcmp() function compares two strings, character by character, (ASCII comparision) to decide the greatest one. Whenever two characters in the string differ, there is no need to compare the other characters of the strings.

Example Program 5

```
#include <string.h>
#include <stdio.h>
main()
 {
char Strl [] = "aaa", Str2[] = "bbb", Str3[] =
"ccc";  int ptr; clrscr();
ptr = strcmp(Str2, Str]);
if (ptr > 0)
    printf("String 2 is greater than String 1 \n");
       else
printf("String 2 is less than String 1
\n"); ptr = strcmp(Str2, Str3);
if (ptr > 0)
  printf("String 2 is greater than String
3\n"); else
```

```
printf("String 2 is less than String 3\n");
getch();
 }
```

4. strcpy() function

This function copies one string to another. The strcpy() function accepts two strings as parameters and copies the second string, character by character into the first string, upto and including null character of the second string. Its syntax is as follows strcpy (string 1, string2);

Example Program 6

```
#include
<stdio.h>
#include
<string.h>
main() {
char string[25];
char strl [ ] = "Computer
Science";  clrscr( );
strcpy(string, strl);
printf("Copied string: %s\n", string);
getch();
}
```

14. Mention suitable string functions to do the following:
    a. To find length of the string strlen() function returns the number of characters in the given string. Here, the length does not include the terminating character '\0' (NULL). Its syntax is as follows.

    Len = strlen(string);

    Where Len is an integer type variable and string is an one dimensional array of characters.

    Example Program 4

    /* Program to find the length of the entered string */

```
#include
<stdio.h>
#include
<string.h>
main() { char
str[50]; clrscr();
printf("Enter the string\n");
gets( str) ;
printf("Length of the entered string: %d",
strlen(str)); getch();
    }
```

b. To copy one string to another string

This function copies one string to another. The strcpy() function accepts two strings as parameters and copies the second string, character by character into the first string, upto and including null character of the second string. Its syntax is as follows strcpy (string 1, string2);

Example Program 6

```
#include
<stdio.h>
#include
<string.h>
main() { char
string[25];
char strl [ ] = "Computer
Science";  clrscr( );
strcpy(string, strl);
printf("Copied string: %s\n", string);
getch();
}
```

c. To add two strings

This function is used to concatenate two strings, i.e., it appends one string at the end of another string. This function accepts two strings

as parameters and adds the contents of the second string at the end of the first string. Its syntax is follows strcat (string!, string2); here, the string string2 is appended to the end of string1.

Example Program 3

```
/* Program to concatenate two given strings */
#include <stdio.h>
#include <string.h>
main()
{
char strl [50], str2[25];
clrscr();
printf("Enter the first string\n");
gets( strl);
printf("Enter the second string\n");
gets(str2); strcat(strl, str2);  1*
concatenate strl and str2 */
printf("Concatenated string :\t %s", strl);
getch();
}
```

d. To reverse given string strrev() is a non-standard C library function, sometimes found in <string.h>, that is used to reverse a string.
Example:

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

int main() {
    char str[20] = "coffee";

    printf("String before strrev(): %s\n",str);

    strrev(str);
```

```
    printf("String after strrev(): %s\n",str);

    return 0;
}
```

15. What are the possible values that the function strcmp( ) can return? What do they mean?
    This function is used to compare two strings. The function accepts two strings as parameters and returns an integer Return Value from strcmp()

Return Value	Remarks
0	if strings are equal
>0	if the first non-matching character in `str1` is greater (in ASCII) than that of `str2`.
Return Value	Remarks
<0	if the first non-matching character in `str1` is lower (in ASCII) than that of `str2`.

Example:

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

int main () {
char str1[15];
char str2[15];
   int ret;


   strcpy(str1, "abcdef");
   strcpy(str2, "ABCDEF");

   ret = strcmp(str1, str2);

   if(ret < 0) {
      printf("str1 is less than str2");
   } else if(ret > 0) {
      printf("str2 is less than str1");
   } else {
      printf("str1 is equal to str2");
   }

   return(0);
}
```

UNIT-4 Multiple Choice Questions:

1. Choose correct statement about Functions in C Language.

A) A Function is a group of c statements which can be reused any number of times.

B) Every Function has a return type.

C) Every Function may no may not return a value.

D) **All the above.**

2. Choose a correct statement about C Language Functions.

A) A function name can not be same as a predefined C Keyword.

B) A function name can start with an Underscore(_) or A to Z or a to z.

C) Default return type of any function is an Integer.

D) **All the above.**

3. Choose a correct statement about C Function.?

```
main() {
    printf("Hello");
}
```

A) "main" is the name of default must and should Function.

B) main() is same as int main()

C) By default, return 0 is added as the last statement of a function without specific return type.

D) All the above

4. A function which calls itself is called a ___ function.

A) Self Function

B) Auto Function

C) Recursive Function

D) Static Function

5. What is the output of C Program with functions.?

```
int main() {     show();
printf("BANK ");
    return 0;
}

void show()
{
    printf("CURRENCY ");
}
```

A) CURRENCY BANK

B) BANK CURRENCY

C) BANK

D) **Compiler error**

6. How many values can a C Function return at a time.?

A) Only One Value

B) Maximum of two values

C) Maximum of three values

D) Maximum of 8 values

7. What is the output of C Program with functions.? int show();

```
void
main() {
    int a;
    printf("PISTA COUNT=");
    a=show();
printf("%d", a);
}

int
show() {
    return 10;
}
```

A) PISTA COUNT=

B) PISTA COUNT=0

C) PISTA COUNT=10

D) Compiler error

8. What are types of Functions in C Language.?

A) Library Functions

B) User Defined Functions

C) Both Library and User Defined

D) None of the above

9. What is the limit for number of functions in a C Program.?

A) 16

B) 31

C) 32

D) None of the above

10. Every C Program should contain which function.?

A) printf()

B) show() C) scanf()

D) main()

11. What is the minimum number of functions to be present in a C Program.?

A) 1

B) 2

C) 3

D) 4

12. What characters are allowed in a C function name identifier.?

A) Alphabets, Numbers, %, $, _

B) Alphabets, Numbers, Underscore (_)

C) Alphabets, Numbers, dollar $

D) Alphabets, Numbers, %

13. Arguments passed to a function in C language are called ___ arguments.

A) Formal arguments

B) Actual Arguments

C) Definite Arguments

D) Ideal Arguments

14. Arguments received by a function in C language are called ___ arguments.

A) Definite arguments

B) Formal arguments

C) Actual arguments

D) Ideal arguments

15. Choose a corrects statement about C language function arguments.
A) Number of arguments should be same when sending and receiving
B) Type of each argument should match exactly
C) Order of each argument should be same
D) All the above

16. Choose a non Library C function below.
A) printf()
B) scanf()
C) fprintf()
D) printf2()

17. What is the default return value of a C function if not specified explicitly.? A) -1
B) 0
C) 1
D) None of the above

18. A recursive function can be replaced with __ in c language.
A) for loop
B) while loop
C) do while loop
D) All the above

19. A recursive function without If and Else conditions will always lead to.?
A) Finite loop
B) Infinite loop
C) Incorrect result
D) Correct result

20. What is the C keyword that must be used to achieve expected result using Recursion.? A) printf
B) scanf

C) void
D) **return**

21. How many functions are required to create a recursive functionality.?
A) One
B) Two
C) More than two
D) None of the above

22. Choose a correct statement about Recursive Function in C language.
A) Each recursion creates new variables at different memory locations
B) There is no limit on the number of Recursive calls
C) Pointers can also be used with Recursion but with difficulty.
D) All the above

23. Identify wrong C Keywords below.
A) auto, double, int, struct
B) break, else, long, switch
C) case, enum, register, typedef
D) char, extern, intern, return

24. What is a C Storage Class.?
A) C Storage decides where to or which memory store the variable.
B) C Storage Class decides what is the default value of a variable.
C) C Storage Class decides what is the Scope and Life of a variable.
D) **All the above.**

25. Find a C Storage Class below.
A) static
B) auto
C) register & extern
D) All the above

26. What is the default C Storage Class for a variable.?
A) static

B) auto

C) register

D) extern

27. Variables of type auto, static and extern are all stored in .?

A) ROM

B) RAM

C) CPU

D) Compiler

28. Which among the following is a Local Variable.?

A) register

B) auto

C) static

D) extern

29. which among the following is a Global Variable.?

A) auto

B) register

C) static

D) extern

30. Choose a correct statement about static variable.

A) A static global variable can be accessed in other files.

B) **A static global variable can be used only in a file in which it is declared.**

C) A static global variable can not be declared without extern keyword.

D) Default value of a static variable is -1.

Descriptive Questions:

1) Explain with example how to declare, define and call a function in C

Function prototype (Declaration)

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.
The function prototype is not needed if the user-defined function is defined before the
main() function._____
Syntax of function prototype
returnType functionName(type1 argument1, type2 argument2,...);
Function definition
Function definition contains the block of code to perform a specific task. Syntax of function definition
return type functionName(data-type argument1, data-type argument2, ...) { Local Variable Declarations

Execution statement(s)

[return value/expression]
}
Where .
Return data type- type of the return value by the functions.
If nothing is returned to the calling function, then data type is void.
Function_name - It is the user defined function name.
Argument(s) - The argument list contains valid variable name separated by commas. . return statement- It is used to return value to the calling function. A function can have multiple return statements . But only one return statement is executed (A function can return only single value). Execution of return statement causes execution to be transferred back to calling function.
When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.
Calling a Function:
A function can be accessed or called by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. If the function call does not require any arguments, an empty pair of parentheses must follow the function name.

2) Explain with the help of a code example declaring, defining and calling a function in C

Function prototype (Declaration)

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

The function prototype is not needed if the user-defined function is defined before the

main() function._____

Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2,...);

Function definition

Function definition contains the block of code to perform a specific task. Syntax of function definition

return type functionName(data-type argument1, data-type argument2, ...) { Local Variable Declarations

Execution statement(s)

[return value/expression]

}

Where .

Return data type- type of the return value by the functions.

If nothing is returned to the calling function, then data type is void.

Function_name - It is the user defined function name.

Argument(s) - The argument list contains valid variable name separated by commas. . return statement- It is used to return value to the calling function. A function can have multiple return statements . But only one return statement is executed (A function can return only single value). Execution of return statement causes execution to be transferred back to calling function.

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

Calling a Function:

A function can be accessed or called by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas. If the function call does not require any arguments, an empty pair of parentheses must follow the function name.

Example Program 1

```
   # include <stdio.h>
int Add(int x, int y);  /*function prototype or function
declaration*/ main()
      {
       int a, b, sum;
      printf("enter two
number");
scanf("%d %d", &a, &b);
      sum=Add(a,b );
 printf("sum of %d and %d is %d", a, b, sum);
   }/*End of the main*/

int Add (int x, int y) /*Function to add two numbers */
{
  return (x + y);
}
```

3) Explain with example function with no arguments and no return values Such functions can either be used to display information or they are completely dependent on user inputs.

Below is an example of a function, which takes 2 numbers as input from user, and display which is the greater number.

```
#include<stdio.h>
void greatNum();       // function
declaration int main() {
```

```
    greatNum();        // function
call     return 0;
}

void greatNum()        // function definition
{
    int i, j;
    printf("Enter 2 numbers that you want to compare...");
scanf("%d%d", &i, &j);
    if(i > j) {
        printf("The greater number is: %d", i);
    }
else {
        printf("The greater number is: %d", j);
    }
}
```

4) Explain with example function with arguments and no return value

We are using the same function as example again and again, to demonstrate that to solve a problem there can be many different ways. This time, we have modified the above example to make the function greatNum() take two integer values as arguments, but it will not be returning anything.

```
#include<stdio.h>
void greatNum(int a, int b);       // function
declaration int main()
{
    int i, j;
    printf("Enter 2 numbers that you want to
compare...");     scanf("%d%d", &i, &j);
greatNum(i, j);        // function call     return 0;
}

void greatNum(int x, int y)        // function definition
```

```
{     if(x >
y) {
        printf("The greater number is: %d", x);
    }
else {
        printf("The greater number is: %d", y);
    }
}
```

5) Explain with example function with no arguments but a return value

We have modified the above example to make the function greatNum() return the number which is greater amongst the 2 input numbers.

```
#include<stdio.h>

int greatNum();       // function declaration

int main() {     int result;     result =
greatNum();        // function call
printf("The greater number is: %d",
result);     return 0;
}

int greatNum()        // function definition
{
    int i, j, greaterNum;
    printf("Enter 2 numbers that you want to
compare...");     scanf("%d%d", &i, &j);     if(i > j)
{
        greaterNum = i;
    }
    else {
        greaterNum = j;
    }
    // returning the result
return greaterNum;
```

```
}
```

6) Explain with example functions with arguments and a return value

This is the best type, as this makes the function completely independent of inputs and outputs, and only the logic is defined inside the function body.

```
#include<stdio.h>
int greatNum(int a, int b);       // function
declaration int main() {
   int i, j, result;
   printf("Enter 2 numbers that you want to
compare...");     scanf("%d%d", &i, &j);     result
= greatNum(i, j); // function call     printf("The
greater number is: %d", result);     return 0;
}

int greatNum(int x, int y)        // function definition
{     if(x >
y) {
return x;
   }     else
{
return y;
   }
}
```

7) Write a note on a) nesting of functions b) recursive functions

Nesting of Functions

C language also allows nesting of functions i.e to use/call one function inside another function's body. We must be careful while using nested functions, because it may lead to infinite nesting.

```
function1()
{
   // function1 body here
function2();
```

```
    // function1 body here
}
```

If function2() also has a call for function1() inside it, then in that case, it will lead to an infinite nesting. They will keep calling each other and the program will never terminate.

Not able to understand? Lets consider that inside the main() function, function1() is called and its execution starts, then inside function1(), we have a call for function2(), so the control of program will go to the function2(). But as function2() also has a call to function1() in its body, it will call function1(), which will again call function2(), and this will go on for infinite times, until you forcefully exit from program execution.

Recursive functions are functions that call themselves, so a recursion is process of calling function by itself.

Example program 7

```
/*Recursive function program to find the factorial of a number*/

#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
clrscr( );
 printf("Entered number: %d",num);
 printf("\nFactorial of the entered number: %d",
fact(num));  getch( ); }
int fact(int num)
{
if(num==0
```

```
) return
(1); else
 return (num * fact(num-1);
}
```

Suppose the definition fact is called with the value 2, then the value of num in the function fact is 2, and the condition in the if statement is false, and the function gets called again with the value num - 1, which is 1. In the second invocation of fact, the value of num is 1. The condition in the if statement is again false, and a third invocation of fact results, passing 0 to it. In the third invocation, the value of num 0, and the condition of fact returns 1 to the second invocation. Hence, the statement return (num * fact (num-1);

Multiplies 1 by 1 and causes the number 1 to be returned to the first invocation; where the value of num was 2. This brings back to the statement return (num * fact(num-l));

Which is now executing in the first invocation. The return value of the second invocation, 1 is multiplied by 2 and the value 2 is returned. Two important conditions must be satisfied by any recursive function:

i) Each time when a function calls itself, it must be closer, in some sense to a solution. ii) There must be a decision criterion for stopping the process or computation.

For the function factorial, each time when the function calls itself, its argument is decremented by one. The stopping criterion is the if statement that checks for the zero argument.

8) What are the actual and formal arguments?

Function parameters are the means of communication between the calling and the called functions. They can be classified into formal parameters and actual parameters. The formal parameters are the parameters given in the function declaration and function definition. The actual parameters(commonly called arguments), are specified in the function call.

```
main ( )
{
   ---------    --
-------
                actual parameters
  function1(a1, a2,  a3 am);   /* function call*/
¬  -------
--   -----
---- }


function1  (fl,   f2,    f3 fn)     /* called function*/
{       formal parameters
  --------
  --------
}
```

Arguments matching between the function call and the called function.

The actual and formal arguments should match in number, type and order. The values of actual arguments are assigned to the formal arguments on a one to one basis starting with the first argument as shown in figure. In case, the actual arguments are more than the formal parameters (m > n), the extra actual arguments are discarded. But if the actual arguments are less than the fomal arguments, the unmatched formal arguments are initialized to some garbage values. Any mismatch in data type may also result in passing of garbage values.

The formal parameters must be valid variable names, but the actual arguments may be variable names, expressions or constants. The variables used in actual arguments must be assigned values before the function call is made.

Example

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

```
int Add(int x, int y);  /*function prototype or function
declaration*/ main()
      {
  int a, b, sum;
printf("enter two
number");   scanf("%d
%d", &a, &b);
sum=Add(a,b );
 printf("sum of %d and %d is %d", a, b, sum);
 }/*End of the main*/

int Add (int x, int y) /*Function to add two numbers */
{
  return (x + y);
}
```

9) What is a recursive function explain with example?

Recursive functions are functions that call themselves, so a recursion is process of calling function by itself.

Example program 7

```
/*Recursive function program to find the factorial of a number*/

#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
clrscr( );
 printf("Entered number: %d",num);
```

```
 printf("\nFactorial of the entered number: %d",
fact(num));  getch( ); }
int fact(int num)
{
if(num==0
)  return
(1); else
 return (num * fact(num-1);
}
```

Suppose the definition fact is called with the value 2, then the value of num in the function fact is 2, and the condition in the if statement is false, and the function gets called again with the value num - 1, which is 1. In the second invocation of fact, the value of num is 1. The condition in the if statement is again false, and a third invocation of fact results, passing 0 to it. In the third invocation, the value of num 0, and the condition of fact returns 1 to the second invocation. Hence, the statement return (num * fact (num-1);

Multiplies 1 by 1 and causes the number 1 to be returned to the first invocation; where the value of num was 2. This brings back to the statement

return (num * fact(num-l));

Which is now executing in the first invocation. The return value of the second invocation, 1 is multiplied by 2 and the value 2 is returned. Two important conditions must be satisfied by any recursive function:

i) Each time when a function calls itself, it must be closer, in some sense to a solution. ii) There must be a decision criterion for stopping the process or computation.

For the function factorial, each time when the function calls itself, its argument is decremented by one. The stopping criterion is the if statement that checks for the zero argument.

10) Write recursive function to calculate factorial of a given number.

/*Recursive function program to find the factorial of a number*/

```
#include<stdio.h>
int fact(int n);
main( )
(
 int num;
clrscr( );
 printf("\nEnter a number :");
 scanf("%d" ,&num);

/* processing and output part */
clrscr( );
 printf("Entered number: %d",num);
 printf("\nFactorial of the entered number: %d",
fact(num));  getch( ); }
int fact(int num)
{
if(num==0
)  return
(1); else
 return (num * fact(num-1);
}
```

11)Write a note on automatic variable.

The variables in main() are private or local to main(). Because they are declared within main, no other function can have direct access to them. The same is true of the variables in other functions. Each local variable in a function comes into existence only when the function is called and disappeared when the function is exited. Such variables are usually known as automatic variables.We can make use of the optional keyword in the declaration, auto for Example

```
Auto int  x;
```

auto has been the default class for all the variables we have used so for. The value of the automatic variables cannot be changed in some other function

in the program. So, we can declare and use the same variable name in different functions in the same program.

Example:

```
#include<stdio.h
> int Add (int
a,int b);  main( )
{
auto int a=10,b=15,c;
clrscr( ); c=
a+b; printf("c=
%d", c); c=
Add(a,b);
printf("\nc= %d", c);
      }
int Add( int a, int b)
      {
       int c;
c= a+b;
return c;
  }
```

12) Write a note on register variable.

Register variables are local variables (similar to automatic variables) except that they are placed in machine registers. A register declaration advises the compiler that the variable in question will be heavily used. The idea is that register variables are to be placed in machine registers, which may result in smaller and faster programs. But compilers are free to ignore the advice.

The register declaration looks like

register int i;

Then the compiler may try to place the variable 'i' in a machine register if available.

The register declaration can only be applied to automatic variables and the parameters of a function. Since only a few variables can be placed in the

register, 'C' will automatically convert register variables into auto variables once the limit is reached. Loop indices, accessed more frequently, can be declared as register variables. For example 'index' is a register variable in the program given below.

Example: #
inc1ude<stdio.h
> main( )
{

register index, Sum = 0;
for (index== 1; index<==100;
index++) Sum == Sum + index;
printf(" Sum == %d", Sum);
}
Finally one cannot assign large or aggregate data types, such as doubles or arrays to registers and the & operator cannot be applied to register variables.

13) Write a note on external variable.

. External variables are defined outside of any function, and are thus available to many functions. Hence, they are also known as global variables. By default, external variables and functions have the property that all references to them by the same name.

Because external variables are globally accessible, they provide an attention to function arguments and return values for communicating data between functions. Any function may access an external variable by referring to it by name. If a number of variables must be shared among functions, external variables are more convenient and efficient than long argument lists. External variables are also useful because of their greater scope and lifetime. Automatic(local) variables are internal to a function; they come into existence when the function is entered, and disappeared (destroyed) when it is left. External variable on the other hand, are permanent, so they retain values from one function invocation to the next.

Thus if functions must share some data, yet neither calls the other, it is often most convenient if the shared data is kept in external variables rather than passed in and out via arguments.

One important feature of global variable is that if a function has a local variable, of the same name as a global variable, the local variable has precedence over the global variable. Analogously, a variable declared within a block has precedence over an external variable of the same name.

In order to be able to use an externally declared variable inside' a function, such a variable may explicitly redeclared inside the function with the extern qualifier, for example: extern int X;

Example:

```
#include
<stdio.h> int
Sum; void Inc(
); main( ) {  int
a,b;
 clrscr( );
 priiltf("Enter two
numbers");  scanf("%d
%d", &a,&b);  Sum = a+b;
printf("Sum of %d and %d is %d", a,b,Sum);
Inc( );
printf("\nAfter incrementing the value of Sum is
%d",Sum); getch( ); }
 void Inc( )
{
Sum ++;
}
```

14) Write a note on static variable.

Another C storage declaration is static. The keyword static is used to declare variables which must not lose their storage locations or their values when control leaves the functions or blocks where in they are defined. In C, static variables retain their values between function calls. The initial value assigned to a static variable must be a constant, or an expression

involving constants. But static variables are by default initialized to 0, in contrast to autos.

For example

Static int i;
Initalises variable i to 0

¬Example Program 13

```
#include<stdio.h
> void
Display(void);
main( ) {
int i;
clrscr( );
for ( i=1; i<=4; i++)
Display( );
}
void Display(void)
 {
   static int x= 0;
x++;
  printf("\nx=%d" ,x);
}
```

output
: x=1
x=2
x=3
x=4
A static variable is initialized only once, when the program is compiled. It is never initialized again. During the first call to Display( ), x is incremented to l. Because x is static, this value retains and therefore, the next call adds

another I to x giving its value equal to 2. The value of x becomes 3 when third call is made and 4 when fourth call is made.

UNIT – 5

Multiple Choice Questions:

1. Which of the following are themselves a collection of different data types?

   A. String
   B. **Structures**
   C. Char
   D. None of the above

2. Which operator connects the structure name to its member name?

   A. - B. ->
   C. .
   D. both . and ->

3. Which of the following cannot be a structure member?
   A. **Function**
   B. Array
   C. Structure
   D. None of the above

4. Union differs from structure in the following way

   A. All members are used at a time
   B. **Only one member can be used at a time** C. Union cannot have more members
   D. Union initialized all members as

structure