1) In call by value, a copy of the variable is passed, whereas in call by value reference, a variable itself is passed.

In call by value actual & ~~forward~~ formal arguments will be created in different memory locations, whereas in call by ~~value~~ reference, actual & formal arguments will be created in the same memory location.

2) //C Program to multiply two square matrices.

```
#include <stdio.h>
#define N 4
//This function multiplies mat1[][] & mat2[][]
//and stores the result in res[][]
void multiply (int mat1[][N], int mat2[][N],
               int res[][N])
{
int i, j, k;
for (i=0; i<N; i++){
  for (j=0; j<N; j++) {
    res[i][j]=0;
  for (k=0; k<N; k++)
    res[i][j] += mat1[i][k]*mat2[k][j];
  }
```

Scanned with OKEN Scanner

```
    }
  }
  int main()
  {
    int mat1[N][N] = {{1,1,1,1},
                      {2,2,2,2},
                      {3,3,3,3},
                      {4,4,4,4}};
int mat2[N][N] = {{1,1,1,1},
                  {2,2,2,2},
                  {3,3,3,3},
                  {4,4,4,4}};
  int res[N][N];
  int i,j;
  multiply (mat1, mat2, res);
  Printf("Result matrix is");
     for(i=0; i<N; i++){
       for(j=0; j<N; j++)
     Printf("%d", res[i][j]);
   }
   return 0;
}
```

Output:

```
Result Matrix is
10  10  10  10
20  20  20  20
30  30  30  30
40  40  40  40
```

3) //C Program to print Fibonacci series program using recursive methods.

```
#include <stdio.h>
#include <conio.h>
int fibonacci(int);
int main(){
    int n,i;
    Printf("Enter the number of element you want in
                series : n");
    scanf("%d",&n);
    Printf("fibonacci series is : n");
    for(i=0;i<n;i++){
     Printf("%d",fibonacci(i));
    }
    getch();
}
int fibonacci(int i){
    if(i==0)return 0;
    else if(i==1)return 1;
    else return(fibonacci(i-1)+fibonacci(i-2));
}
```

4) String handling functions in C:

String function	Syntax	Description
strlen( )	strl(string-name)	it returns the length of given string
strrev( )	strrev(string-name)	reverses the given string & returns to main function.
strcpy( )	strcpy(string1, string2)	the character in string2 copied in the string 1.
strncpy( )	strncpy(string1, 2, 5)	the first n characters of string 2 are copied in the string 1.
Strcat( )	strcat()(string1, string2)	used to append The Characters of string 2 into string 1.
strlwr( )	strlwr(string 1)	all characters in string1 are converted into lower case characters.

5) C program for the given set of strings

strlen()

```
#include <stdio.h>
#include <string.h>
int main(){
        char s[100] = "JavaPoint";
    //Printing the length of the string
Printf("The length of the given string is: %d", strlen(s));
    return 0;
}
```

strrev()

```
#include <stdio.h>
#include <string.h>
 int main(){
char s[100] = Javapoint";
        Printf("The reverse of the given string is: %s", strrev(s));
    return 0;
    }
```

strcpy() strncpy()

```
#include <stdio.h>
#include <string.h>
    int main(){
char s[50] = a[50];
printf("Enter the second string: \n");
  scanf("%s, a);
printf("The first string is: %s\n", strncpy(s,a,3));
 printf("The first string is: %s", strcpy(s,a));
return 0;
}
```

6) Function

A function is a unit of codes that is often defined by its role within a greater code structure, function contains a unit of code that works on various inputs, many of which are variables, and produces concrete results involving changes to variable values (or) actual operations based on the inputs.

Structure of a user defined function:

1. Function declaration.
2. Function definition.
3. Function call.

Function with no Argument and No returning value in this method, we dont pass any arguments & mention void as a function return type, so a function will not return any value. we declare a few variables in the function definition & Perform certain operations on them.

7)
```
#include <stdio.h>
    struct std{
        int m1,m2,m3;
    };
     int main()
{
struct std s[10];
  for(int i=0; i<3;i++){
Printf("Enter the marks of student %d :", +1);
  scanf("%d", &s[i],m1);
  scanf("%d", &S[i] m2);
```

Scanned with OKEN Scanner

```
    scanf("%d", &s[i].m3);
  }
  for(int i=0; i<3; i++){
    printf("\n AVG \n student %d = %.2f", i+1, (s[i].m1+s[i].
                                          m2+s[i].m3)/3.0);
  }
}
```

Scanned with OKEN Scanner

8) Self-Referential structures are those structures that have one (or) more pointers which point to the same type of structure, as their member in other words, structures pointing to the same type of structures are self-referential in nature.

Example:

```
struct node{
    int data1;
    int data2;
    struct node* link;
};
int main()
{
struct node ob;

return0;
}
```

Scanned with OKEN Scanner

Nesting structure have the feature of nesting one structure within another structure by using which, Complex data types are created.

Example:

```
#include <stdio.h>
  struct address
  {
    char city[20];
    int pin;
    char phone[14];
  };
  void main()
  {
    struct employee emp;
    printf("Enter employer information ? \n");
    scanf("%s %s %d %s", emp, name, emp.add.city, &emp.add.pin,
                                    emp.add.phone);
    printf("Printing the employee information: \n");
    printf("name: %s \n city: %s \n Pincode: %d \n Phone: %s",
            emp.name, emp.add.city, emp.add.pin, emp.add phone);
  }
```

9) There are following functions:

* malloc: it is used to allocate specified no. of bytes.

Syntax: ptr = (cast-type*) malloc (byte-size)

Ex: ptr = (int*) malloc (100* sizeof (int));

* Calloc: it is used to allocate specified no of bytes & initialize all memory with zero.

Syntax: ptr = (cast-type*) calloc (n, element-size);

Ex: ptr = (float*) calloc (25, sizeof (float));

* realloc: - it is used to reallocate the dynamically allocated memory to increase (or) decrease amount of the memory.

Syntax: ~~ptr = (cast-type*) calloc~~ ptr = realloc (ptr, newsize);

* free :- it is used to release dynamically allocated memory.

Syntax: free (ptr);

10) Storage Classes:

We use the storage class in 'C-language for determining the visibility, lifetime, initial values, & memory location of any given variable. The storage classes define the visibility (scope) & lifetime of any function/variable within a C-group. These classes precede the type that they are going to modify.

Types:

* Automatic storage class
* Register
* External
* Static

11) #include <stdio.h>

```
struct book{
    int acc_num;
    char auth[30];
    char title[50];
    int year;
    int price;
};
int main()
{
  Struct book b1={1234,"Joseph Murphy","Subconsious Mind",200};
  Printf("ACCESS NUMBER: %d\n", b1.acc_num);
  Printf("BOOK NAME: %s\n", b1.title);
  Printf("YEAR OF PUBLICATION: %d\n", b1.price);
}
```

12) The command line arguments are one of the ways to pass input to programs.

Command line argument is a parameter supplied to the program when it is involved (or) run. They are used when we used to control our program from outside, instead of hard-coding it. they works installation of programs easier, Command line arguments are passed to the main() method.

Ex: int main (int argc, char * argv[])

13) Pointers in C are used to store the address of variables (or) a memory location. This variables can be of any data. ie; int, char, function, array (or) any other pointer. The size of pointer depends on the architecture.

The pointer arithmetic is performed relative to the base type of the pointer. For example, if we have an integer pointer "ip" which contains address "1000", then on incrementing it by '1' we will get "1004" instead of '1001'. Because, the size of the int data type is 4 bytes, if we had been using a system where the size of int is 2 bytes then we could get 1002.

Ex: * int i=12; * ip=&i;
* double d=2.3, * dp=&d;
* char ch='a'; * cp=&ch;

14) A 'C'-file is a source code file for a C (or) C++ program, it may include an entire program's source code, (or) may be one of many source files referenced within a programming project. C files can be edited using a basic text editor, but will not show syntax highlighting like most software development programs do.

Binary file is opened for both appending & readi
If the file is not existing, then a new file is created. If t
file exists then, old content gets erased & current conta
will be stored. if the file is not existing, then open
function returns NULL VALUE.

15) READING FROM FILE

```
#include <stdio.h>
Int main()
FILE *f;
int t;
f = fopen("C:\\Users\\balar\\Desktop\\CNotes.txt","r");
if (f==NULL){
  Printf("ERROR IN OPENING FILE");
}
else{
fscanf(f,"%d",&t);
Printf("%d",t);
}
}
```

WRITING INTO FILE

```
#include <stdio.h>
int main(){
  FILE *f;
  int t;
  f = fopen("C:\\Users\\balar\\Desktop\\C.txt","w");
if (f==NULL){
  Printf("ERROR IN OPENING FILE");
}
else{
  fPrintf(f,"HELLO");
}
}
```

16) fscanf()

The C library function int fscanf(FILE *stream, const char *format, ...) reads for created input from a stream.

fgets()

Reads a line from the specified stream & stores it into the string pointed to by str.2.

fPrintf()

Sends output that is formatted to a stream. it is almost similar to normal printf() function except in the fact that it writes data into the file.

fwrite()

Writes data from the array pointed to by ptr to the given stream.

17)
```
#include <stdio.h>
#include <stdlib.h>
int main(){
  FILE * sourceFile;
  FILE * destFile;
  Char SourcePath[100];
  char destPath[100];
  Char ch;
Printf("Enter the Source file path:");
 Scanf("%s", Source Path);
 Printf("Enter destination file Path:");
 Scanf("%s", dest path);
```

Scanned with OKEN Scanner

```
Source file = fopen (source path, "r");
dest file = fopen (dest Path, "w");
if (SourceFile == NULL || dest File == NULL)
Printf ("\n Unable to open file.\n");
Printf ("Please check if file exists and you have read/write
                                    privilage.\n");
    exit (EXIT_FAILURE);
  }
ch = fgetc (SourceFile);
while (ch != EOF)
{
   fputc (ch, destFile);
      ch = fgetc (SourceFile);
  }
   Printf ("\n Files copied Successfully.\n");
     fclose (Source File);
     fclose (dest File);
   return 0;
}
```

18) File handling functions:

most often program are executed on terminal but in industries, the application runs should have some part proof (or) records to be referred at some point in time.

* fopen - Opening of an existing file.
* fscanf (or) fgetc - Reading from a file.
* fprintf (or) fputs - writing to file
* fclose - closing of a file.

Syntax

Reading from a file

```
filepointer = fopen("file.txt", "r");
fscanf(filepointer, "%s %s %s %d", str1, str2, str3, &date);
```

Writing a file

```
filepointer = fopen("file.txt", "w");
fprintf(filepointer, "%s %s %s %d", "We", "live", "in", 2020);
```

Closing a file

```
file pointer = fopen("file.txt", "w");
    #Perform some file operations and then close it
    fclose.
```

Scanned with OKEN Scanner