Assignment 3

(1) Explain different types of Parameters passing mechanism techniques with suitable examples.

Ans:- There are two types of parameters passing mechanism :-

- Call by value
- Call by refrence / address

Call by value

In Call by value parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called functions. The changes made on the formal parameters does not effect the values of actual parameters.

eg:

```
#include <stdio.h>
#include <conio.h>
void main() {
 int num1, num2;
 void swap(int, int);
   num1 = 10;
   num2 = 20;
printf("\n Before swap: num 1 = %d, num2 = %d", num1, num2);

 swap(num1, num2);
 printf("\n After swap: num1 = %d \n num 2 = %d", num1, num2);
    getch();
}
 void swap(int a, int b)
 {
    int temp;
    temp = a;
    a = b;
    b = temp;
 }
```

Output:

Before swap : num 1 = 10,
num 2 = 20

After swap : num 1 = 10, num2 = 20

Call by Refrence

In Call by Refrence parameter passing method, the memory location address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be pointer variables. Here, the changes made on the formal parameters effects the values of actual parameters

eg:-

```
# include <stdio.h>
# include <conio.h>
  void swap(int *, int *)
  main()
  {
    int a, b;
  printf("\n Enter any two numbers : ");
  scanf("%d %d", &a, &b);
 printf("\n Before swap a=%d \t b=%d", a, b);
   swap(&a, &b);
 printf("\n After swap a=%d \t b=%d", a, b);
  }
  void swap(int *x, int *y)
  {
    int temp;
    temp = *x;
   *x = *y;
   *y = temp;
  }
```

- Output:

```
Enter any two numbers: 10 20
Before swap a = 10  b = 20
After swap a = 20  b = 10
```

(2) Multiplying matrix of c program.

```
#include<stdio.h>
#define N 50
 int main()
 {
 int a[N][N],b[N][N],c[N][N],i,j,k,sum,m,n,p,q;
Printf("enter rows and columns for first matrix:\n");
scanf("%d %d",&m,&n);
Printf("enter first matrix:\n");
for(i=0;i<m;i++)
 {
 for(j=0;j<n;j++)
  {scanf("%d",&a[i][j]);
  }
 }
 Printf("enter rows & columns for second matrix:\n");
 scanf("%d %d",&p,&q);
 Printf("\nenter second matrix:\n");
 for(i=0;i<p;i++)
 {for(j=0;j<q;j++)
   { scanf("%d",&b[i][j]);
   }
 }
  printf("\n first matrix is:\n");
  for(i=0;i<m;i++)
   {for(j=0;j<n;j++)
      {
        Printf("%d",a[i][j]);
      }
```

```
        printf("\n");
    }
    printf("\n second matrix is :\n");
    for(i=0;i<p;i++)
    {
        for(j=0;j<q;j++)
        {
            printf("%d", b[i][j]);
        }
        printf("\n");
    }
    if (n!=p)
    {
        printf("It can not multiply");
    }
    else
    {
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
            {
                sum=0
                for(k=0;k<m;k++)
                {
                    sum=sum+(a[i][k]*b[k][j]);
                }
                c[i][j]=sum;
            }
        }
        printf("multiplication is :\n");
        for(i=0;i<m;i++)
        {
            for(j=0;j<q;j++)
```

```
{
printf("%d\t", c[i][j]);
}
printf("\n");
}
}
}
```

Assignment 4

Q(1) Explain

Q(3) Write a C programme to implement fibonacci series using recursion.

```
#include <stdio.h>
void fibonacci(int);
int main()
{
int n ;
printf("enter the no. of terms ");
scanf("%d", &n);
printf("\n 0 \n 1 ");
fibonacci(n-2);
return 0;
}
void fibonacci(int n)
{
static int a=0, b=1, c;
for(int i=2; i<
if(n>0)
{
c = a+b;
a=b;
b=c;
printf("\n %d", c);
fibonacci(n-1);
}
```

Another method:

```
#include<stdio.h>
  int fib(int);
  main()
{
   int i, n;
   printf("Enter no. of terms");
   scanf("%d", &n);
   for(i=0; i<n; i++)
  {
     printf("%d\t", fib(i));
  }
}
int fib(int n)
{
  if(n==0)
    return 0;
  else
    if(n==1)
    return 1;
  else
   return(fib(n-1)+fib(n-2));
}
```

Q(4) Describe and explain different built-in handling functions with suitable examples.

Ans :- The different built-in string handling functions are :-

(i) strcpy()

↳ Syntax :- strcpy(string 1, string 2)

It copies string 2 value into string 1.

(ii) strncpy()

↳ Syntax :- strncpy(string 1, string 2, 5)

It copies first 5 characters string 2 into string 1

(iii) strlen()

↳ Syntax :- strlen(string 1)

It returns total number of characters in string 1.

(iv) strcat()

↳ Syntax : strcat(string 1, string 2)

Appends string 2 to string 1.

(v) strncat()

↳ Syntax :- strncpy(string 1, string 2, 4)

Appends first 4 characters of string 2 to string 1.

(vi) strcmp()

Syntax : strcmp(string 1, string 2)

It Returns 0 if string 1 and string 2 are the same; less than 0 if string 1 < string 2; greater than 0 if string 1 > string 2.

Q(12) C program to sort strings in alphabetical order.

```
#include <stdio.h>
#include <string.h>
char ch[50];
void
  main()
{
  int i, j, n;
  char str[100];
```

```
int main()
{
  char string[100];
  printf("Enter the string: ");
  scanf("%s", string);
  char temp;
  int i, j;

  int n = strlen(string);
  for(i=0; i<n-1; i++)
  {
    for(j=i+1; j<n; j++)
    {
      IF (string[i] > string[j])
      {
        temp = string[i];
        string[i] = string[j];
        string[j] = temp;
      }
    }
  }
  printf("The sorted string is: %s", string);
  return 0;
}
```

Ex-1

Q(6) What do you mean by a function? Give the structure of user defined function and explain about the arguments & return values.

Ans :- A function is a block of code that performs a specific task and can be called by other parts of a program.

⇒ The structure of user defined function is —

```
return_type function_name (parameter_list)
{

}
```

- 'return_type' is data type of the value returned by the function. If the function does not return value, the return type should be 'void'.
- 'function_name' is a unique identifier chosen by the programmer.
- 'parameter_list' is a list of variables that are passed to the function when it is called. The parameter list can be empty if the function doesn't require any inputs.

Argument value :-

Argument values passed to the function when it is called. They are matched with the parameters in the function definition. The number of arguments passed to the function should match the number of parameters in the function definition.

- Return value:-

Return values is the value returned by the function when it is executed. the return statement is used to return a value from the function. If the return type of the function is void, the function does not return any value.

Eg:-

```
#include <stdio.h>
int add (int, int)
int main ()
{
int a=20, b=30;
int result = add (a,b);
printf ("%d", result);
return 0;
}
int add (int x, int y)
{
return x+y;
}
```

In the above example, the function 'add' takes two arguments of type 'int', 'x' & 'y' are returns their sum as an 'int'. the function is called with the arguments 'a' & 'b'. with the value '30' & '40' in the 'main' function, & the returned value is assigned to the variable 'result'.

```
#include (stdio.h)
 Struct student
 {
  int rollno, tot;
  char name [25];
  int mark [5];
  float Avg;
 };
  Void main ()
{
struct student S[5];
 int i, n, j;
printf ("Enter the number of students:");
 scanf ("y.d", &n);
 printf ("\t*Students Records*\n");
 for (i=0; i<n; i++)
{
printf ("\n Enter student Roll number:");
scanf ("y.d", &S[i].rollno);
printf ("\n Enter student name:");
scanf ("y.s", s[i].name);
printf ("\n Enter subject 3 marks:");
  for (j=0; j<3; j++)
   scanf ("y.d", &S[i].mark[j]);
  }
  // Calculation
   for (i=0; i<n; i++)
  {
    S[i].tot=0;
```

```
for (j=0 ; j<3 ; j++)
 s[i].tot = s[i].tot + s[i].mark[j];
}
 // Average
  for(i = 0 ;i<n; i++)
  {
     s[i].Avg = (s[i].tot)/3
   }
// Display Result
  for(i=0 ; i<n ; i++)
{ printf ("\t *Students Records *\n");
  printf ("\n ---------- =\n");
  printf ("\n Students Roll no. - %d ", s[i].rollno);
  printf ("\n Students Name . - %s ", s[i].name);
  printf ("\n Students Total marks - %d ", s[i].tot);
  printf ("\n Students Avg marks - %f ", s[i].Avg);
}
}
=
```

Q (8) Differentiate between self-referential struct. and nested structure with examples.

Ans :- A self-refrential structure is a structure that contains a pointer to its own type. for eg—

```
Struct node {
  int data;
  Struct node *next;
};
```

In this example, the 'node' structure contains an integer data and a ~~pointer~~ pointer to another 'node' structure. This allows for the creation of linked lists, where each node points to the next node in the list.

On the other hand, a nested structure is a structure that contains another structure as a member. for example :

```
Struct address {
    Char street [50];
    Char city [20];
    Char state [20];
};
Struct person {
    Char name [50];
    Int age;
  Struct address addr;
};
```

In the example, the 'address' structure is nested within the 'person' structure, and it contains members for street, city and state. The 'person' structure uses this nested structure to represent the address of the person.

In Summary, a self-referential structure is a structure that contains a pointer to its own type, and it is used to create complex data structures, like linked lists, while a nested structure is a structure that contains another structure as a member, it is used to group related data together and make the code more readable and organized.

Q (b) Explain three dynamic memory allocations functions with suitable examples.

Ans:- The three dynamic memory allocating are :-

1. malloc(): The malloc() function is used to dynamically allocate a block of memory of a specified size. For example, the following code allocates memory for an array of 10 integers:

```
eg - int *p;
     p = (int *) malloc (10 * sizeof (int));
```

2. calloc(): The calloc() function is similar to malloc(), but it initializes the allocated memory to zero. For eg.-

```
int **p;
p = (int **) calloc (4, sizeof(int *));
for(int i = 0; i < 4; i++)
    P[i] = (int *) calloc (4, sizeof(int));
```

3. realloc() : The realloc() function is used to change the size of a previously allocated block of memory. For example, the following code increases the size of the previously allocated array of integers to 15:

```
p = (int *) realloc (p, 15 * sizeof(int));
```

(10) Explain about storage classes?

Ans:- The different storage classes are :-

1) Auto : variables declared within a function block having automatic storage class by default. These variables are created when the function or block is called and are destroyed when the function or block exits.

2) Register : Variables declared with the register storage class are stored in CPU registers, rather than in memory. This can lead to faster access time, but there are fewer registers available than memory locations so the use of register should be used sparingly.

3) Static : Variables declared with the static storage class retain their value between function calls. They are initialized only once and retain their value throughout the program's execution.

4) External : Variables declared with the extern storage class are defined in one source file and can be accessed by other source files. This allows for the sharing of variables between source files.

Q(11) Develop a programme to create a library Catalogue with the following members: access number, authors name, title of the book, year of publication and book price using structures.

```
#include <stdio.h>
struct writer
{
  int number;
  char name[20];
  char title[20];
  int year;
  int price;
}aa;
int main()
{
  printf("Enter access number of the book:");
  scanf("%d", &aa.number);
  printf("Enter name of the author:");
  ~~scanf("%s",~~
    gets(aa.name);
  printf("Enter title of the book:");
  ~~scanf("%d", &aa.title~~
    gets(aa.title);
  printf("Enter year of publication:");
  scanf("%d", &aa.year);
  printf("Enter price of the book:");
  scanf("%d", &aa.price);
```

```
printf("The data of the following member is :\n");
printf("%d\n%s\n%s\n%d\n%d", aa.number, aa.name, aa.time, aa.year,
                             aa.price);
return 0;
}
```

(12) Explain about Command line argument with an example.

Ans:- Command line arguments are input passed to a program when it is run from the command line. In C, they can be accessed using the 'argc' (argument count) and 'argv' (argument vector) variable in the main function.

Example of program that takes in one command line argument and ponts it out.

```
#include <stdio.h>
int main (int argc, char * argv[])
{
  if (argc != 2)
  {
    printf ("Expected 1 argument. Got %d\n", argc-1);
    return 1;
  }
  printf ("You entered : %s \n", argv [1]);
  return 0;
}
```

In this example, 'argc' is the number of arguments passed to the program, including the name of the program itself. 'argv' is an array of strings, where each string is an argument.

Another ex:-

```
#include <stdio.h>
int main (int argc, char * argv[])
{ int i;
  if (argc >= 2)
  { printf ("the argument supplied are: \n");
    for (i=1; i< argc ; i++)
    {
```

```
        printf("%s\t", argv[i]);
    }
  }
else
  { printf("Argument list is empty.\n");
  }
  return 0;
}
```

Output: ./a.out
i) welcome
ii) to
iii) study tonight

Q(13) what is pointer? Explain pointer arithmetic operations with suitable example.

Ans:- A pointer is a variable that stores the memory address of another variable.
or ⇒ A pointer is a special type of variable that holds the address as a data item in it.

Pointer are used to dynamically allocate memory and to manipulate memory directly. Pointer arithmetic is a set of operations that can be performed on pointers.

For examples:
We can increment or decrement a pointer to point to the next previous memory locations respectively.

```
#include <stdio.h>
int main()
{
  int num = 5
  int *ptr = &num;
  ptr++;
  printf("%d %d", *ptr, *ptr);
  return 0;
}
```

In this example, 'ptr' is a pointer to an 'int' variable and '&num' is the memory address of the 'num' variable. The '++' operator increments the pointer to point to the next memory locations. The '*' operator is the dereference operator, which is used to access the value stored at the memory location pointed by the pointer.

Another ex :- We can subtract two pointers and it will give you the number of elements bet^n the two pointers.

```
#include <stdio.h>
int main()
{
int arr[5] = {1,2,3,4,5};
int *ptr1 = &arr[0];
int *ptr2 = &arr[4];
printf("%d", ptr2-ptr1);
return 0;
}
```

It prints 4th value, because its pointing 4th element in array.

It's important to remember that pointer arithematic only works on pointer of the same data type, & the result of pointer arithematic is also a pointer.

Q(14) What is a file? Explain different modes of opening a file.

Ans :- A file is a collection of data that is stored on a computer's hardware drive or other storage device.

In C programming, files can be opened & read or written to using the standard input/output Library (stdio.h).

There are several different modes that a file can be opened in :

(a) 'r' (read-only) :- it is opened an existing file for reading only.

(b) 'w' (write only) :- It is open a new file for writing only. If a file with specified filename currently exists, it will be destroyed and a new file with the same name will be created in its place.

(c) 'a' (append only) :- It is open an existing file for appending (i.e. for adding new information at the end of file). A new file will be created if the file with the specified filename doesn't exist.

(d) 'r+' (read & write) :- It is open an existing file for update (i.e. for both reading & writing).

(e) 'w+' (write & read) :- open a file for both writing and reading. If a file with the specified filename currently exists, it will be destroyed and new file will be created in its space.

(f) 'a+' (append & write) :- open an existing file for both appending and reading. A new file will be created if the file with specified filename doesn't exist.

Q15) Write a programme to demonstrate read and write operations on a file.

Ans:- Reading operations on a file

Ex-1
```
FILE *fp;
fp = fopen ("sample.txt", "w");
IF (fp == NULL)
{
  printf ("\n Unable to open a file");
  exit (0);
}
```

Ex-2
```
#include <stdio.h>
int main()
{
FILE *file;
char filename[] = "example.txt";
char buffer [255];
file = fopen (filename, "r");
IF (file == NULL) {
printf (" Error opening file! \n");
return 1;
}
while (fgets (buffer, 255, file) != NULL) {
  printf ("%s", buffer);
} fclose (file);
  return 0; }
```

Writing operations on a file

```
#include <stdio.h>
int main()
{
  FILE *file;
  file = fopen("example.txt", "w") //opens the file for
                                    writing.
  if(file == NULL)
  {
    printf("Error opening file!\n");
     return 1;
  }
  fprintf(file, "Hello world!\n");
  fclose(file);
  printf("writing operation on the file is succesful.\n");
     return 0;
}
```

(16) Explain about fscanf(), fgets(), fprintf(), and fwrite() functions with suitable examples.

Ans :-

(a) fscanf() :- It is a function in C that reads formatted input from a file stream. It works similarly to the scanf() functions, but reads input from a file rather than the standard input.

Eg —

```
#include <stdio.h>
int main()
{
int x, y;
FILE *fp = fopen("input.txt", "r");
fscanf(fp, "%d %d", &x, &y);
printf("x: %d y = %d \n", x, y);
fclose(fp);
return 0;
}
```

(b) fgets() :- It is a function in C that reads a string from a file stream. It reads a line of text from the file, up to a specified maximum number of characters, & stores it in a character array.

Eg :-

```
#include <stdio.h>
int main()
{
char str[100];
FILE *fp = fopen("input.txt", "r");
fgets(str, 100, fp);
printf("Read: %s \n", str);
fclose(fp);
return 0;
```

c) fprintf() : It is a function that writes formatted output to a file stream. It works similarly to 'printf()' function but writes output to a file rather than the standard output.

E.g :-

```
#include <stdio.h>
int main()
{
  int x = 10, y = 20;
  FILE *fp = fopen("output.txt", "w");
  fprintf(fp, "x: %d, y: %d\n", x, y);
  fclose(fp);
  return 0;
}
```

d) fwrite() : It is a function that writes a specified number of bytes to a file stream. It can be used to write binary data to a file.

Eg :-

```
#include <stdio.h>
int main()
{
  int data[] = {1, 2, 3, 4, 5};
  FILE *fp = fopen("output.bin", "wb");
  fwrite(data, sizeof(int), 5, fp);
  fclose(fp);
  return 0;
}
```

In this, the fwrite() function writes 5 integers (size of(int) * 5 bytes) to the file "output.bin" in binary format.

(17) Write a programme to copy one file contents to another.

```
#include <stdio.h>
{  main()
   Char Source [15], destination [15], ch;
   FILE *fp1, *fp2;
   printf ("\n Enter the an existing file name
                to be opened.");
   Scanf ("%s", source);
   Printf ("\n Enter the file name to which
                the content should be copied");
   Scanf("%s", destination);
```

```
fp1 = fopen (source, "r"); //open source file in read mode
IF (fp1 == NULL)
{
  printf ("\n unable to open");
   exit (0);
}
fp2 = fopen (destination, "w"); //open destination file in write mode
IF (fp2 == NULL)
{
  printf ("\n unable to open");
  exit (0);
}
ch = fgetc (fp1); //get a character from source file
while (ch != EOF) //check that character is not EOF
{
  fputc (ch, fp2); //write that character to a destination file
  ch = fgetc (fp1); // Again, read a character from source file.
}
fclose (fp2); // close the file
}
```

Output: Enter an existing filename to be opened:
master.txt
Enter the file name to which the content should be copied: copy.txt

Q (18) Explain different file handling functions with syntaxes & suitable examples.

⇒ The use of file handling functions to perform operations on files such as opening, reading, writing & closing.

(a) fopen() :- This function is used to open a file and returns a pointer to the file.

Syntax :- FILE *fopen (const char *filename, const char *m

Eg:
```
FILE *fp;
fp = fopen ("test.txt", "r");
```

(b) fclose() :- This function is used to close a file.

Syntax :- `int fclose (FILE *fp);`

Eg - `Fclose (fp);`

(c) fgetc () :- This function is used to read a single character from a file.

Syntax :- `int fgets (FILE *fp);`

Eg -
```
char ch;
ch = fgets (fp);
```

(d) fputs () :- This function is used to write a single character to a file.

Syntax :
`int fputs (int char, FILE *fp);`

Eg - `fputs ('A', fp);`

(e) fread () :- this function is used to read a block of data from a file.

Syntax :
```
size-t fread (void *ptr, size_t size, size-t count,
              FILE *fp);
```

Eg -
```
char buffer [100];
fread (buffer, size of (char), 10, fp);
```

(f) fwrite() :- This function is used to write a block of data to a file.

Syntax :-

size_t fwrite (const void *ptr, size_t size, size_t count, FILE *fp);

Eg- char buffer[100] = "Hello world!";
fwrite (buffer, sizeof(char), 100, fp);

(g) fseek() :- This function is used to move file pointer to a specific position in a file.

Syntax :

int fseek (FILE *fp, long int offset, int origin);

Eg -

fseek (fp, 0, SEEK_END);

(h) ftell() :- This function is used to determine the current position of the file pointer.

Syntax : long int ftell (FILE *fp);

Eg - long int position;
position = ftell (fp);

(i) rewind() :- This function is used to move the file pointer to the begining Syntax :- void rewind (FILE *fp);
Eg :- rewind (fp);

(j) remove() :- This function is used to delete a file.

Syntax :- int remove (const char *filename);

Eg — remove ("test.txt");