CSE programming
ASSIGNMENT - 4

(1) **Call by value** → In this particular passing method, the values of the actual parameters copy into the functions formal parameters. It stores both types of parameters in different memory locations. Thus, if one makes any changes inside the function - it does not show on the caller's actual parameters.

Ex → Swapping the values of the two variables

```
#include<stdio.h>
void swap(int, int); //prototype of the function
int main()
{
  int a=10;
  int b=20;
  printf("Before swapping the values in main
    a=%d , b=%d \n", a,b); // printing the value of
                              a and b in main

  Swap(a,b);
  printf(" After swapping the values in main a=%d,
          b =%d \n", a,b); // The value of actual
    parameters do not change by changing the formal
    parameters in call by value, a=10, b=20 }
  }
  void swap(int a, int b)
  {
   int temp;
   temp=a;
   a=b;
   b=temp;
printf("After swapping values in function a=%d, b=%d\n",
        a,b); // Formal parameters a=20, b=10

}
```

Output :-

Before swapping the value in main a=10, b=20
After swapping the value in function a=20, b=10
After swapping in main a=10, b=20

Call by Reference :- In the case of Call by Reference, when we pass the parameter's location reference/address, it copies and assigns them to the function's local argument.

Thus, both the actual argument and passed parameters refer to one similar location.

Ex →

```
#include<stdio.h>
void change (int *num) {
printf("Before adding value inside function num = %d \n",
        *num);
(*num) += 100;
printf(" After adding value inside function num = %d \n",
        *num);
}
int main()
{
int x = 100;
printf("Before function call x = %d \n", x);
change(&x); // passing reference in function
printf("After function call x = %d \n", x);
return 0;
}
```

Output :-

Before function call x = 100
Before adding value inside function num = 100
After " " " num = 200
After function call x = 200

Q.2) Multiplication of 2 Matrices :-

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 int a[10][10], b[10][10], mul[10][10], r, c, i, j, k;
 system("cls")
 printf("enter the number of row=");
 scanf("%d", &r);
 printf("enter the number of column=");
 scanf("%d", &c);
 printf("enter the first matrix element=\n");
 for(i=0; i<r; i++)
  {
   for(j=0; j<c; j++)
   {
    scanf("%d", &a[i][j]);
   }
  }
 printf("enter the second matrix element=\n");
 for(i=0; i<r; i++)
 {
   for(j=0; j<c; j++)
   {
    scanf("%d", &b[i][j]);
   }
 }
printf("multiply of the matrix=\n");
for(i=0; i<r; i++)
{
   for(j=0; j<c; j++)
   {
    mul[i][i] =0;
   for(k=0; k<c; k++)
    {
     mul[i][j] += a[i][k]*b[k][j];
    }
   }
  }
```

```
// for printing result
for(i=0;i<r;i++)
{
  for(j=0;j<c;j++)
  {
    printf("%d\t", mul[i][j]);
  }
  printf("\n");
}
return 0;
}
```

Q.3) Fibonacci Series using recursion

```
#include<stdio.h>
void printFibonacci(int n) {
static int n1=0, n2=1, n3;
if(n>0) {
  n3 = n1+n2;
  n1 = n2;
  n2 = n3;
  printf("%d ", n3);
  printFibonacci(n-1);
  }
}
int main(){
  int n;
  printf("Enter the number of elements :");
  scanf("%d Fibonacci Series:");
  printf("%d %d ", 0, 1);
  printFibonacci(n-2); // n-2 because 2 numbers are
    already printed
  return 0;
}
```

Output →

Enter the number of elements : 10

0 1 1 2 3 5 8 13 21 34

Q.4) String Handling Functions :-

C programming language provides a set of pre-defined functions called string handling functions to work with string values. The string handling functions are defined in a header file called string.h.

Q. 6) Function :- A function is a unit of code that is often defined by its role within a greater code structure. A function contains a unit of code that works on various inputs, many of which are variables, and produces results involving changes to variable values or actual operations based on the inputs.

User-defined Functions :- These are functions that you use to organize your code in the body of a policy. Once we define a function, we can call it in the same way as the built in action and parser functions.

(1.) Function with no argument and no return value :-
When a function has no argument, it does not recieve any data from the calling function. Similarly, when it does not return a value, the calling function does not recieve any data from the called functions.

(2) Function with argument and no return value :-
When a function has arguments, it recieves any data from the calling function but it returns no value.

(3) Function with no arguments but returns a value :-
There could be occassions where we may need to design functions that may not take any arguments but returns a value to the calling function.
An example of this is getchar function : it has no parameters but it returns an integer and integer type data that represents a character.

(4) Function with arguments and return value :-
Here the function is taking an input argument, and also returns something.

Q.7)

```
#include <stdio.h>
int main()
{
  int n, i ;
  float num[100] , sum = 0.0 , avg;
  printf("Enter the numbers of subjects: ");
  scanf ("%d", &n);

  while (n>100 || n<1) {
    printf ("Error! number should be in range of (1 to 100).
            \n");
    printf("Enter the number again: ");
    scanf ("%d", &n);
  }
  for (i=0; i<n; ++i) {
    printf ("%d. Enter number :", i+1);
    scanf("%f", &num[i]);
    sum+= num[i];
  }
  avg = sum/n
  printf ("Average = %.2f ,avg);
  return 0;
}
```

Output:-

```
Enter the number of subjects : 6
1. Enter number : 45.3
2. Enter number : 67.5
3. Enter number : -45.6
4. Enter number : 20.34
5. Enter number : 33
6. Enter number : 45.6
  Average : 27.69
```

Q.8) Self Referential Structures :-

Self Referential Structures are those structures that have one or more pointers which point to the same type of structures, as their member.

Ex→
```
struct node {
    int data1;
    char data2;
    struct node *link;
};
int main()
{
    struct node ob;
    return 0;
}
```

There are 2 types of self Referential Structures :-

1.) Self Referential structures with single link.
2.) Self Referential structures with Multiple Links.

Nested Structure :-

A structure inside another structure, is called nested structure.

Ex →
```
struct emp {
    int eno;
    char ename[30];
    float sal;
    float da;
    float hra;
    float ca;
} e;
```

(9) Dynamic Memory Allocation :-

In the concept of Dynamic Memory allocation, it enables the C programmer to allocate memory at runtime. In this memory can be increased while executing program. This is used in linked list
dynamic memory allocation in C language is possible by 4 functions of stdlib.h header file.

1) malloc() → allocates single block of requested memory

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

Ex →
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int n, i, *ptr, sum=0;
printf("Enter the number of elements:");
scanf("%d", &n);
ptr = (int*) malloc(n* sizeof(int)); // memory allocated using malloc
if (ptr == NULL)
{
printf("Sorry! Unable to allocate memory");
exit(0);
}
printf("Enter elements of array:");
for (i=0; i<n; ++i)
{
scanf("%d", ptr+i);
sum += *(ptr+i);
}
printf("Sum = %d", sum);
free(ptr);
return 0;
}
```

Output :- Enter elements of array : 3
Enter elements of array : 10
10
10
Sum = 30

calloc() function →

It allocates multiple blocks of requested memory.
It initially initialize all bytes to zero.
Returns NULL if memory is not sufficient.

Syntax :- ptr = (cast-type *) calloc(number, byte-size)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i, *ptr, sum=0;
    printf("Enter the number of elements : ");
    scanf("%d", &n);
    ptr = (int*) calloc (n, sizeof(int)); // memory allocated using calloc
    if (ptr == NULL)
    {
        printf("Enter elements of array:");
        printf("Sorry! Unable to allocate memory");
        exit(0);
    }
    printf("Enter element of array:");
    for (i=0; i<n; ++i)
    {
        scanf("%d", ptr+i);
        sum += *(ptr+i);
    }
    printf("Sum = %d", sum);
    free(ptr);
    return 0;
}
```

Output →

```
Enter elements of array : 3
Enter elements of array: 10
10
10
Sum = 30
```

3) realloc() function :-

If memory is not sufficient for malloc() or calloc(); you can reallocate the memory by realloc() function. In short, it changes the memory size.

Syntax → ptr = realloc (ptr, new-size)

Syntax for free function() →

free (ptr).

Q.10) Storage Classes :-

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life time which helps us to trace the existance of a particle variable during the runtime of a program.

C language uses 4 storage classes :-

Storage Specifies	Storage	Initial value	Scope	Life
auto	Stack	Garbage	Within	End of block
extern	Data Segment	Zero	Global Multiple files	Till end of program
Static	Data segment	Zero	Within block	Till end of program
Register	CPU Register	Garbage	Within block	End of block

Q.11)

```
#include <stdlib.h>
#include <string.h>
#define MAX_Books 10

struct book {
int access_no;
char author[50];
char title[100];
int year;
float price;
}
int main() {
struct book library[MAX_Books];
int i, n;
printf("Enter the no. of books!");
scanf("%d", &n);
for(i=0; i<n; i++) {
printf("Enter details for book %d !\n", i+1);
printf("Access number :");
scanf("%d", &library[i].access_no);
printf("Author:");
scanf("%s", library[i].author);
printf("title!");
scanf("%s", library[i].title);
printf("Year of publication:");
scanf("%d", &library[i].year);
printf("Price!");
scanf("%f", &library[i].price);
}
printf("\n Library Catalogue!\n");
for(i=0; i<n; i++){
printf("Access Number: %s\n", library[i].title);
printf("Author: %s\n", library[i].author);
printf("Title: %s\n", library[i].title);
printf("Year of publication: %d\n", library[i].year);
printf("Price: %.2f\n", library[i].Price);
}
return 0;
}
```

Q. 12) Command Line Argument :-

The argument passed from command line are called command line argument. These argument are handled by main() function.

Command line arguments are given after the name of the program in command-line shell of Operation-System. To pass command line arguments, we typically define main() with two arguments: first argument is the number of command line argument and second is list of command-line argument.

Ex →

```
#include<stdio.h>
void main(int argc,char *argv[]){
  printf("Program name is : %s\n", argv[0]);
  if (argc<2){
    printf("No argument passed through command line.\n");
  }
  else{
  printf("First argument is : %s\n", argv[1]");
  }
}
```

Q. 13) Pointer :- A pointer is a variable that stores the memory address of another variable as its value. A pointer variable points to a data type (like int) of the same type, and is created with the * operator.

There are only five operations that are allowed to perform on Pointer in C language. The operation are slightly different from the ones that we generally use for mathematical calculation.

The operations are :-

1) Increment / Decrement of a Pointer →

Increment → when a pointer is incremented, it actually increments by the number equal to the size of the data type for which it is a pointer.

This condition comes under addition.

Decrement → also

This condition also comes under substraction. When a pointer is decremented, it actually decrements by the number equal to the size of the data type for which it is a pointer.

Ex:- // include <stdio.h>
// pointer increment and decrement
// pointers are incremented and decremented by the size of the data type they point to

```
int main()
{
 int a = 22;
 int *p = &a;
 printf("p = %u\n", p); // p = 6422288
 p++;
 printf("p++ = %u\n", p); // p++ = 6422292 + 4  // 4 bytes
 p--;
 printf("p-- = %u\n", p); // p-- = 6422288  - 4 // restored to original value

 float b = 22.22;
 float *q = &b;
 printf("q = %u\n", q); // q = 6422284
 q++;
 printf("q++ = %u\n", q); // q++ = 6422288  + 4 // 4 bytes
 q--;
 printf("q-- = %u\n", q); // q-- = 6422284  - 4 // restored to original value

 char c = 'a';
 char *r = &c;
 printf("r = %u\n", r); // r = 6422283
 r++;
 printf("r++ = %u\n", r); // r++ = 6422284 + 1  // 1 byte
 r--;
 printf("r-- = %u\n", r); // r-- = 6422283  - 1 // restored to original value

 return 0;
}
```

Q:) Addition of integer to a pointer:-

When a pointer is added with a value, the value is first multiplied by the size of data type and then added to the pointer.

Ex→

```
// C program to illustrate pointer Addition
#include <stdio.h>

// Driver Code
int main()
{
// Integer variable
int N=4;
// Pointer to an integer
int *ptr1, *ptr2;

// Pointer stores the address of N
ptr1= &N;
ptr2= &N;
printf("Pointer ptr2 before Addition");
printf("%p\n", ptr2);

// Addition of 3 to ptr2
ptr2 = ptr2+3;
printf("Pointer ptr2 after Addition:");
printf("%p\n", ptr2);
return 0;
}
```

3.) Substraction of integer to a pointer :-

When a pointer is subtracted with a value, the value is first multiplied by the size of the data type and then subtracted from the pointer.

Ex →
```
#include<stdio.h>
int main()
{
  int N=4;
  int *ptr1, *ptr2;
  ptr1 = &N;
  ptr2 = &N;
  printf("Pointer ptr2 before substraction: ");
  printf("%p\n", ptr2);
  ptr2 = ptr2-3;
  printf(" pointer ptr2 after substraction!");
  printf("%p\n", ptr2);
  return 0;
}
```

4.) Substracting two pointers of the same type :-

The substracting of two pointers is possible only when they have the same data type. They result is generated by calculating the difference between the addresses of the two pointers and calculating how many bits of data it is according to the pointer data type.

Ex →
```
# include<stdio.h>
int main()
{
  int x=6;
  int N=4;
  int *ptr1, *ptr2;
  ptr1 = &N;
  ptr2 = &N;
  printf("ptr 1 = %u, ptr 2 = %u\n", ptr1, ptr2);
  x = ptr1 - ptr2;
  printf(" Substraction of ptr 1"
         "& ptr2 is %d\n", x);
  return 0;
}
```

5.) Comparison of Pointer of the same type :-

Pointers containing addresses ;

Ex→
```
#include <stdio.h>
int main()
{
 int N=5;
 int arr[] = {1,2,3,4,5};
 int *ptr;
 ptr = arr;
 for(int i=0; i<N; i++) {
  printf("%d", ptr[0]);
  ptr++;
 }
}
```

14) ~~What is~~ C file

In C, a file is a collection of data stored on a storage device, such as a hard drive or flash drive. Files can be created, modified, and deleted by the operation system and can be used to store various types of information, such as text, images, videos and audio.

In C programming, files are accessed using the file pointer. The 'fopen()' function is used to open a file and returns a pointer to a 'FILE' structure, which can be used to access the file.

Ex →
```
#include <stdio.h>
int main() {
 FILE *fp;
 char ch;
 fp = fopen("example.txt", "r");
 if (fp == NULL) {
  printf("Error opening file\n");
  return 1;
 }
while ((ch = fgetc(fp)) != EOF) {
 printf("%c", ch);
}
 fclose(fp);
 return 0;
}
```