ASSIGNMENT - 4

Q1. Explain about call by value and call by reference with suitable examples?

Ans. Call by Value:-

This method of passing arguments by value is known as call by value. In this method, the values of actual arguments are copied to the formal parameters of the function. If the arguments are passed by value, the change made in the values of formal parameters inside the called function are not reflected back to the calling function.

```
#include <stdio.h>
// function prototype
void swapx(int x, int y);

// main function
int main()
{
   int a=10, b=20;

// Pass by value
   swapx (a, b);
printf ("a=%d  b=%d\n", a, b);
return 0;
}
   // swap function that swaps two values
void swapx (int x, int y)
{
   int t;
   t=x;
   x=y;
   y=t;
printf("x=%d  y=%d", x,y);
}
```

Output:

x = 20 y = 10

a = 10 b = 20

Call by Reference (aliasing):

The method of passing arguments by address or reference is also known as Call by address or call by reference. In this method, the address of the actual argument are passed to the formal parameters of the function. If the arguments are passed by reference, the changes made in the values pointed to by the formal parameter in the called function are reflected back to the calling function.

Example:

```
#include <stdio.h>
void swapx (int*, int*);
int main ()
  {
    int a=10, b=20;
    Swapx (&a, &b);
  printf ("a=%d  b=%d\n", a, b);
  return 0;
  }

Void Swapx (int *x, int* y)
  {
    int t;
    t = *x;
    *x = *y;
    *y = t;
  printf ("x=%d  y=%d\n", *x, *y);
}
```

Output:

x=20 y=10

a=20 b=10

Q2 WAP for Multiplication of two matrixes using C.

```
#include <stdio.h>
int main()
{
  int a[100][100], b[100][100], c[100][100], j, i, k, sum, m, n, p, q;

printf("Enter rows and column 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 row & column 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 cannot be multiplied");
  }
   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");  } } }
```

Q3. WAP to implement fibonacci series using recursion in C.

```
#include<stdio.h>
int fibonacci (int);
int main ()
{
  int i, n;
printf ("Enter the number of element to be in the series");
scanf ("%d", &n);
for(i=0; i<n; i++)
  {
   printf ("%d\t", fibonacci(i));
  }
}
  int fibonacci (int n)
  {
    if (n==0)
      {
        return 0;
      }
    else if (n==1)
      {
        return 1;
      }
    Else
        return (fibonacci (n-1) + fibonacci (n-2));
  }
```

Q 4. Explain about string handling functions?

Ans Strings are defined as an array of characters. The difference between a character array & a string is the string is terminated with a special character '\0'.

String handling functions are explained below:-

- Strcmpi (str1, str2):
  Compares str1 and str2 ignoring the case sensitivity.
- strlwr (str1) : Converts string str1 to all lowercase & return a pointer.
- strupr (str1) : Converts string str1 to all uppercase & return a pointer.
- memcopy (dest, src, n):
  Copies a block of n bytes from src to dest
- memcmp (str1, str2, n):
  Compares first n bytes of strings str1 & str2.
- Strchr (str1, ch):
  Scans the string str1 for the first occurrence of character ch.
- Strset (str1, ch):
  Sets all characters in the string str1 to the character ch
- Strrev (str1): Reverse all character in string str1.

Q5. WAP to sort of the given set of strings in C.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[100], chTemp;
    int i, j, len;
printf("Enter any string:");
gets(str);
len = strlen(str);
for(i=0; i<len; i++)
  {
    for(j=0; j<(len-1); j++)
      {
        if(str[j] > str[j+1])
          {
            chTemp = str[j];
            str[j] = str[j+1];
            str[j+1] = chTemp;
          }
      }
  }
printf("\n Same String in ascending Order: \n%s", str);
return 0;
}
```

Qb: What do you mean by a function? Give the structure of user defined function & explain about the arguments & return values.

Ans: A function is defined as a group of statements that performs a specific task & is relatively independent of the remaining code. Function are used to organize programs into smaller & independent units.

Structure of User-Defined function is

Syntax:

```
return type function name( datatype var1, datatype var2,...)
{
 statement 1;
 statment 2;
 statment n;
}
```

The arguments & return values are explained below:-

(1) Function with no argument and no return value:

When a function has no argument. it does not receive any data from the calling function. Similarly, when it does not return a value, the calling function does not receive any data from the called function.

Syntax:

```
Function declaration: void function ();
Function call       : function ();
Function defination : void function ()
                      {
                        statements;
                      }
```

Example:

```
#include <stdio.h>
void value (void);
void main()
{
    value();
}
Void value (void)
{
  float year =1, period = 5; amount = 5000, inrate =0.12;
  float sum;
sum =amount;
while (year <= period)
  {
    Sum= sum+ (1+inrate);
     year = year +1;
  }
  printf ("The total amount is %f:", sum);
}
```

(ii) Function with arguments but no return value:

When a function has arguments, it receives any data from the calling function but it returns no values.

Syntax:

```
void function (int); - function declaration
function (x);        - function call
void function (int x); - function defination
  {
    statement;
  }
```

Example:

```
#include <stdio.h>
void function (int, int [], char []);
int main()
{
  int a = 20;
  int ar[5] = {10, 20, 30, 40, 50};
  char str[30] = "code with C";
  function (a, &ar[0], &str[0]);
  return 0;
}
void function (int a, int *ar, char *str)
{
  int i;
  printf ("value of a is %d\n\n", a);
  for (i=0; i<5; i++)
  {
    printf ("value of ar[%d] is %d\n", i, ar[i]);
  }
  printf ("\nValue of str is %s\n", str);
}
```

Output:

Value of a is 20

Value of ar[0] is 10
Value of ar[1] is 20
Value of ar[2] is 30
Value of ar[3] is 40
Value of ar[4] is 50

Value of str is CodewithC.

iii) Function with no arguments but returns a value:

There could be occasions where we may need to design function 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 & integer type data that represent a character

Syntax:

```
int function();      . function declaration
function();          . function call
int function()       function definition.
{
  Statements;
  return x;
}
```

Example:

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

int sum();
int main()
{
  int num;
  num = sum();
  printf("\n Sum of two given values =%d", num);
  return 0;
}
int sum()
{
  int a=50, b=80, sum;
  sum = sqrt(a) + sqrt(b);
  return sum;
}
```

Output: Sum of two given value = 16.

iv) Function with arguments and return value:

Syntax:

```
int function(int);
function(x);
int function (int x)
{
  statements;
  return x;
}
```

Example:

```
#include <stdio.h>
#include <string.h>
int function [int, int []);
int main()
{
  int i, a=20;
  int arr[5] ={10, 20, 30, 40, 50};
  a = function (a, & arr[0]);
  printf ("value of a is %d\n", a);
for (i=0; i<5; i++)
  {
    printf("value of arr[%d] is %d\n", i, arr[i]);
  }
  return 0;
}
int function (int a int* arr)
{
  int i;
a = a+20;
arr[0] = arr[0]+50;
```

```
arr[1] = arr[1] + 50;
arr[2] = arr[2] + 50;
arr[3] = arr[3] + 50;
arr[4] = arr[4] + 50;
return a;
}
```

Output:

Value of a is 40
value of arr[0] is 60
Value of arr[1] is 70
Value of arr[2] is 80
Value of arr[3] is 90
Value of arr[4] is 100.

Q7. WAP to read, calculate average & print student marks using array of structures.

```
#include <stdio.h>
Struct student
{
  int marks;
} st [10];
void main ()
{
  int i,n;
  float total=0, avgmarks;
printf ("\n Enter the number of students in class (<=10): ");
scanf ("%d", &n);
for (i=0 ; i<n; i++)
{
  printf ("\n Enter student %d marks :", i+1);
  scanf ("%d", &st [i].marks);
}
  for (i=0 ; i<n; i++)
  {
    total = total + st [i].marks;
  }
    avgmarks = total/n;
    printf ("\n Average marks = %.2f", avgmarks );
  for (i=0 ; i<n; i++)
  {
    if (st [i].marks >= avgmarks)
  {
  printf ("\n Student %d marks = %d above average", i+1, st [i].marks);
  } else {
          printf ("\n Student %d marks = %d below average", i+1, st [i].marks);
  } }
```

Q1 Explain three dynamic memory allocation functions with suitable example

Ans. Dynamic memory allocation can be defined as a procedure in which the size of a data structure (like array) is changed during the runtime.
C provides some function to achieve these tasks. There are 4 library functions provided by C defined under <stdlib.h> header file to faciliate dynamic memory allocation in C programming.
They are:

1. malloc()
2. calloc()
3. free()
4. realloc()

C malloc() method:

The 'malloc' or 'memory allocation' method in C is used to dynamically allocate a single large of memory with the specified size. It returns a pointer of type void which can be cast into a pointer of any form. It doesn't initialize memory at execution time so that it has initialized each block with the default garbage value initially.

Syntax:

```
ptr = (cast-type*) malloc (byte-size)
```

For example:,

```
ptr = (int*) malloc (100 * sizeof(int));
```

Since the size of int is 4 byte, this statement will allocate 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory.

Example :-

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int *ptr;
  int n,i;
  printf("Enter number of elements:");
  scanf("%d",&n);
  printf("Entered number of elements: %d\n",n);

  ptr = (int*) malloc (n* sizeof(int));

  if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
  }
  else {
      printf("Memory Successfully allocated using malloc.\n");

  for(i=0; i<n; i++) {
      ptr[i] = i+1;
  }
    printf("The element of the array are:");
    for (i=0; i<n; i++) {
        printf("%d,", ptr[i]);
    }
  }
  return 0;
}
```

Output : Enter number of elements: 5
Memory successfully allocated usingmalloc.
The elements of array are : 1, 2, 3, 4, 5.

(ii) C calloc() method

- "Calloc()" or "Contiguous allocation" method in C is used to dynamically allocated the specified number of blocks of memory of the specified type. It is very much similar to malloc() but has two different points & these are:

* It initializes each block with a default value '0'.
It has two parameters or argument as compare to malloc().

Syntax:

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

here, n is the no. of elements & element-size is the size of each block

for element:

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

This statement allocates contiguous space in memory for 25 elements each with the size of the float.

(iii) C free() method:

"free" method in C is used to dynamically de-allocate the memory. The memory allocated using functions malloc() & calloc() is not de-allocated on their own. Hence, the free() method is used whenever the dynamic memory allocation takes place. It helps to reduce wastage of memory by freeing it.

Syntax:

free(ptr);

Q10. Explain about storage classes.

* Storage classes are used to describe the features of variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, they are explained below:

1. auto:

This is the default storage class for all the variable declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope) of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they are can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variable reside. They are assigned a garbage value by default whenever they are declared.

2. extern :-

Extern storage class simply tells us that the variable is defined elsewhere & not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by

placing the 'extern' keyword before its declaration/definition in any function / block. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. The main purpose of using extern variable is that they can be accessed between two different files which are part of a large program.

3. Static :-

This storage class is used to declare static variables which are popularly used while writing program in C language. Static variables have the property of preserving their value even after they are ~~not~~ out of their scope. Hence, static variable preserve the value of their last use in their scope. So, we can say that they are initialized only once & exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

4. register :-

This storage class declares register variable that have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free registration is available. This makes the use of ~~re~~ register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free registration is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently

in a program are declared with the register keyword this improves the running time of the program. An important & interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

To specify the storage class for a variable, the following syntax is to be followed:

Syntax:-

Storage_class var_data_type var_name;

Q11. Develop a programme to create a library catalogue with the following members: access number, authors name, title of the book, year of publication & book price using structures.

```
#include <stdio.h>
#include <string.h>
#define Size 30

Struct bookdetail
{
   Char name [30];
   Char author [30];
   int publication;
   int page;
   float price;
};
   void output (struct bookdetails v[], int n);
void main ()
{
   Struct bookdetail b [Size];
```

```
int num, i;
printf ("Enter the Numbers of Books:");
scanf ("%d", &num);
printf ("\n");
for (i=0; i<num; i++)
{
  printf ("\t=: Book %d Detail:=\n", i+1);

  Printf ("\n Enter the Title of Book :\n");
  Scanf ("%s", b[i].name);

  Printf ("\n Enter the Author of Book:\n");
  Scanf ("%s", b[i].author);

  Printf ("\n Enter the Year of Publication of Book:\n");
  Scanf ("%d", &b[i].publication);

  Printf ("\n Enter the Access Number of Book:\n");
  Scanf ("%d", &b[i].page);

  Printf ("\n Enter the Price of Book:\n");
  Scanf ("%f", &b[i].price);

}
  Output (b, num);
}
void output (struct bookdetail v[], int n)
{
   int i, t=1;
for (i=0; i<n; i++, t++)
  {
   printf ("\n");
```

```
printf ("Book No. %d\n", t);
printf ("\t\tBook Access number is = %d\n", t, v[i].page);
printf ("\t\tTitle of Book %d is = %s\n", t, v[i].name);
printf ("\t\tBook %d Author is = %s\n", t, v[i].author);
printf ("\t\tBook %d Year of Publication is = %d\n", t, v[i].publication);
printf ("\t\tBook %d Price is = %f\n", t, v[i].price);
printf ("\n");
    }
}
```

Q52. Explain about command line arguments with an example.

Ans. Command line argument using which we can enter values from the command line at the time of execution. Command line argument is a parameter supplied to program when it is invoked or run.

Use of Command Line Arguments in C

- They are used when we need to control our program from outside instead of hard-coding it.
- They make installation of programs easier.

Command line arguments are passed to the main() method.

Syntax:

```
int main(int argc, char *argv[])
```

Here, argc counts the number of arguments on the command

line and argv[] is a pointer array which holds pointers of type char which points to the arguments passed to the program.

Example:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    if (argc < 2)
        printf("No argument supplied. The only argument here
                is %s", argv[0]);

    return 0;
}
```

Q3. What is a Pointer & Explain pointer arithematic operation with suitable examples.

Ans A pointer is a variable that represents the location (rather than the value) of a data item, such as a variable of an array.

Pointer Arithemetic operation with suitable examples are explained bel

1. Increment / Decrement of a Pointer
2. Addition of integers to a pointer
3. Subtraction of integes to a pointer
4. Subtracting two pointers of the same type.
5. Comparison of pointers of the same type.

1. Increment:

It is a condition that comes under adition. 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.

For example:

If an integer pointer that stores address 1000 is incremented, then it will be increment by 4 (size of an int) and the new address it will points to 1004. While if a float type is incremented then it will increment by 4 (size of a float) and the new address will be 1004.

Decrement:-

It is a condition that also comes under subtraction. 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.

For example:

If an integer pointer that stores address 1000 is decremented, then it will decrement by 4 (size of an int) & the new address it will points to 996. While if a float type pointer is decremented then it will decrement by 4 (size of a float) & the new address will be 996.

Example:

```
#include <stdio.h>
int main()
{
 int  a = 22;
 int  *p = &a;
 printf ("p = %u\n", p);      // P = 6422288
  p++;
 printf ("p++ = %u\n", p);    // p++ = 6422292  +4
```

```
p--
printf("p-- = %u\n", p); // p-- = 6422288   -4
float b = 22.22
float *q = &b;
printf("q = %u\n", q);
q++;
printf("q++ = %u\n", q);
q--;
printf("q-- = %u\n", q);

Char c = 'a';
Char *r = &c;
printf("r = %u\n", r);
r++;
printf("r++ = %u\n", r);
r--;
printf("r-- = %u\n", r);
return 0;
}
```

2. Addition :-

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

```
#include <stdio.h>
int main()
{
    int N=4;
    int *ptr1, *ptr2;
```

```
ptr1 = &N;
ptr2 = &N;

printf("Pointer Ptr2 before addition: ");
printf("%p\n", ptr2);

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

* Subtraction of Two pointers

- The subtraction of two pointers is possible only when they have the same data type. The result is generated by calculating the difference between the address of the two pointers and calculating how many bits of data it is according to the pointer data type. The subtraction of two pointers gives the increament between the two pointers.

For example:

Two integer pointers say ptr1 (address: 1000) & ptr2 (address: 1004) are subtracted. The difference between address is 4 bytes. Since the size of int is 4 bytes, therefore the increment between ptr1 and ptr2 is given by (4/4) = 1.

Example:

```
#include <stdio.h>
int main()
{
  int x=6;
  int N=4;
int *ptr1, *ptr2;

  ptr1 = &N;
  ptr2 = &x;
printf ("ptr1 = %u, ptr2 = %u\n", ptr1, ptr2);
  x = ptr1 - ptr2;
  printf ("Subtraction of ptr1" "& ptr2 is %d\n", x);
return 0;
}
```

Comparison of pointers of the same type:

We can compare the two pointers by using the comparison operators in C. We can implement this by using all operation in C >, >=, <, <=, ==, !=. It returns true for the valid condition and returns false for the unsatisfied condition.

Step-1: Initialize the integer values & point these integer values to the pointer.

Step-2: Now, check the condition by using comparison or relational operator on pointer variables

Step-3: Display the output.

Example:

```
#include <stdio.h>
int main()
{
    int num1=5, num2=6, num3=5;
    int *p1 = &num1;
    int *p2 = &num2;
    int *p3 = &num3;

if (*p1 < *p2)
  {
   printf ("\n%d less than %d", *p1, *p2);
  }
 if (*p2 > *p3)
        {
         printf ("\n%d greater than %d", *p2, *p1);
  }
 if (*p3 == *p1)
                {
                 printf ("\nBoth are equal ");
                }
 if (*p3 != *p2)
                {
                 printf ("\nBoth are not equal");
        }
         return 0;
        }
```

Q14. What is a file? Explain different modes of opening a file?

Ans A file is a collection of data that is stored permanently on disk.

The different modes of opening a file are explained below-

① r (read-only) :- open an existing file for reading only.

② w (write-only) :- open a new file for writing only. If a file with specified filename currently exists, it will be destroyed & a new file with the same name will be created in its place.

③ a (append-only) :- 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 does not exist.

④ r+ (read & write) :- Open an existing file for update (i.e for both reading & writing).

⑤ w+ (write & read) :- open a new file for both writing & reading. If a file with the specified filename currently exists, it will be destroyed & a new file will be created in its place.

⑥ a+ (append & read) :- Open an existing file for both appending and reading. A new file will be created if the file with specified filename does not exi

Q16. WAP to demonstrate read and write operation on a file.

```
#include <stdio.h>
#define NULL 0
main()
{
  FILE *fpt;
  fpt = fopen("sample.dat", "rt");
  if (fpt == NULL)
    printf("\nERROR - Cannot open the designated file\n");
  else
    {
      fclose(fpt);
    }
}
```

Q17. WAP to copy one file contents to another.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  FILE *fptr1, *fptr2;
  char filename[100], c;

  printf("Enter the filename to open for reading \n");
  scanf("%s", filename);

  fptr1 = fopen(filename, "r");
  if(fptr1 == NULL)
  {
```

```
printf ("Cannot open file %s \n", filename);
exit(0);
}
printf ("Enter the filename to open for writing");
scanf ("%s", filename);

fptr2 = fopen (filename, "w");
if (fptr2 == NULL)
  {
    printf ("Cannot open file %s \n", filename);
    exit(0);
  }
    c = fgetc (fptr1);
    While (c1 = EOF)
      {
        fputc (c, fptr2)
        c = fgetc (fptr1);
  }
  printf ("\nContents copied to %s", filename);

  fclose (fptr1);
  fclose (fptr2);
  return 0;
  }
```