Assignment - 4 : Shivam Singh - AP22110011[redacted]478

(1) Explain about the Call by value and call by reference with suitable examples.

=> @Call by value :

=> This is the method in which the value of the actual parameters get copied into the formal parameters. We can say that the value of variable is used in the function call in the Call by value method. In this method we can modify the value of actual parameter by the [redacted] formal parameter. In this method different memory allocated for actual and formal parameters since the value of actual parameters is copied to formal parameter.

Eg: Call by value to Swap.

```
#include <stdio.h>
void swap(int x, int y)
{
  int temp = x;
  x=y
  y=temp;
}
int main() {
int x=10;
int y=11;
Printf("value before swap: x=%d, y=%d\n", x,y);
Swap(x,y);
Printf("value after swap: x=%d, y=%d", x,y);
}
```

- ② Call by reference :

=> In this method, the address of the variable is passed into the function call as the actual parameter. The value of the actual parameter can be modified by changing the formal parameter since the address of actual parameter is passed. In this method, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the functions are performed on the value stored at the address of the actual parameters and the modified value get stored at the same address.

Eg : Swapping value of two variable by call by reference.

```
# include<stdio.h>
void swap(int *x, int *y){
  int temp = *x;
  *x = *y;
  *y = temp; }
int main(){
  int x = 10;
  int y = 11;

printf("value before swap: x= %d, y=%d \n", x,y);
swap(&x, &y);
Printf("value after swap: x=%d, y=%d", x, y);
}
```

② Write a C programme for multiplication of two matrices.

:) Here it is:

```
#include <stdio.h>

int main() {
int row1, col1, row2, col2;
Printf("Enter the no. of rows and columns for matrix A:");
Scanf("%d %d", &row1, &col1);
Printf("Enter the num. of rows and columns for matrix B:");
Scanf("%d %d", &row2, &col2);

if (col1 != row2) {
 Printf("Error: The no. of columns in matrix A must be equal to the
        no. of rows in matrix B. \n");
 return 0;
}
int A[row1][col1];
int B[row2][col2];
int C[row1][col2] = {0};

Printf("Enter the values for matrix A(%d x %d):\n", row1, col1);
for(int i=0; i<row1; i++){
for(int j=0; j<col1; j++){
Scanf("%d", &A[i][j]);
}
}

// values for matrix B
```

```
Printf("Enter the values for matrix B(%d * %d):\n", row2, col2);
for(int i=0; i<row2; i++){
for(int j=0; j<col2; j++){
  Scanf("%d", &B[i][j]);
}
}

  // multiply the matrices:

for(int i=0; i<row1; i++){
for(int j=0; j<col2; j++){
for(int k=0; k<col1; k++){
  C[i][j] += A[i][k] * B[k][j];
}
}
}

// Printing of matrix:

Printf("Answer matrix:\n");
for(int i=0; i<row1; i++){
for(int j=0; j<col2; j++){
 printf("%d ", C[i][j]);
}
  printf("\n");
}
 return 0;
}
```

#C

(3) Write a C to implement Fibonacci Series using recursion.

=>
```
#include <stdio.h>
int fibonacci(int);
int main()
{
 int i,n;
printf("Enter the no 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));
}
```

(9) Explain about String handling function:

=> String handling functions are defined in a header file called String.h.

=> String are defined as an array of characters. The diff betn a characters arrcy and a String is that is terminated with a special character.

String handeling function examples are eitce:

=> Str upr(Str1): This convers String to upper case (all) and return a pointer.

=> Str lwr(Str1): This Converts String to lower case and returns pointer.

=> Str cmp(Str1, Str2): This compare 2 String, ignoring the case sensitivity

=> mem cmp(Str1, Str2, n): This compare first n bytes of string 1 Str1 & Str2

=> Str chr(Str1, ch): scans the String 1 for the first occurence of Character ch.

=> Str set(Str1, ch): This set all character in the String Str 1 to the Character ch.

=> Str rev(Str1): This String handling function reverse all characters in String Str1.

(5) Write a C Programmes to Sort the given set of String.

=> Here is the C Short String program:

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
 Char Str[100], Ch Temp;
 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])    // For descending. For ascending >
   {
    Ch Temp = Str[j];
     Str[j] = Str[j+1];
     Str[j+1] = Ch Temp;
   }
  }
}
Printf("\n Sorted String is : %s", Str);
getch();
return 0;
}
```

(5) What do you mean by a function? Give the structure of user defined function and explain about the arguments and return value.

=) A function is a block of code which only runs when it is called. Function are used to perform certain actions, and they are important for reusing code. It is a group of statements that perform a specific task and is relatively independent of the remaining code.

=) Structure of user defined function are:

-) Syntax:

```
return type function name (data type vars, data type vars,..)
{
 Statement 1 ;
 Statement 2;
 Statement 3:
 - - -
 Statement n ;
}
```

=) Arguments are:

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 definition : void function ()
{
Statements;
}

Eg:

```
#include <stdio.h>
 void value(void);
 void main()
 {
  value ();
 }
 void value (void)
 {
  float year=1, period=5, amount=500, inrate=0.12;
  float sum;
  sum=amount;
 while (year<=period)
 {
  sum=sum+(1+inrate)
  year=year+1
 }
 printf ("The total amount is %f", sum);
}
```

(11) Function with argument but no return value:

=> When a function has argument, it receives any data from the calling function but it returns no value.

Eg:

```
#include <stdio.h>
void function (int, int [], char []);
int main ()
{
 int a=20;
 int ar[5] = {10,20,30,40,50};
 char str[30]= " [redacted] Programming in 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("\n value of str is %s \n", str);
}
```

(iii) Function with no arguments but returns a value:

⇒ There could be occassions 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 and integer type data that represent an character.

⇒ 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=64, b=[illegible], Sum;

  Sum= Sqrt(a) + Sqrt(b);
  return Sum;
}
```

Output : Sum of two given value = 12

(iv) Function with arguments and return value :

⇒

= Function with arguments and one return value means both the calling function and called function will receive data from each other. It's like a dual communication.

eg:

```
#include <stdio.h>
int area(int square_side);
int main()
{
 int Savare_area, Savare_side;
 Printf("Enter the side of Savar : ");
 Scanf("%d", &Savare_side);
 Savare_area = are(Savare_side);
 Printf(" Area of Savare = %d ", Savare_area);
 return 0;
}
 int area( int Savare_side)
{
 int Square_area;
 Square_area = Square_side * Square_side;
 return Square_area;
}
```

(*) we also have function with multiple return values.

(7) Write a program to read, calculate average and print Student mark using array structure.

=>
```
#include<stdio.h>
 Struct student
{
int marks; }
St [10];
void main ()
{
int i,n;
float total =0, avg marks;
Printf ("Enter the no. of students in class (<= 10); ");
Scanf ("%d ", &n);
for (i=0; i<n ; i++)
{
 for (i=0 ; i<n; i++)
{
 total = total + St (i) mark ;
}
 avg marks = total /n;
 Printf ("\n Avg marks = %.2f ", avg marks);
 for (i=0; i <n ; i++)
{
 if (St[i] marks >= avg mark)
{
 Printf ("\n Student %d marks =%d above avg ", i+1, St(i), marks)
}
 else{
   Printf ("\n Student %d marks =%d below avg.", i+1, St(i)marks);}
}
}
```

(8) Differentiate betn self-referential structure and nested structure with example.

⇒ Self-referential Structure	Nested Structure.
1) The self-referential structure is a structure that points to the same type of structure.	The nested structure are those in which one structure is a member of other structures.
2) This structure is useful to create dat structure like linked lists, stacks etc.	The members of a nested structure can be accessed by following syntax: i) variable name of outer_structure ii) variable name of nested_structure. iii) data member.

⇒ Example:
Struct node
{
int data;
Struct node* next;
};
	⇒ Example:
	Struct data
	{
	int day, month, year;
	}
	Struct student
	{
	int roll;
	Struct data. birthday;
	};

(9) Explain 3 dynamic memory allocation functions with suitable examples.

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

=> i) Calloc()
ii) realloc()
iii) free()
iv) malloc()

In example way;

i) Calloc():

=> This initializes each blocks with a default value '0'.
=> It has two parameters or arguments as compared to malloc().
=> In this method of allocation, C is used to dynamically allocate the specified number of blocks of memory of the specified type. It is very much similar to malloc() but has two different points.

=> Syntax: Ptr_name = (*cast type) Calloc (No. of blocks, int size);

Eg:

Ptr = (int*) Calloc (5, (0));

Calloc() function example:

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
void main()
{
char *str = NULL;
str = (char *) calloc (20, sizeof (char));
strcpy (str, "welcome to world");
printf("string is : %s\n", str);
free (str);
}
```

(11) malloc() function:

=> Used to allocate a contiguous block of memory in bytes. It return 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 initialize memory at execution time so that it has initialized each block with default garbage value.

=> Syntax:

```
ptr_name = (*cast type) malloc (int size);
```

Example:

```
char (*a)[20];
A = (char*) malloc (10* 20* sizeof (char));
```

⇒ malloc ( ) function example :

```
#include <stdio.h>
#include<alloc.h>
#include <process.h>
 void main ()
 {
Char * Str;
if((Str=(char*) malloc (10))== NULL){
Printf ("Not enough memory to allocate buffer\n");
exit(1);}
Str Cpy (Str, "Hello world");
Printf (" String is %s \n", Str);
free (Str);
}
```

(iii) Free ( ) function

⇒ Used to free (release or deallocate) the block of unused or already used memory. The memory allocated using functions malloc() or calloc ( ) is not de allocated on their own. Hence the free ( ) method is used . This helps to reduce wastage of memory by freeing it

⇒ Syntax : free (pointer_ variable);

⇒ Example :

```
Ptr = (int*) malloc (10);
free (ptr);
```

(10) Explain about Storage classes.

=> A Storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable. A Storage class in C is used to describe the following things:

=> The variable Scope
=> The location where the variable will be stored.
=> The initialized value of a variable.
=> A lifetime of a variable.
=> Who can access a variable.

=>=> There are total 4 type of Standard Storage classes.

1) Auto Storage class in C:

=> The variable defined using auto Storage class are called as local variable. Auto Stands for Automatic Storage class. A variable is in auto Storage class by default if it not explicitly specified.
The Scope of an auto variable is limited with the particular block only. Once the control goes out of the block, the access is destroyed. This means only the block in which the auto variable is declared can access it.

Eg: auto int age;

Eg1: To define two local variable.

```
int add(void) {
 int a=13;
 auto int b=48;
 return a+b; }
```

(ii) Extern Storage Class in C:

=> Extern Stands for external Storage Class. Extern Storage Class is used when we have global functions or variable which are shared betn two or more files.

=> keyword extern is used to declaring a global variable or function another file to provide the reference of variable or function which have been already defined in the original file.

=> Example, extern void display();

=>
```
#include <stdio.h>
extern i;
main() {
Printf("value of the external integer is = %d\n", i);
return 0;
```

=>
```
#include <stdio.h>
  i=48;
```

=> Result: value of the external integer is = 48 //

(iii) Static Storage Class in C

=> The Static variables are used within function / file as local Static variables. They can also be used as a global variable.

=> Static local variable is a local variable that retains and Stores its value betn function calls or block and remains visible only to the function or block in which it is defined.

=> Example : Static int count = 10;

=>
```
#include <stdio.h>
void next (void);
Static int counter = 7;
main () {
while ( counter <10) {
 next ();
Counter ++; }
return 0;}
void next (void) {
Static int interation = 13;
iteration ++;
Printf (" iteration = %d and Counter = %d \n", iteration, counter); }
```

(IV) Register Storage Class in C :

=> We can use the register Storage class when we want to store local variables within functions or blocks in CPU registers instead of RAM to have quick access to these variables. Eg: "counter" are a good candidate to be stored in the register.

=> Example => register int age;

=> The variables declared using register Storage class has no default value. These variables are often declared at the beginning of a Program.

=>

```
#include <stdio.h>

main() {
{register int weight ;
 int *ptr = &weight ;/* it produces an error when the
 compilation occurs, we cannot get a memory location when
 declaring with CPU register */ }
 }
```

(11) Develop a program to create a library catalogue with the following members: access number, authors name, title of the books, year of publication and book price using structure.

=>
```
#include <stdio.h>
#include<string.h>
#define [redacted] 30

typedef struct catalogue
{
```

```
    int no, price, year;
    char name[N], title[N];
};
lib_cat;
int main()
{
    lib_cat library[30];
    int n,i;
    printf("Enter no. of books: ");
    scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        printf("Enter access number: ");
        scanf("%d", &library[i].no);
        printf("Enter author's first name: ");
        scanf("%s", library[i].name);
        printf("Enter the title of the book: ");
        scanf("%d", &library[i].title);
        printf("Enter the year of publication: ");
        scanf("%d", &library[i].year);
    }
    printf("**Book's Catalouge**\n");
    for(i=0;i<n;i++)
    {
        printf("\t Book access number: %d\n", library[i].no);
        printf("\t Book author name: %s\n", library[i].name);
        printf("\t Book's title: %s\n", library[i].title);
        printf("\t Book's year of publication: %d\n", library[i].year);
        printf("\t Book's price: $%d\n", library[i].price);
    }
    return 0;
}
```

12 Explain about Command line arguments with an example.

=> Command line arguments are simple parameters that are given on the system's command line, and the values of these arguments are passed on to your program during the program execution

=> Use of command line argument 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 age, 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 (arg < 2)
  printf(" No argument Supplied. The only argument
          here is %s ", argv[0]);

  return 0;
}
```

13 What is a pointer? Explain pointer arithmetic operations with suitable examples.

⇒ A pointer is a variable that stores the memory adress of another variable as its value.

⇒ Pointer arithmetic operations are like:

1) Incrementing a pointer

⇒ Incrementing a pointer increases its value by the number of bytes of its data type. This is very important feature of pointer arithmetic operations which we will use in array traversal using pointers.

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

Decrement of a pointer

⇒ It is just the opposite of the increment operation. 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.

⇒ If the pointer is of type integer and if the pointer stores the address 1000 and if the pointer is decremented, then it will decrement by 2 (size of an int), and the new address that the pointer will point to is 998

⇒ Eg: for increment / program for decrement.

```
#include <stdio.h>
int main ()
{
   int a=10, b=20;
   int *ptr1, *ptr2;
   ptr1 = &a;
   ptr 2 = &b;
   Printf ("Pointer ptr1 before Increment : ");
   Printf ("%p\n", ptr 1);
    ptr 1++;
   Printf (" pointer ptr 1 after Increment :");
   Printf (" %p\n\n", ptr 1);
   Printf (" Pointer ptr 2 before Decrement : ");
   Printf (" %p\n", ptr 2);
   ptr 2--;
   Printf ("pointer ptr 2 after Decrement : ");
   Printf ("%p\n\n", ptr 2);
   return 0;
}
```

(2) Addition: When a pointer is added with a value, the value is first multiplied by the size of data and then added to the pointers.

```
#include <stdio.h>
int main()
{
  int N = 5;
  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;
}
```

(3) Subtraction of two pointers:

→ The substraction of two pointers is possible only when they have the same data type. The result is generated by calculating the difference between the adress of the two pointers and calculating how many bits of data it is according to the points data type.

→ Eg:

```
#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;
}
```

(*) Comparision of pointers of the same type ;

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

=> Eg :

```
#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 > *p1)
{
    Printf("\n %d greater than %d ", *p2, *p1);
}
if (*p3 == *p1)
{
    Printf("\n Both are equal");
}
if(*p3 != *p2)
{
    Printf("\n Both are not equal ");
}
    return 0;
}
//
```

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

=) A file is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So files are used for storing information that can be processed by the programs.

=) The different modes of opening a file are explained as:

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

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

(3) Append-only (a) :- They are in open an existing file for appending (ie 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) Read and write (r+) : Open an existing file for update (ie. for both reading & writing).

(5) Write & read (w+) : 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) Append & read (a+) : Open an existing file for both appending and reading. A new file will be created if the file with specified file name does not exists.

(15) Write the program to demonstrate read and write operations on a file.

=>
```
#include <stdio.h>
#define NULL 0
main ()
{
  FILE *fpt;
```

```
fpt = fopen ("Sample.data", "r");
if (fpt == NULL) {
  printf ("\n Error cannot open the searched file\n")
} else
  {
    fclose (fpt);
  }
}
```

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

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

int main ()
{
  FILE *scr, *dst;
  char ch;

scr = fopen ("Source.txt", "r");
if (scr == NULL)
{
  Printf ("Cannot open Source file\n");
  exit (1);
}

dst = fopen ("destination.txt", "w");
```

```
if (dst == NULL)
{
 Printf ("Cannot open destination file \n");
 fclose(src);
  exit(1);
}

while ((ch = fgetc (scr)) != EOF)
{
   fputc (ch, dst);
}

 Printf (" File Copied Successfully \n");

 fclose (scr);
 fclose (dst);

  return 0;
}
```

(18/16) Explain about fscanf(), fgets(), fprintf() and fwrite(). Explain diff file handling function with syntaxes.

=> File handeling function are a set of standard C library functions that are used to perform various operation on files, such as reading and writing data, formatting data, and manipulating the position of the file pointer.

=> Explaining on file handeling function;

1) fscanf() : (FILE *fp, Constant char *format, ...) :

=> This function is used to read formatted data from a file. The first argument is the file pointer, the second argument is a format string that specifies the format of the data to be read, and the remaining arguments are pointer to variables where the data will be stored.

```
int age;
char name[20];
fscanf (fp, "%s %d", name, &age);
printf("Name: %s, Age: %d \n", name, age);
```

(11) fgets() : (char *str, int n, FILE *fp)

=> This function is used to read a line of text from a file. The function returns the buffer is sucessful or NULL if an error occurs.

```
Char line[100];
fgets(line, sizeof(line), fp);
printf("Line: %s", line);
```

(3) fprintf() : (FILE *fp, const char *format, ...)

=> This function is used to write formatted data to a file. The first is pointer, and is string, remaining arguments are the values to be written. The function returns the number of characters written.

Eg:

```
int age = 25;
Char name [] = "Sam Shuvam";
fprintf(fp, "Name : %s , Age : %d\n", name, age);
```

④ fwrite() (const void *ptr, size_t size, size_t count, FILE *fp);

» This function is used to write a block of data to a file. The first argument is pointer to the buffer containing the data to be written, the second argument is the size of each element to be written, 3rd is no. of element to be written, 4th is file pointer. The function returns the no of elements successfully written.

```
Char buffer [] = "Hello, world!";
fwrite(buffer, sizeof(char), sizeof(buffer), fp);
```

⑪ Develop a program to create a library catalogue with the following members: access number, authors name, title of the books, year of publication and book price using structure.

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