A. Eshatwayi Assignment-4 AP22110011002
Section-P

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

Call by value and call by reference are two different ways in which a function can receive arguments in C.

Call by value:

In this method, the function receives a copy of the argument's value. This mean that any changes made to the argument within the function have no effect on the original value outside of the function

Example:-

```
void increment (int x)
{
x++;
}
int main()
{
   int a=5;
   increment (a);
   printf ("%d", a);
}
```

Output :

5

Call by reference :

In this method, the function receives a pointer to the argument. This means that any changes made to argument within the function will affect the original value outside of the function.

Example :

```
void increment (int * x)
{
  (*x)++;
}
int main ()
{
  int a = 5;
  increment (& a);
  printf ("%d", a);
}
```

Output :

6

2. Write a C program for Multiplication of 2 Matrices.

```
#include <stdio.h>
int main()
{
  int a[10][10], b[10][10], c[10][10], i, j, k, m,
      n, p, q;

  printf ("Enter no. of rows & columns
          of Matrix A");
  scanf ("%d %d", &m, &n);
  printf ("Enter elements of matrix A:");
  for (i=0; i<m; i++)
  {
   for (j=0; j<n; j++)
   {
     scanf ("%d", &a[i][j]);
   }
  }
  printf ("Enter no. of rows & columns of
          Matrix B:");
  scanf ("%d %d", &p, &q);
```

```
printf("Enter the elements of Matrix B:\n");
for (i=0; i<p; i++)
{
    for (j=0; j<q; j++)
    {
        scanf("%d", &b[i][j]);
    }
}
if (n!=p)
{
    printf("No. of columns in Matrix A
            must be equal to No. of rows
            in Matrix B\n");
    return 0;
}
for (i=0; i<m; i++)
{
    for (j=0; j<q; j++)
    {
        c[i][j]=0;
        for (k=0; k<n; k++){
        c[i][j]+=a[i][k] * b[k][j];
    }
```

```
    }
}
printf ("Product of given 2 Matrices:\n");
for (i=0; i<m; i++)
{
    for (j=0; j<q; j++)
    {
        printf ("%d", c[i][j]);
    }
    printf ("\n");
}
return 0;
}
```

3. write a c program to implement Fibonacci series using recursion:

```
#include <stdio.h>
int fibonacci (int n)
{
    if (n <= 0)
    return 0;
    if (n == 1)
    return 1;
```

```
else
return fibonacci(n-1) + fibonacci(n-2);
}
int main()
{
    int n, i;
    printf("Enter no. of terms:");
    scanf("%d", &n);
    printf("Fibonacci series:");
    for(i=0; i<n; i++)
        printf("%d", fibonacci(i));
        return 0;
}
```

4 Explain about string handling functions

C provides a set of standard library functions for handling strings, which are defined in the string.h header file. Some of the commonly used string handling functions in C include:

→ strlen(): This function is used to find the length of a given string

→ strcpy(): This function is used to copy one string to another.

→ strcat(): This function is used to concatenate two strings.

→ strcmp(): This function is used to compare 2 strings. It returns 0 if the strings are equal, a negative value if the first string is lexicographically less than second string, and a positive value if the first string is lexicographically greater than the second string.

→ strchr(): This function is used to search for the first occurance of a given character in a string.

→ strstr(): This function is used to search for the first occurance of a given substring in a string.

There are several other string handling functions in C, such as strcpy(), strncat(), strncmp(), etc.. These functions work similarly to the functions mentioned above, but they accept an additional argument specifying the maximum no. of characters to be used.

5. Write a C program to sort the given set of strings

```
#include <stdio.h>
#include <string.h>
#define MAX_STRINGS 10
#define MAX_LENGTH 50

void sortstrings (char strings[][MAX_LENGTH], int n)
{
  char temp[MAX_LENGTH];
  for(int i=0; i<n-1; i++)
  {
```

```
for (int j=0;j<n;j++)
{
  if (strcmp(strings[i],strings[j])>0)
  {
    strcpy(temp,strings[i]);
    strcpy(strings[i],strings[j]);
    strcpy(strings[j],temp);
  }
}
}
int main()
{
  char strings[MAX_STRINGS][MAX_LENGTH];
  int n;
  printf("Enter the no. of strings:");
  scanf("%d",&n);
  printf("Enter %d strings:\n",n);
  for(int i=0;i<n;i++)
  {
    scanf("%s",strings[i]);
  }
```

```
sortstrings (strings, n);
printf ("sorted strings:\n");
for (int i=0; i<n; i++)
{
    printf ("%s\n", strings[i]);
}
return 0;
}
```

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

In programming, a function is a block of code that performs a specific task. The structure of a user defined function in C language typically includes the following elements:

1) The function declaration, which includes the return type, function name, and the list of parameters (if any)

enclosed in parenthesis.

2) The function body, which contains the statements that are executed when the following function is called.

for example, the following is a simple C function that takes 2 integer arguments and returns the sum of two numbers.

```
int add (int a, int b)
{
    int c = a + b;
    return c;
}
```

Arguments:

In the above example, the variables 'a' & 'b' are the arguments passed to the function. They are used to pass data into the function.

Return value: In the given example, the variable 'c' is the return value of the function. It is used to return a value back to the calling code. The return statement is used

to return the value of the variable 'c' to the calling code.

When the function is called, the values passed as arguments are used to perform the operations defined in the function, and the return value is to pass the results back to the calling code.

7. Write a program to ~~find~~ read, calculate average and print student marks using array of structures.

```
#include <stdio.h>
Struct student
{
    int roll_no;
    char name[20];
    float marks[3];
    float average;
};
int main()
{
```

```
int i,j,n;
struct student s[10];
printf ("Enter the no. of students :");
scanf ("%d", &n);
for (i=0; i<n; i++)
{
    printf ("Enter details for student %d:
                \n", i+1);

    printf ("Roll number :");
    scanf ("%d", &s[i].roll_no);
    printf ("Name :");
    scanf ("%s", s[i].name);
    for (j=0; j<3; j++)
    {
        printf ("Marks in subject %d:",
                                    j+1)

        scanf("%f", &s[i].marks[j]);
    }
}
for (i=0; i<n; i++)
{
    float sum=0;
    for (j=0; j<3; j++)
```

```
    {
        sum += s[i].marks[j];
    }
    s[i].average = sum/3;
}
printf("\n student details : \n");
for (i=0; i<n; i++)
{
    printf("Roll number : %d\n", s[i].
                                roll_no);
    printf("Name : %d\n", s[i].name);
    printf("Average Marks : %.2f\n", s[i].
                                average);
}
return 0;
}
```

8. Differentiate between self-referential structure and nested structure with example.

In C programming, a self-referential structure is a structure that contains a pointer to an instance of the same structure type. It is used to create linked data

structures, such as linked lists and trees.

For Eg:

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

In this example, the 'node' structure contains an integer "data" and a pointer "next" to another instance of the "node" structure. This allows us to create a linked list 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. It is used to group related data together and to create more complex data structures.

For example:

```
struct address {
char street [20];
char city [20];
char state [20];
};
```

```
struct employee {
int id;
char name[20];
struct address addr;
};
```

In this example, the "address" structure contains three character arrays for this street, city and state and the "employee" structure contains an integer 'id', a character array 'name' and a nested address structure 'addr'. This allows us to group the address details of an employee in a seperate structure.

In summary, a self-referential structure is a structure that contains a pointer to an instance of the same structure type, while a nested structure is a structure that contains another structure as a member.

9. Explain the dynamic memory allocation functions with suitable examples.

In C programming, dynamic memory allocations refers to the process of allocating memory at runtime, as opposed to compile time. There are several functions available in the C standard library for allocating dynamic memory including:

1] malloc(): This function is used to allocate a block of memory of a specified size. It takes one argument, which is the size of the memory block in bytes. It returns a pointer to the first byte of the allocated memory block. If the memory allocation is successful, the pointer returned by malloc() points to the first byte of the allocated memory block, otherwise it returns a null pointer.

Example:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
```

```
int n,i;
int *p;
printf("Enter No. of elements: ");
scanf("%d",&n);
p = (int *)malloc(n * sizeof(int));
if (p == NULL)
{
    printf("Memory allocation failed \n");
    return 1;
}
for(i=0;i<n;i++)
{
    printf("Enter element %d: ", i+1);
    scanf("%d", &p[i]);
}
printf("Entered elements are: ");
for (i=0; i<n; i++)
{
    printf("%d ", p[i]);
}
printf("\n");
free(p);
return 0;
}
```

2. (alloc):

This function is used to allocate a block of memory for an array of a specified number of elements, each of a specified size. It takes two arguments, the first argument is the number of elements in the array and the second argument is the size of each element in bytes. It returns a pointer to the first byte of the allocated memory block. If the memory allocation is succesful, the pointer returned by (alloc) points to the first byte of the allocated memory block, otherwise it returns a null pointer.

Eg:-

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n, i;
    int *p;
    printf("Enter No. of elements : ");
    scanf("%d", &n);
    p = (int*) alloc (n, size of (int));
```

```
if (P==NULL)
{
 printf("Memory allocation failed \n");
 return 1;
}
for(i=0;i<n;i++)
{
 printf("Enter element %d: ", i+1);
 scanf("%d", &p[i]);
}
printf("Entered elements are: ");
for(i=0;i<n;i++)
{
 printf("%d", p[i]);
}
 printf("\n");
 free(p);
 return 0;
}
```

3) realloc()

This function is used to change the size of previously allocated memory block. It takes two arguments, the first argument is a pointer to the previously

allocated memory block in bytes. It returns a pointer to the first byte of the re-allocated memory block. If the memory reallocation is succesful, the pointer returned by re-alloc() points to the first byte of the re-allocated memory block.

Eg:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int n,i,new_n;
    int *p;
    printf("Enter the no.of elements:");
    scanf("%d",&n);
    p=(int*)malloc(n*sizeof(int));
    if(p==NULL)
    {
        printf("Memory allocation failed \n");
        return 1;
    }
    for(i=0;i<n;i++)
    {
        printf("Enter element %d:",i+1);
```

```
    scanf("%d", &p[i]);
}
printf("Entered elements are :");
for(i=0; i<n; i++)
{
    printf("%d", p[i]);
}
printf("\n");
printf("Enter the new number of
        elements :");
scanf("%d", &new_no);
p = (int*)realloc(p, new_no * sizeof(int));
if(p==NULL)
{
    printf("Memory allocation failed\n");
    return 1;
}
for(i=n; i<new_no; i++)
{
    printf("All elements are :");
    for(i=0; i<new_no; i++){
        printf("%d", p[i]);
    }
    printf("\n");
    free(p);
    return 0;
}
```

10. explain about storage classes.

In C programming a storage class is way to specify the duration and visibility of variable (or) function. There are 4 storage classes in C.

1. Automatic:

These are local variables that are defined inside a function. They are also called "local variables" & automatic variables. They are automatically created when the function is called and automatically destroyed when the function returns. They donot retain their value between function calls. They are the default storage class for local variables if no storage class is specified.

Eg:

```
void func(){
 int x;
 x=5;
 printf("%d",x);
 }
```

2. Register:

These are local variables that are stored in a register instead of memory. Using a register storage class can improve the performance of the program by reducing memory access time. However, the number of registers is limited, so not all variables can be stored in registers.

Eg:

```
void func(){
register int x;
x=5
printf("%d", x);
}
```

3. Static: These are the variables that retain their value b/w function calls. They are also used to create variables that are only visible within a specific file, rather than being visible throughout the program. A variable defined as static inside a function maintains its value b/w

function calls.

Eg:
```
void func (){
static int x=0;
x++;
printf ("%d", x);
}
```

Extern:
These are variables that are defined in one file and can be accessed in another file. They are used to share variables b/w different files (or) modules in a program. An extern variable can be defined in one source file & used in another source file.

Eg:
```
//file1.c
int x;
x=5;
//file 2.c
extern int x;
printf ("%d", x);
```

In summary, storage class in C specify the duration and visibility of a variable

(a) function in C has 4 types of storage classes: Automatic, Register, Static, and Extern. They are used to control the lifetime and scope of variables and functions.

11. Develop a program to create a library catalogue with the following members: access number, author's name, title of book, year of publication and book price using structures.

```
#include <stdio.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:%d\n",library[i].access_no);
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 : %.2s\n", library[i].price);
}
return 0;
}
```

12. Explain about command line arguments with an example.

Command line argument :

These are given after the name of the program in command-line shell of operating systems. Command line arguments are passed to the main() method.

Syntax :-

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

argc counts the number of arguments on the command line and argv[] is a pointer array, which holds pointers of the type char which points to the

arguments.

Eg:
```
#include<stdio.h>
int main(int argc, char *argv[])
{
int i;
if(argc>=2)
{
printf("the arguments are:");
for(i=1;i<argc;i++){
printf("%s\t",argv[i]);
}
}
else{
printf("argument list is empty");
}
return 0;
}
```

Output:
Argument list is empty.

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

A pointer is a variable that stores the memory address of another variable. Pointers are useful for many tasks in C, including dynamic memory allocation, function pointers and passing arguments to functions by reference.

Pointer arithmetic is the manipulation of pointers to perform various operations like addition, subtraction, increment, decrement on pointers.

In C, pointer arithmetic is performed in the following way:

→ "ptr++": Increases the pointer to point to the next element of the same type.

→ "ptr+n": Adds n to the pointers memory address, moving it n elements ahead of the same type.

→ "ptr--": Decrements the pointer to point the previous element of the same type

"ptr - n" :- Subtracts n from the pointers memory address, moving it ~~or~~ n elements back at the same type

Eg:

```
#include <stdio.h>
int main()
{
    int arr[] = {1,2,3,4,5};
    int *ptr = arr;
    printf("value of first element: %d\n", *ptr);
    ptr++;
    printf("value of second element: %d\n", *ptr);
    ptr = ptr + 2;
    printf("value of third element: %d\n", *ptr);
    ptr = ptr - 1;
    printf("value of fourth element: %d\n",
                                    *ptr);
    return 0;
}
```

Output :-

Value of first element = 1
Value of second element = 2
Value of third element = 4
Value of fourth element = 3

14) What is a file? Explain different modes of operating a file?

In c, a file is a collection of data stored in a storage device such as a hard drive or flash drive. Files can be created and modified and deleted by operating 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

"r" : opens a text file for reading

"w" : opens a text for writing. If the file already exists its content will be truncated.

"a" : opens a text file for writing. The file is created if doesnot exist the file is opened in append mode.

"rb" : opens in binary file for reading.

"wb": opens a binary file for writing. If the file is already exists, its content will be truncated.

"ab": opens a binary file for writing. The file is created if it doesnot exist. The file is opened in append.

example of opening a file in c:

```
#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;
}
```

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

```
#include <stdio.h>
int main()
{
    FILE *fp; // FILE pointer
    fp = fopen("example.txt", "w");
    fprintf(fp, "writing a file in C");
    fclose(fp);
    fp = fopen("example.txt", "r");
    char ch;
    while (ch = fgetc(fp)) != EOF)
    {
        printf("%c", ch);
    }
    fclose(fp);
    return 0;
}
```

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

"fscanf()": This function is used to read formatted input from a file.

```
FILE *fp;
int i;
char str[100];
float f;
fp = fopen ("data.txt", r);
fscanf (fp, "%d %s", &str, &f);
printf ("Read : %d %s %f", i, str, f);
fclose (fp);
```

fgets() : This function is used to read a line of text from a file. It takes input a file pointer, a buffer and stole the read texts, and the max no. of characters to read as arguments.

```
FILE *fp;
char line[100];
fp = fopen ("data.txt", r);
fgets (line, sizeof line, fp);
printf ("Read : %s", line);
fclose (fp);
```

fprintf() : This function is used to write formatted output to a file.

```
FILE * fp;
int i=42
chag str[] = "Hello world";
float f=3.14;
fp = fopen ("data.txt", w);
fprintf (fp, "%d %s", i, str, f);
fclose (fp);
```

fwrite() :-

This function is used to write a binary data to a file.

"data.bin"

```
FILE * fp;
int data[] = {1,2,3,4,5};
fp=fopen ("data.bin", wb)
fwrite [data, size (int), size of (data) (size of int), fp);
fclose (fp);
```

Its important to note that when reading and writing is binary data you should use "rb" and "wb" mode.

17) write a program to copy one file contents to another.

```
#include <stdio.h>
int main()
{
    FILE *source *target;
    source = fopen("source.txt", r);
    is (source == NULL)
    {
        prints("could not open source file\n");
        return 1;
    }
    target = fopen("target.txt", w);
    is (target == NULL)
    {
        prints("couldnot open target file\n");
        fclose(source);
        return 1;
    }
    char ch;
    while ((ch = fgetc(source)) != EOF)
    {
        fputc(ch, target);
    }
```

```
    printf("File copied successfully");
    fclose(source);
    fclose(target);
    return 0;
}
```

18. Explain different file handling functions with syntaxes and suitable examples.

C standard library provides several functions for file handling some of the commonly used functions are :-

1) "fopen" :- (const char *filename, const char *mode);

This function is used to open a file. It takes the name of the file and the mode in which the file should be opened as arguments. The mode can be "r" for reading, "w" for editing, "a" for appending, "r+" for reading and "w+" for writing and reading.

Syntax :-

FILE *fopen(const char *filename, const char *mode);

Example :-

```
FILE *fp;
fp = fopen ("example.txt", "r");
```

"fclose (FILE *fp);

This function is used to close an open file. It takes a file pointer as an argument.

Syntax :-

```
int fclose (FILE * fp);
```

Example :-

```
fclose (fp);
```

"fgetc (FILE *fp) :-

This function is used to read a single character from a file. It takes a file pointer as an argument and returns the character read as an int.

Syntax :-

```
int fgetc (FILE * fp);
```

Example :-

```
int ch;
ch = fgetc (fp);
```

iv) 'fputc(int c, FILE * fp) :-

This function is used to write a single character to a file. It takes an int and a file pointer as arguments.

Syntax :-

int fputc(int c, FILE * fp);

Example :-

fputc ('A', fp);

v) 'fread' :- This function is used to read binary data from a file. It takes a pointer to the buffer, the size of each elements, the no. of elements and a file pointer as arguments.

Example :-

int data[100];

fread [data, size of (int), 100, fp);

Syntax :-

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

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

This function is used to write binary data into a file. It takes a pointer to the data to a file. It takes a pointer to data, the size of each element, the number of elements, and a file pointer as the arguments.

Syntax :-

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

Example :-

```
int data[100] = {1,2,3,4,5};
fwrite (data, sizeof (int), 100, fp);
```

vii) 'fprintf' (FILE *fp, const char *format, ...);

This function is used to write formatted output to a file. It takes a file pointer, a format string, and a variable number of arguments.

Syntax :-

```
int fprintf (FILE *fp, const char *format
             ...);
```

Example :

```
FILE *fp;
int i=42;
float f=3.14;
char str[] = "Hello World";
fp = fopen ("example.txt", "w");
fprintf (fp, "Integer : %d, float : %f,
                string : %s", i, f, str);
fclose (fp);
```