CSE

Assignment - 4

AP22110011019
N. Sujitha

1. Explain about call by value and call by reference with the 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 means 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 the 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 elements of Matrix B:");
    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<m; 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 the Fibonacci series using the 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 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 the strings, which are defined in the string.h headerfile some of the commonly used string handling functions in c include

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

strcpy() :- This function is used to copy one string to the another

strcat() :- This function is used to concatenate the two strings

strcmp() :- This function is used to compare the two 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 sub-string in a string.

There are several other string handling functions in C, such as strncpy(), strncat(), strncmp(), etc. These functions works similarly to the function mentioned above, but they accept an additional argument specifying the maximum no-of the 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-LENGHT 50
void sortstrings(char strings[][MAX LENGTH],int n)
{
    char temp[MAX LENGHT];
    for(int i=0; i<n-1; i++)
    {
        for(int j=0; j<n ;j++)
        {
            if(strcmp(string[i], strings[j]>0)
            {
                strcpy(temp, strings[i]);
                strcpy(strings[i], shings[j]);
                strcpy(strings[j],temp);
            }
        }
    }
}
int main()
{
    char strings[MAX_STRINGS][MAX_LENGTH];
```

```
int n; scanf("%d", &n);
printf("Enter the no.of strings : \n", n);
for (int i=0; i<n ; i++)
{
   scanf("%s", strings[i]);
}
sort strings (string, 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 the 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 include the following elements

1. The function declaration, which includes the return type, function name, and the list of parameters (if any) enclosed in the parameters
2. The function body, which contains the statements that are executed when the function is called

For example, the following is a simple c function that takes 2 integer arguments and returns the sum of the 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 values:- In the given value 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 is to pass the result back to the calling code.

7. Write a c program to read, calculate the average and print the 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++)
    {
        piintf("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<m3; j++)
        {
            printf("Marks in subject %d:", j+1)
            printf
            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 the 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 the linked data structure such as the linked lists and trees for example

Ex:-

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

In this example, the 'node' structure contains an integer "data" and a pointer "next" to the 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 the 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 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 three dynamic memory allocation function with the suitable examples

In c programming, dynamic memory allocation refers to the process of allocating the memory at runtime, as opposed to the compile time. There are several functions available in the c standard library for allocating the 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 memory allocation is successful.

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 elements %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. calloc() :- This function is used to allocate a block of memory for an array of a specified number of element, 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 elements in bytes

Example :-

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

3. realloc() :- This function is used to change the size of the previously allocated memory block. It takes two arguments the first argument is a pointer to the previously allocated memory block, and the second argument is the new size of the memory block in bytes. In returns a pointer to the first byte of the re-allocated memory

Example :-

```
#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;j<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_n*sizeof(int));
if(p=NULL)
{
    printf("Memory allocation failed\n");
    return 1;
}
for(i=n;i<new_n;i++)
{
    printf("Enter new element %d:",i+1);
    scanf("%d", &p[i]);
}
printf("All elements are:");
for(i=0;i<new_n;i++)
{
    printf("%d", p[i]);
}
printf("\n");
free(p);
return 0;
}
```

10. Write about storage classes.

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

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

Example :-

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

2. Register :- They are local variables they are stored in a register instead of the memory. using a register storage class can improve the performance of the following by reducing memory access time. However the number of register is to be limited, so not all variables can be stored in registers

Example :-

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

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

Example :.

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

4. Extern :- These are variables that are defined in one file and can be accessed in the another file. They are used to share the variables between the different files (or) modules in a program An extern variables can be defined in one source file and used in another source file.

Example :-

In this

```
// file1.c
int x;
x=5;
// file2.c
extern int x;
printf("%d", x);
```

In summary, storage class in c specify the duration and visibility of a variable (or) function in c has 4 types of storage classes: Automatic, Register, static, extern. They are used to control the lifetime and scope of variables and functions

11) Develop a programme to Create a library catalogue with the following members. access number, authors, name, title of book, year of publication and book price using structures

```
#include<stdio.h>
#include<string.h>
#define MAX_BOOKS 10
struct book
{
    struct
    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 books %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("%d", &
        printf("year of publication:");
        scanf("%d", &library[i].price);
    }
    printf["In library catalogue \n");
    for (i=0; i<n; i++)
    {
        printf("Enter details for book %d:\n", i+1);
        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: %2f\n", library[i].price);
    }
    return 0;
}
```

12. Explain about Command line arguments with an example.

In c programming, command line arguments are parameters passed to a program when it is executed from the command line. These arguments can be used to provide input to the program or to specify options for how the program should run command line arguments are passed to the main() function of a c program and are received by the program in form of an array of strings. The first element of this array (arg[0]) contains the name of the program, and the remaining elements contain contain the arguments passed to the program. The number of arguments passed to the program is passed as the second argument to the main function

Here is an example of a program that takes 2 command line arguments, an output file name and output file name, & copies the content of input file to the output file

Example:-

```
#include<stdio.h>
int main (int argc, char *argv[])
{
    FILE *in, *out;
```

```
int c;
if (argc != 3)
{
    printf("usage: %s input_file output_file\n", argv[0]);
    return 1;
}
in = fopen(argv[1], "r");
if (in == NULL)
{
    printf("Error: unable to open input file %s\n", argv[1]);
    return 1;
}
out = fopen(argv[2], "w");
if (out == NULL)
{
    printf("Error: unable to open output file %s\n", argv[2]);
    fclose(in);
    return 1;
}
while ((c = fgetc(in)) != EOF)
fputc(
```

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 function by reference

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

In C, pointer arithmetic is performed in the following ways

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

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

"ptr+n" :- Adds n to the pointers memory address, moving it n elements a head of the same type

"ptr-n" :- Subtracts n from the pointers memory address, moving it n elements back of the same type.

Example :-

```
#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 which can be used to access the file

The "fopen()" function takes two parameters the name of the file and the mode in which the file should be opened. The mode in which a file is opened determines what operations can be performed on file d different modes of opening a file in c are:

"r" : opens a text file for reading

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

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

"wb": opens a binary file for working. If the file is already exists, its contents 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 programme 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 examples

'fscanf():- This function is used to read formatted input from a file It works similarly to the scanf() function, but it takes an additional file pointer as first argument. For example. the following code reacts an integer, a string and a float from file called 'data txt'

```
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 store the read text, and the maximum number of characters to read as arguments For example, the following code reacts a line of text form of a file called "data txt" and prints it to the console

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

fprintf():- This function is used to write formatted output to a file. It works similarly to the printf(). function, But it takes an additional file pointer as the first argument for example, the following code writes an integer, a string, and a float to a file called "data txt"

```
FILE *fp;
int i=42
char 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 binary data to a file It takes a pointer to the data, the size of each element, the number of elements, and a file pointer as arguments for example, the following code writes an array of integers to a file called

"data.bin":

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

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

17) write a programme to copy one file contents to another

```
#include<stdio.h>
int main()
{
    FILE *source, *target;
    source = fopen("source.txt", r);
    if (source == NULL)
    {
        printf("could not open source file \n");
        return 1;
    }
    target = fopen("target.txt", w);
    if (target == NULL)
    {
        printf("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 writing, "a" for appending, "r+" for reading and writing, and "w+" for writing & reading.

Syntax:-

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

Example:-

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

(ii) "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);
```

(iii) "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(intc, 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(intc, 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 element, the no.of elements and a file pointer as arguments

Example:-

```
int data[100];
fread [data, sizeof(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, size of (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);
```