CSE ASSIGNMENT-4

K. CHARISHMA
AP22110010997
CSE-P

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

Ans: 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.

Ex:
```
void increment (int n)
{
n++;
}
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 mack to argument within the function will affect the original value outside of the function.

En:
```
void increment (int *n)
{
(*n)++;
}
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 and 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 and 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<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 cprogram to implement fibonacci series using recursion.

Program:

```
#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

Ans: 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 ~~funct~~ 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 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 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 similar to the functions metioned above, but they accept an additional argument specifying the maximum no. of characters to be used.

5. Write a cprogram to sort the given set of strings.

program:

```
#include<stdio.h>
#include<string.h>
#define MAX_STRINGS 10
#defined MAX_LENGTH 50
void sortstring(char string[][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(string[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]);
}
 Sort strings(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 and return values.

Ans: Function: It 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. function declaration: It includes the return type, function name and the list of parameters (if any) enclosed in paranthesis.
2. function body: Contains statements that are executed when the function is called.

Eg: The following is a simple c function that takes 2 integer arguments and returns the sum of 2 numbers.

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

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

Return values: Variable 'c' in the above example 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, values are passed as arguments are used to program 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 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 student:");
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.

Ans: 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.

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

In this example, the 'node' structure contains a integer "data" & 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

Eg:
```
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 3 character arrays for this street, city & state & 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 stucture that contains another structure as a member.

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

Ans: ~~1. Matlo~~ 1. malloc(): This function is used to allocate a block of memory of a specified size. It takes 1 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 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*)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. calloc(): This function is used to allocate a block of memory for an array of specified number of elements, each of a specified size. It takes two arguments, the 1st argument is the no. of elements in the array & the 2nd argument is the size of each element in bytes. It returns a pointer to the 1st byte of the allocated memory block. If the memory allocation is successful, the pointer returned by calloc() 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*)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");
 tree(p);
 return 0;
}
```

3. realloc(): This function is used to change the size of previously allocated memory block. It takes 2 arguments, the first argument is a pointer to the previously allocated memory block, 2nd argument is the new size of the memory block in bytes. It returns a pointer to the 1st byte of the re-allocated memory block. If the memory re-allocation is successful, the pointer returned by realloc() points to the 1st byte of the re-allocated memory block.

Ex:
```
#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_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 element are:");
 for(i=0;i<new_n;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" or "automatic variables". They are automatically created when the function is called & 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.

Ex: void ~~ma~~fun(){
 int n;
 n=5;
 printf("%.d",n);
 }

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 no of registers is limited, so not all variables can be stored in registers.

En: void func(){
    register int n;
    n=5;
    Printf("%d",n);
}

3. static: These are variables that retain their value blw function calls. They are also used to created 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 maintains its value between function calls.

En: void ~~main~~ func(){
    static int n=0;
    n++;
    Printf("%d",n);
}

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

En: //file 1.c
int n;
n=5;
//file 2.c
extern int n;
printf("%d",n);

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 & functions.

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

Program:
```
#include<stdio.h>
#include<string.h>
#define MAX_BOOK 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: %.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. These are given after the ~~ma~~ 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 at the type char which points to the arguments.

Ex:

```
#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;
}
```

Ans. output:

Argument is empty.

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

Pointer: It is a variable that stores the memory address of another as its value. Pointer is created with the * operator. There are few operations that are allowed to perform on pointers.

* Increment / Decrement:

→ Increment: When pointer is incremented, it actually increments by the number equals to the size of the datatype for which it is a pointer.

→ Decrement: When pointer is decremented, it actually decrements by the number equal to the size of the datatype for which it is a pointer.

Ex:

```
#include<stdio.h>
int main(){
int a=22;
int *p=&a;
```

```
printf("P = %u", p);
p++;
printf("P++ = %u\n", p);
p--;
printf("P-- = %u\n", p);
}
```

Output: P = 1uu1900792
P++ = 1uu1900796
P-- = 1uu1900792

* Addition: when a panter is added with a value the 1st multiplied by the size of datatype and then added to pointer.

Ex:
```
int main(){
int n = 4;
int *ptr1, *ptr2;
ptr1 = &n;
ptr2 = &n;
ptr2 = ptr2+3;
Printf("Pointer ptr2: %p", ptr2);
}
```

output: pointer ptr2 = 0x7ffca373daa8

* subtraction: when a pointer is subtracted with a value, the value first is multiplied by the size of the data type and then subtracted from pointer.

Ex:
```
#include<stdio.h>
int main(){
int n = 4;
int *ptr1, *ptr2;
ptr1 = 4n;
ptr2 = 4n;
ptr2 = ptr2-3;
printf("pointer ptr2: %p", ptr2);
return 0;
}
```

output: pointer ptr2: 0x7ffd718ffeb0

14. what is a file? Explain different modes of opening a file.

File: It is a collection of data stored in the secondary memory. It is used to storing informations that can be processed by programs. File mode is categorized into 4 types.

1. Create mode: opens the specified file and positions it to the beginning. To create a new file, open the file in create mode. User can't read, position a file opened with create mode.

2. Read mode: This mode opens a file for the reading of data. Read mode opens a file to the beginning.

3. Update mode: This mode allows both reading & writing of data. Update mode file opens a file to the beginning.

4. Append mode: This allows writing data to the end of a file. user ~~dont~~ can't read, position or rewind a file opened with append mode.

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

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  FILE *fptr1, *fptr2;
  char filename[100], c;
  printf("enter file name to pen: \n");
  scanf("%s", filename);
  fptr1 = fopen(filename, "r");
  if (fptr1 == NULL){
    printf("can't open file %s\n", filename);
    exit(0);
  }
  printf("enter file name to open for writing\n");
  scanf("%s", filename);
fptr2 = fopen(filename, "w");
if (fptr2 == NULL){
  printf("can't open file %s\n", filename);
  exit(0);
}
  c = fgetc(fptr1);
  while (c != EOF){
  fputc(c, fptr2);
    c = fgetc(fptr1);
  }
    printf("\n contents copied to %s", filename);
    fclose(fptr1);
    fclose(fptr2);
    return 0;
  }
```

output: enter filename to open:
a.txt

enter filename to open for writing
b.txt
content copied to b.txt.

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

Ans: '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 the first argument. For example, the following code reads an integer, a string, and a float from a file called "data.txt":

```
FILE *fp;
int i;
char str[100];
float f;
fp = fopen("data.txt", "r");
fscanf(fp, "%d %s %f", &i, 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 a file pointer, a buffer to store the read text, & the maximum number of characters to read as arguments. For example, the following code reads a line of text from a file called "data.txt" and prints it to the console.

```
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. 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 & 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 %f", i, str, f);
fclose(fp);
```

'fwrite()': This function is used to write binary data to a file. It taken a pointer to the data, the size of each element, the number of elements, and a file pointer a 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);
```

It's important to note that when reading & writing binary data, you should use "rb" and "wb" mode respectively.

17. Write a program to copy one file contents to another.

```
#include<stdio.h>
int main()
{
FILE *source, *target; // file pointers.
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("could not open target file.\n");
fclose(source);
return 1;
}
char ch;
while((ch=fgetc(source)) != EOF){
fputc(ch,target);
};
printf("File changed successfully.");
fclose(source);
fclose(target);
return 0;
}
```

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

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

(i) 'fopen (const char *filename, const char *mode)': this function is used to open a file. It takes the name of the file & 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 and 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);

En: int ch;
ch = fget (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);

En: fputc('A', fp);

(v) 'fread (void *ptr, size_t size, size_t count, FILE *fp): This function is used to read binary data from a file. It takes a pointer to the buffer, the size of each element, the number of elements and a file pointer as arguments.

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

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

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

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

Ex: 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, ...);

Ex: 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);