Name: Losta Sah

AP22110011490

Assignment-4

3. Write a C programme to implement Fibonacci series using recursion.

```
#include <stdio.h>
void fibonacci(int);
int main()
{
int n;
Printf("Enter the number of terms :");
scanf("%d", &n);
Printf("\n 0\t 1");
fibonacci(n-2);
 return 0;
}
void fibonacci (int n)
{
static int f1=0 ,f2=1, f3 ;
if(n>0)
{
  f3=f1+f2;
  f1=f2;
  f2=f3;
Printf("\t%d", f3);
fibonacci (n-1);
}
}
```

1. Explain different types of parameter passing mechanism techniques with suitable examples.

⇒ There are two types of parameters passing mechanism :-

- Call by value
- Call by refrence / address

Call by Value

In call by value parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called functions. The changes made on the formal parameters does not effect the values of actual parameters.

e.g :-

```
#include <stdio.h>
#include <conio.h>
void main() {
int num1, num2;
~~void main() {~~
void swap (int ,int);
num1 = 10 ;
num 2 = 20 ;
Printf("\n Before swap : num1 = %d , num 2 = %d",
        num1, num2);
```

```
swap(num1, num2);
Printf("\n After swap: num1 = %d\n num2 = %d
        num1, num2);
getch());
}
void swap(int a, int b)
{
int temp;
temp = a;
a = b;
b = temp;
}
```

Output

```
Before swap: num1 = 10
             num2 = 20
After swap: num1 = 10
            num2 = 20
```

Call by Refrence

In call by Reference parameter passing method, the memory location address of the actual parameter is copied to formal parameter. This address is used to access the memory locations of the actual parameter is called function. In this method of parameter passing, the formal parameters must be pointer variables. Here, the changes made on the formal parameter effects the values of actual parameters.

e.g:-
```
#include <stdio.h>
#include <conio.h>
void swap (int*, int*)
main()
{
int a,b;
Printf("\n Enter any two numbers:");
scanf("%d%d", &a, &b);
Printf("\n Before swap a=%dlt b=%d", a,b);
swap(&a, &b);
Printf("\n After swap a=%dlt b=%d", a,b);
}
void swap (int*x, int*y)
{
int temp;
temp=*x;
*x=*y;
*y=temp;
}
```

output

```
Enter any two numbers: 10 20
Before swap a=10 b=20
After swap a=20 b=10
```

2. C program for multiplication of two matrices in C.

```
#include <stdio.h>
void main()
{
int a[10][10], b[10][10], c[10][10], i, j, m, n, p, q, k;
Printf("Input row and column of A matrix \n");
scanf("%d %d", &p, &q);
if(n==p)
{
Printf("Matrices can be multiplied : \n");
Printf("Input A-matrix \n");
for(i=0; i<m; ++i)
for(j=0; j<n; ++j)
scanf("%d", &a[i][j]);
Printf("Input B-matrix \n");
for(i=0; i<p; ++i)
for(j=0; j<q; ++j)
scanf("%d", &b[i][j]);
Printf("The resultant matrix is : \n");
for(i=0; i<m; ++i)
{
for(j=0; j<q; ++j)
{
c[i][j]=0
for(k=0; k<p; ++k)
c[i][j]=c[i][j]+a[i][k]*b[k][j];
```

```
}
Printf("\n");
}
for(i=0 ; i<m ;i++)
{
for(j=0; j<q ;j++)
printf("%d\t", c[i][j]);
 printf("\n");
 }
 else
Printf("Matrices cannot be multiplied \n");
}
```

4. Describe and explain different built-in string handling functions with suitable examples.

⇒ The different built-in string handling functions are :

i) strlen() : It counts the number of character in a given string & returns the integer values.

Syntax : strlen(string 1)

It returns total number of characters in strings.

ii) strcpy() : It copies contents of one string into another.

syntax : strcpy(string 1, string 2)

It copies string 2 value into string 1.

iii) strncpy() :

syntax : strncpy (string 1, string 2, 5)

It copies first 6 characters string 2 in string 1.

iv) strcat() : It is concatenates (appends) portion of one string at the end of another string.

Syntax : strcat (string 1, string 2)

It joins (appends) string 1 & string 2.

v) strncat() :

Syntax : strncat (string 1, string 2)

Appends first 4 characters of string 2 to string 1.

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

⇒

```
#include <stdio.h>
#include <string.h>
int main()
{
char string[100];
Printf("Enter the string:");
scanf("%s", string);
char temp;
int i, j;
int n=strlen(string)
for(i=0; i<n-1; i++)
{
for(j=i+1; j<n; j++)
{
if(string[i] > string[j])
{
temp = string[i];
string[i] = string[j];
string[j] = temp;
}
}
}
Printf("The sorted string is: %s", string);
return 0;
}
```

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

⇒ A function is a block of code that performs a specific task and can be called by other parts of a program.

The structure of user defined function is:

```
return_type function_name(parameter_list)
{
  :
}
```

- return_type is data type of the value returned by the function. If the function does not return value, the return type should be 'void'.
- function_name is a unique identifier choosen by the programmer.
- Parameter_list is a list of variables that are passed to the function when it called. The parameter list can be empty if the function doesn't require any inputs.

Argument value:-
Argument values passed to the function when it is called. They are matched with the

parameter in the function definition. The number of arguments passed to the function should match the number of parameters in the function definition.

- Return Values:

Return values is the value returned by the function when it is executed. The return ~~statement~~ statement is used to return a value from the functions. If the return type of the function is void, the function does not return any value.

Eg.

```
#include <stdio.h>
int add (int, int)
int main()
{
  int a=20, b=30;
  int result = add(a,b);
  printf("%d", result);
  return 0;
}
int add (int a, int y)
{
  return a+y;
}
```

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

```
#include <stdio.h>
struct student
{
 int stno;
char name[10];
int m1, m2, m3;
int total; float average;
}; void main()
 {
struct student s[10];
int n, i;
Printf("Enter number of students:");
scanf("%d", &n);
for(i=0; i<n; i++)
 {
Printf("enter student number:");
scanf("%d", &s[i].stno);
Printf("enter student name:");
scanf("%s", s[i].name);
Printf("Enter marks m1, m2, m3");
scanf("%d%d%d", &s[i].m2, &s[i].m2, &s[i].m3);
s[i].total = s[i].m1+s[i].m2+s[i].m3;
s[i].average = (float) s[i].total / 3.0;
}
```

```
Printf("students information :\n");
for(i=0; i<n ; i++)
{
Printf("%d\n", s[i].sno);
Printf("%s\n", s[i].name);
Printf("%d %d %d %d\n" s[i].m1, s[i].m2, s[i].m3);
Printf("%d\n", s[i].total);
Printf("%f\n", s[i].average);
}
}
```

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

⇒ A self referential structure that contains a pointer to its own type. for E.g.

```
struct student
{
int roll no;
char name[20];
  struct node * w;
};
```

In this example, the 'student' structure contains an integer data and pointer to another 'student' structure. This allows for the creation of linked lists, where each

student points to the next student in the list.

On the other hand, a nested structure is a structure that contains another structure as a member. for Eg.:-

```
struct address {
    char street [50];
    char city [50];
    char state [20];
};
struct person {
    char name [50];
    int age;
    struct address man;
};
```

In this example the 'address' structure is needed within the 'person' structure uses this nested structure to represent the address of the person.

In summary, a self-referential structure is a structure that contains a pointer to its own type, and it is used to create complex data structures, like linked lists, while a nested structure is a structure that contains another structure as a member, it is used to group related data together and make the code more readable and organized.

9. Explain three dynamic memory allocations functions with suitable example.

⇒ The three dynamic memory allocations functions are:-

10.

(i) Malloc(): The malloc() function is used to dynamically allocate a block of memory of a specified size. for E.g. the following code allocates memory for an array 10 integers.

```
int *p
p=(int*) malloc (10*sizeof(int));
```

(ii) Calloc(): The calloc() function is similar to but it initializes the allocated memory to zero. for E.g. the following code allocates memory for a 2D array of 4 x 4 integers and sets all elements to 0.

```
int **p
p=(int**) calloc(4, sizeof(int*));
for(int i=0; i<4; i++)
p[i]=(int*) calloc(4, sizeof(int));
```

iii) realloc(): The realloc() function is used to change the size of a previously allocated block of memory. for E.g. the following

code increases the size of the previously allocated array of integers from 10 to 15.

```
p=(int *) realloc (p, 15*sizeof(int));
```

10. Explain about storage classes in C.

⇒ The different storage classes are:

a) Auto : Variables declared within a function or block have automatic storage class by default. These variables are created when the function or block is called and are destroyed when the function or block exists.

b) Register : Variables declared with the register storage class stored in CPU registers, rather than in memory. This can lead to faster access time, but there are fewer registers available than memory locations, so the use of register should be used sparingly.

c) Static : Variables declared with the static storage class retain their values between function calls. They are initialized only one and retain their value throughout the program's execution.

d) External : Variables declared with the external storage class are defined in one source file and can be accessed by other source files. This allows for the sharing of variables between source files.

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

```
#include <stdio.h>
struct writer
{
int number;
char name [20];
char title [20];
 int year;
 int price;
};
int main()
{
struct writer w;
Printf("Enter access Number:");
Scanf("%d", &w.number);
Printf("Enter authors name:");
```

```
scanf("%s", w.name);
printf(" Enter title name: ");
gets(w.title);
Printf(" Enter year of publication: ");
scanf("%d", &w.year);
Printf(" Enter price of book: ");
scanf("%d", &w.price)
Printf(" The data of the following member
        is: \n");
Printf(" access number = %d, authors name is
        %s title of book is %s year of
        publication %d book price = %d",
        w.number, w.name, w.title, w.year,
        w.price);
return 0;
}
```

12. Explain about command line argument with an example.

⇒ Command line arguments are input passed to a program when it is run from the command line. In C, they can be accessed using the 'argc' (argument count) and 'argv' (argument vector) variable in the 'main' function.

Example of program that takes in one command line argument and prints it out:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
if (argc != 2)
{
Printf("Expected 1 argument. Got %d\n", argc-1);
return 1;
}
Printf("You entered: %s\n", argv[1]);
return 0;
}
```

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

⇒ A pointer is a variable that stores the memory address of another variable. Pointer are used to dynamically allocate memory and to manipulate memory directly. Pointer arithmetic is a set of operations that can be performed on pointers.

For Example:

We can increment or decrement a pointer to point to the next previous memory locations, respectively.

```
#include <stdio.h>
int main()
{
int num = 5
int *ptr = &num;
Ptr ++;
ptr --;
Printf("%d %d", *ptr, *ptr);
return 0;
}
```

In this example, 'Ptr' is a pointer to an 'int' variable and '&num' is the memory address of the 'num' variable. The '++' operator increments the pointer to point

to the next memory location. The '*' operator is the dereference operator which is used to acess the value stored at the memory location pointed by the pointer.

Another Eg: We can subtract two pointers and it will give you the number of elements betn the two pointer.

```
#include <stdio.h>
int main()
{
int arr[5] = {1,2,3,4,5}
int * ptr1 = &arr[0];
int * ptr2 = &arr[4];
Printf("%d", ptr2 - ptr1);
return 0;
}
```

It prints 4th value, because it's pointing 4th element in array.

It's important to remember that pointer arithmetic only works on pointers of the same data type, & the result of pointer arithmetic is also a pointer.

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

⇒ A file is a collection of data that is ~~define~~ stored on a computer's hardware drive or other storage device.
In C programming files can be opened & read or written to using the standard input/output library (stdio.h).

There are several different modes that a file can be opened in:

(a) 'r' (read-only): It is open an existing file for reading only.
(b) 'w' (write-only): It is open and new file for writing only. If a file with specified filename currently exists, it will be destroyed and a new file with the same name will be created in its place.
(c) 'a' (append-only): It is open an existing file for appending (i.e. for adding new information at the end of file). A new file will be created if the file with the specified filename doesn't exist.
(d) 'r+' (read & write): It is open an existing file for update (i.e. for both reading and writing).

(e) 'w+' (write & read): Open a file with the specified filename for both writing and reading. If a file with the specified filename currently exists, it will be destroyed and new file will be created in its space.

(f) 'a+' (append & write): Open an existing file for both appending and reading. A new file will be created if the file with specified filename doesn't exist.

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

⇒ Ex-1 Reading operations on a file

```
FILE *fp;
fp=fopen("sample.txt", w);
If (fp==NULL)
{
Printf("\n Unable to open a file");
exit(0);
}
```

Ex-2

```
#include <stdio.h>
 int main()
 {
 FILE *file;
char filename[] = "example.txt");
char buffer[255];
file = fopen(filename, "r");
 if(fille == NULL)
 {
Printf("Error Opening file !\n");
 return 1;
 }
while (fgets(buffers, 255, file) != NULL)
 {
PrintF("%s", buffer);
 }
fclose(file);
return 0;
}
```

Writing Operation on a file.

```
#include <stdio.h>
 int main()
 {
file = fopen("example.txt", "w");
if(file == NULL)
```

```
{
Printf("Error Operating file!\n");
  return 1;
}
fprintf(file, "Hello World!\n");
fclose(files);
Printf("writing operations on the file is
        successful\n");
  return 0;
  }
```

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

(a) fscanf(): It is a function in C that reads formatted input from a file stream. It works similarly to the scanf() function but reads input from a file rather than the standard input.

E.g:

```
#include <stdio.h>
  int main()
  int x, y;
FILE *fp = fopen("input.txt", "r");
 fscanf(fp, "%d %d", &x, &y);
Printf("x: %d, y: %d\n", x, y);
fclose(fp);
return 0;
  }
```

(b) fgets(): It is a function in C that reads a string from a file stream. It reads a line of text from the file, up to a specified maximum number of characters, & stores it in a character array.

Eg.

```
#include <stdio.h>
int main()
{
char str[100];
FILE *fp = fopen("input.txt", "r");
fgets(str, 100, fp);
printf("Read: %s\n", str);
fclose(fp);
return 0;
}
```

(c) fprintf(): It is a function that writes formatted output to a file stream. It works similarly to the 'printf()' function but writes output to a file rather than the standard output.

Eg:

```
#include <stdio.h>
int main() {
int x=10, y=20;
FILE *fp = fopen("output.txt", "w");
fprintf(fp, "x: %d, y: %d\n", x, y);
fclose(fp);
return 0;
}
```

(d) fwrite(): It is a function that writes a specified number of bytes to a file stream. It can be used to write binary data to a file.

E.g.

```
#include <stdio.h>
int main() {
int data[] = {1,2,3,4,5};
FILE *fp = fopen("Output.bin", "wb");
fwrite(data, sizeof(int), 5, fp);
fclose(fp);
return 0;
}
```

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

```
#include <stdio.h>
int main()
{
FILE *fp1, *fp2;
char ch;
fp1 = fopen("source.txt", "r");
if(fp1 == NULL)
{
printf("Error opening source file.\n");
return 0;
}
```

```
fp2 = fopen ("destination.txt", "w");
if(fp2 == NULL)
{
printf("Error creating destination file.\n");
return 0;
}
while (ch = fgetc(fp1) != EOF)
fputc (ch, fp2);
}
printf("file copied successfully.\n");
fclose (fp1);
fclose(fp2);
return 0;
}
```

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

⇒ The use of file handling functions to perform operations on files such as opening, reading, writing & closing.

(a) fopen(): This function is used to open a file and returns a pointer to the file.

Syntax:

```
FILE *fopen(const char *file name, const
    char * mode);
```

E.g.

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

(b) fclose() : This function is used to close a file.

syntax :

int fclose(FILE *fp);

E.g. : fclose(fp);

(c) fgetc() : This function is used to read a single character from a file.

syntax : int fgets (FILE *fp);

E.g. : char ch;

ch = gets(fp);

(d) fputs() : This function is used to write a single character to a file.

syntax : int fputs (int char, FILE *fp);

E.g. :

fputs ('A', fp);

(e) fread() : This function is used to read a block of data from a file.

syntax :

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

E.g.

Char buffer [100];

fread (buffer, size of (char), 100, fp);

(f) fwrite(): This function is used to write a block of data to a file.

Syntax:

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

Eg.
```
char buffer[100] = "Hello World!";
fwrite(buffer, sizeof(char), 100, fp);
```

(g) fseek(): This function is used to move file pointer to a specific position in a file.

Syntax: `int fseek(FILE *fp, long int offset, int origin);`

Eg.
```
fseek(fp, 0, SEEK_END);
```

(h) ftell(): This function is used to determine the current position of the file pointer.

Syntax: `long int ftell(FILE *fp);`

Eg.
```
long int position;
position = ftell(fp);
```

(i) rewind() This function is used to move the file pointer to the beginning.

Syntax :- `void rewind(FILE *fp);`

Eg: `rewind(fp);`