June/July 2024

Module - 3

5a. Discuss the implementation of user defined function with examples. (Answers will be given next)

→

Nov/Dec 2023

5a. Implement matrix multiplication and validate the rules of multiplication.
(Available in the lab program No. 6)

5b. Summarize the recursive function concept with suitable example. (5M)

→ A recursive functions is defined as a function that calls itself to solve a smaller version of its task until final call is made which doesn't require a call to itself. Every recursive solution has two major cases they are:

→ Base case: The problem is simple enough to solve ~~recursively~~ directly without making any further calls to the same functions.

→ Recursive case: The problem at hand is divided into simpler sub parts. Second the function calls itself but with subparts of the problem obtained in the first step. Third the result is obtained by combining the solution of simpler sub-parts.

→ Recursion is defining large and complex problems in terms of a smaller and more easily solvable problems. In recursive function, a complex problem is defined in terms of simpler problems and the simplest problem is given explicitly.

Factorial of a number

```
#include<stdio.h>
main()
{
    int num;
    printf("\n enter the number: ");
    scanf("%d", &num);
    printf("result = %d", fact(num));
}
int fact(int n)
{
    if((n==0) || (n==1))
    return 1;
    else
    return n*fact(n-1);
}
```

→ Base case is when n=1 or n=0 because if n=1 or n=0 the result will be 1

→ Recursive case of the factorial function. will call itself but with a smaller value of n, this case can be given as n* fact(n-1).

5E Explain declaration and initialization of 1D and 2D Array with an example. (5M)

→ Array: An array is a collection of items of same data type stored at contigous memory locations.

→ Array of a character is a string.

→ Each data item of an array is called an element.

→ And each element is unique and located in seperated memory location.

→ Each of elements of an array share a variable but each element having different index no. known as subscript.

(8)

Any element in an array can be accessed using
1. Name of the array
2. Position of the element in an array.

There are two types of array
1. Single dimensional array
2. Multi dimensional array.

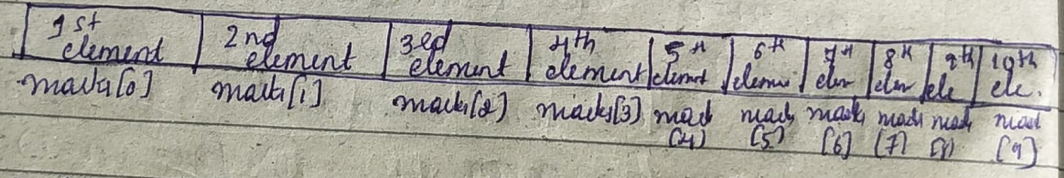
Declaration of 1D arrays.
→ Arrays are declared using following syntax:

data_type array_name [size];

where, type can be int, float or char.

name is the name of the array
size indicates no. of elements in the array.

Example : int marks [10];



Extra 😊 Calculating the address of Array

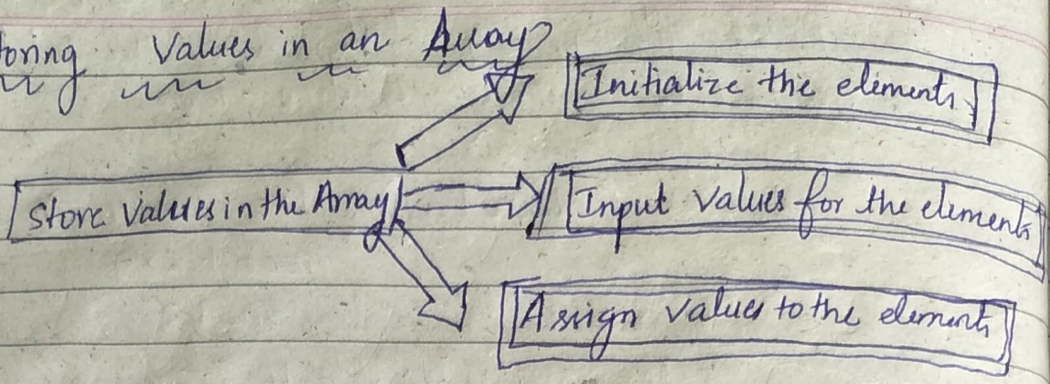
→ Address of data element, A[k] = BA(A) + w (k - lower_bound)

Here,

A → Array
k → index of the element of which we have to calculate the address.
BA → base address of the array, A
w → word size of one element in memory, for example, size of int is 2.

(5)

Storing Values in an Array



Initialization can be done using the following syntax

type array_name [size] = { list of Values };

1. Initializing all specified memory locations:

int a[5] = {10, 20, 30, 40, 50}

10	20	30	40	50
[0]	[1]	[2]	[3]	[4]

2. Partial array initialization :

int a[5] = {10, 20}

10	20	0	0	0
[0]	[1]	[2]	[3]	[4]

3. Array Initialization without Size:

int a[] = {10, 20, 30, 40, 50}

10	20	30	40	50
[0]	[1]	[2]	[3]	[4]

4. Array initialization without elements

int a[5] = {0}

0	0	0	0	0
[0]	[1]	[2]	[3]	[4]

Operations on array:
1. Traversing an array
2. Inserting an element in an array
3. Deleting an element from an array
4. Merging 2 arrays.
5. Searching an element in an array
6. Sorting an element array in ascending or descending order

1. Traversing an array

WAP to read and write one dimensional array

```
#include <stdio.h>
Void main() {
int i, a[5];
printf (" Enter the elements :");
   for (i=0 ; i<5 ; i++) {
      scanf ("%d", &a[i]);
   }
for (i=0; i<5; i++) {
   printf ("Array a[%d] = %d \n", i, a[i]);
   }
}
```

Output:

Enter the elements : 1 2 3 4 5
Array a[0] = 1
Array a[1] = 2
Array a[2] = 3
Array a[3] = 4
Array a[4] = 5

(7)

2-D Arrays:

Arrays with 2 dimensions are called 2-Dimensional arrays or 2-D array.

Declaration of 2-D arrays:

```
data_type array_name [row_size][column_size];
```

data_type can be any primitive data type
array_name is a variable name.
row_size is the maximum numbers of rows in the array.
column_size is the maximum number of column in the array.

Example: in a[2][3];

This can be read as.

R|C	Column 0	Column 1	Column 2
Row 0	a[0][0]	a[0][1]	a[0][2]
Row 1	a[1][0]	a[1][1]	a[1][2]

Initialization of 2-D array:

1. Initialize with total number of elements:
   int a[2][3] = {1,2,3,4,5,6};
2. Initialize with sets.
   int a[2][3] = {{1,2,3},{4,5,6}};
3. Partial initialization.
   int a[2][3] = {{1,1},{2}};
4. Initialize without size.
   int a[][3] = {{1,2,3},{4,5,6}};

Initialization of 2-D array:

```
for (i=0; i<row; i++) {
    for (j=0; j< column; j++) {
        scanf ("%d", &a[i][j]);
    }
}
```

6a. Explain the syntax of function declaration, function call and function definition with an example C program. (06 M)

Functions are independent program modules that are designed to carry out a particular task.

A large program can be divided into manageable pieces called modules where each module does a specific task. Thus, the functions often called modules are self-contained small programs that carry out some specific, well defined tasks. Functions act like building blocks using which any desired activity can be performed by combining one or more function.

```
main() {
    ---------
    ---------
    func1();
    ---------
    ---------
    return 0;
}
```

```
func1()
{
statement block
}
```

There are two types:

1. Built-in (library) functions.
2. User defined functions.

1. Builtin functions: These are C language functions already available with C compilers and can be used by any programmers.

Ex: printf(), scanf().

2. User-defined functions: These are written by programmers for their own purpose and are not readily available.

↓ There are 3 elements
→ Function definition
→ Function call
→ Function declaration / Function prototype

1. Function definition: As we normally declare the variable before they are used the functions also should be declared before they are used. is called function prototype This process of declaring the function before they are used in called function prototype.

The function prototype also known as function declaration.

A function declaration consists of four parts:
→ Function type / return type
→ Function name
→ Parameter List
→ Terminating semicolon

Syntax: return_type function_name (list of parameters);

Ex: int sum (int, int);

The following points should be kept in mind about function declaration:

→ After the declaration of every function there should be a semicolon. If the semicolon is missing, the compiler will generate an error message.

→ The function declaration is global. Therefore the declared function can be called from Any point in the program.

→ A function called be declared within the body of another function.

→ A function having void as its return type cannot return any value.

2. Function definition: The program module that is written to achieve a specific task is called function definition.

Each function definition consists of two parts Namely:

→ Function header

→ Function body

Syntax:

function_type function_name (list of parameters)

{

local variable declaration;

executable statement 1;

executable statement 2;

.....

.....

return statement;

}

function header

function body

(11)

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

→ function header

→ function body

Note: The first line function_type function_name (parameter list) is known as the function header and the statements within opening and closing braces is known as the function body.

i) Function Header : It consists of three parts :

Function type
Function name and
List of parameters.

The semicolon is not present at the end of header.

✔ Function type :

→ The function type specifies the type of value that the function is expected to return to the calling function.
→ If the return type or function type is not specified, then it is assumed as int.
→ If function is not returning anything, then it is void.

✔ Function name :

→ The function name is any valid C identifier and must follow same rules as of other variable.

✓ List of Parameters: The parameter list declares the variables that will receive the data sent by the calling program which serves as input and known as Formal parameter List.

ii) Function Body: The function body contains declarations and statements necessary for performing the required task. The body is enclosed in braces, contains three parts:

1) Local declarations that specify the variable needed by the function.
2) Function statements that perform the task of the function.
3) A return statement the value evaluated by the function.

The number and order of arguments in the function header must be same as that given in the function declaration statement.

The function header is same as that of function declaration.
The only difference b/w the two is that list of variable is not followed by semicolon.

Function Call:
The function call statement invokes the function when a function is invoked the compiler jumps to the called function to execute the statements that are a part of that function

Syntax:

function name (variable 1, variable 2, ...);

Example: add (a,b);

6b. Describe the different types of storage classes with an example. (08M)

→ Storage classes:

→ The storage class of a variable defines the scope and lifetime of variable and/or functions declared within a C program.

→ The storage class of a function or a variable determines the part of memory where storage space will be allocated for that variable or function.

→ It specifies how long the storage allocation will continue to exist for that function or variable.

Syntax of storage class:

Storage class.

Various storage classes in C language are classified as:

1. Automatic (local Variable): Local variables are the variables which are defined with in a function. These variables are also called automatic variables. All variables must be defined before the first executable in the function.

Whenever control ~~events~~ enters into the function memory is allocated for the local variable and whenever control goes out of the function memory will de-allocated.

That is the reason the local variables cannot be altered by any other function.

The local variables have the following features:

These variables cannot be altered by any function and are alive and active within the function.

→ The scope of these variables is limited only to the function in which they are declared and cannot be accessed outside the function.
→ No other fn can use a local variable and change its value.
→ The auto is the default storage class for all local variables.

Ex:

```
#include<stdio.h>
void func1(){
    int a=10;
    printf("\n a=%d",a); // auto integer local variable
}
void func2(){
    int a=20;
    printf("\n a=%d",a); // auto int local to func2
}
void main(){
    int a=30; // auto integer local or main()
    func1();
    func2();
    printf("\n a=%d",a);
}
```

2. Static : Static the default storage class for global variables. The variable that are declared using the keyword static are called static variable. The static variable can be declared outside the function or within the function. They have the characteristic of both local and global variable.
The declaration of static variable should begin with

keyword static

Example: static int a,b;

Static variable have a lifetime over the entire program, i.e memory for the static variables is allocated when the program begins running and is freed when the program terminates.

3. Register:

- When a variable is decided using register as the storage class. It is stored in a CPU register instead of RAM. Since the variable is stored in RAM, the maximum size of the variable is equal to the register size.
- One drawback of using a register variable is that they cannot be operated using the unary '&' operator because it does not have a memory location associated with it.

Ex:

```
#include <stdio.h>
int exp (int a, int b);
main(){
    int a=3, b=5, res;
    res = exp(a,b);
    printf("\n %d to the power of %d= %d", a, b, res);
    getch();
    return 0;
}
int exp (int a, int b)
{
  Register int res=1;
  int i;
```

```
for (i=1; i<b; i++)
    res = res * b;
    return res;
}
```

Output:

3 to the power of 5 = 243.

4. Extern:

The extern storage class is used to give reference of a global variable that is visible to all the program files. Such global variables are declared like any other variable in one of the program file when there are multiple file in a program and you need to use a particular function or variable in a file apart from which it is declared then use the key word extern.

Ex:

```
int x;
void print (void);
int main() {
    x=10;
    printf("\n x in FILE1 = %d", x);
    print();
    return 0;
}
#include <stdio.h>
extern int x;
void print() {
    printf("\n x in FILE 2 = %d", x);
}
main() {
    statements;
}
```

Output:

X in FILE1 = 10
X in FILE2 = 10

8c) Write a C program to sort the array elements in ascending order. (6M)

An array is a group of related data items which share a common name. A particular value in an array is identified with help of its "index number"

```
#include <stdio.h>
    // Header file for standard input/output fns.
void main() {
  int num[20]; // Array to store up to 20
                // integers
  int i, j, a, n; // Loop counters and temporary
                  // variable
//prompt user to enter number of elements
printf("Enter number of elements in an array: ");
scanf("%d", &n); // Read number of elements from
                 // user
//prompt user to enter array elements
printf("Enter the elements: ");
for(i=0; i<n; i++)
  scanf("%d", &num[i]);
  //Read each element into the array.

// Sorting the array using Bubble Sort algorithm
for(i=0; i<n; i++) {
  for(j=i+1; j<n; j++) {
    if(num[i] > num[j]) { // compare two elements
      a = num[i];
```

POP · LAB 5 program is fine
Bubble Sort ☺ for this.

```
                num[i] = num[j];
                // Swap if elements are in wrong order
                num[i] = num[j];
                num[j] = a;
            }
        }
    }
    // print the sorted array in ascending order.
    printf("The number in ascending order are: ");
    for(i=0; i<n; ++i){
        printf("%d", num[i]); // print each element with a space
    }
}
```

Summary:
→ The program asks an array of integers entered by the user in ascending order.
→ It uses the Bubble sort algorithm
→ The final sorted array is displayed using a loop.

Output:
Enter the number of elements in an array.
5
Enter the elements
12
23
89
11
22
The number in ascending order is:
11
12
22
23
89

June/July - 2022

5a) (19) Example/ Program for function definition, fn. dularation & call. (20)

Repeat

```
#include <stdio.h>
int sum (int a, int b); // Function Dularation
int main () {
    int num1, num2, total=0;
    printf ("\n enter the first number: =");
    scanf ("%d", &num2);
    printf ("\n enter the second numbers: ");
    scanf ("%d", &num2);
    total = sum (num1, num2); // function call
    printf ("\n total = %d", total);
}
int sum (int a, int b) // Function Header.
{ // Function Body.
    int result;
    result = a+b;
    return result;
}
```

Please write the comments for external programs for understanding extra program to the evaluator (5)

output:

Enter the first number: 20

Enter the second number: 10.

total = 30.

5b. WAP to swap two numbers using call by reference method.

```
#include<stdio.h>
// Function dularation with pointers to enable call by
// reference
void swap (int *x, int *y){
    int temp;
    // Swapping values using a temporary variable
    temp = *x; // store value at address x in temp
    *x = *y; // Assign value at address y to address x
    *y = temp; // Assign temp (original x) to address y
}
```

10

```
int main() {
    int a, b;
    // input from user
    printf("Enter two numbers to swap: ");
    scanf("%d %d", &a, &b); // Read two integers

    // Display before swapping
    printf("Before swapping: a = %d, b = %d\n", a, b);

    // function call using addresses (call by reference)
    swap(&a, &b);

    // Display after swapping
    printf("After swapping: a = %d, b = %d\n", a, b);

    return 0;
}
```

Explanation:

→ swap(int *x, int *y) takes pointers to the two integers so that it can modify their values directly in memory.

→ *x and *y are dereferenced pointers, giving access to the values of the original variables a and b.

→ This demonstrates Call by Reference; changes inside the function reflect outside.

Output:

Enter two numbers to swap: 10 15
Before swapping: a=10, b=25
After swapping: a=25, b=10

5c. <— Repeat —>

June-July - theory

Q5a. Define Function. Write the syntax of a function. Explain the categories of function (4m)
(Answer are there already in the previous please refer ☺ for Function defn. & syntax)

Categories of function:

Based on parameters and return values the functions are categorized as.

1. Function with no parameters and no return value

This category is also called "Void function with no parameters". In this category, there is no data transfer between the calling function and called function. So calling function cannot send values and called function cannot recieve the data

Example:

```
#include <stdio.h>
Void sum();  /* declaration */
Void main()
{
    sum();
}
```

Calling function

```
Void sum()
/*No parameters*/
{
    int a,b,c;
    printf("Enter the values of a and b");
    scanf("%d%d",&a,&b);
    c=a+b;
    printf("sum=%d",c);
}
/*No return Values*/
```

called function

2. Function with no parameters and return values:
In this category there is no data transfer from the calling function. But, there is data transfer from called function to the calling function. When the function returns a value, the calling function receives one value from the called function.

```
#include <stdio.h>
int sum();          /* declaration */
Void main(){
    int c;
    c= Sum();
    printf("Sum = %d", c);
}
```

[calling function]

```
int sum()          /* No parameters */
{
  int a, b, c;
  printf("Enter the value of a and b");
  scanf("%d %d", &a, &b);

  c= a+b;
  return c;
}      /* returns a value */
```

[called function]

3. Function with parameters and no return values:
This category also called 'void' functions with parameters. In this category, there is data transfer from the calling function to the called function using parameters. But there is no data transfer from called function to the calling function.

Ex:

```
void main() {
    int m, n;
    printf("Enter m and n");
    scanf("%d %d", &m, &n);
    sum(m, n);
}
```

Calling function

```
void sum(int a, int b) {
    /* with parameters */
    int c;
    c = a + b;
    printf("Sum = %d", c);
}   /* No return value */
```

called function

4. F. Functions with parameters and return values

In this category there is a data transfer between the calling function and called function. When parameters are passed, the called function can receive values from the calling function. When the fn. returns a value the calling fn. can receive a value from the called fn..

Ex:

```
#include <stdio.h>
int Sum(int a, int b); /* declaration */
void main() {
    int m, n, c;
    printf("Enter m and n");
    scanf("%d %d", &m, &n);
    c = Sum(m, n);
    printf("sum = %d", c);
}
```

calling function

Q4

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

[called function]

5b) Write a C program to swap two numbers using call by reference method. (6 Marks)

← Answer is already available →

5c) Describe different types of storage classes with examples. (8 Marks)

← Answer is already available →

6a) What is an array? Explain how arrays are declared and initialized with example (8 Marks)

← Answer is already available →

6b) Write a C program to transpose a 3x3 matrix (08 Marks)

```
#include<stdio.h>
int main(){
  int matrix[3][3] ={{1,2,3},{4,5,6},{7,8,9}};
  int transpose[3][3];

  // Transpose the matrix
  for(int i=0; i<3; i++){
    for(int j=0; j<3; j++){
      transpose[j][i]= matrix[i][j];
  }
}
```

Continuation →

```
// print the transposed matrix
Printf ("Transpose of the matrix: \n");
for (int i=0; i<3; i++) {
    for (int j=0; j<3; j++) {
        printf ("%d", transpose[i][j]);
    }
    printf ("\n");
}
return 0;
}
```

Output:

```
Transpose of the matrix:
1 4 7
2 5 8
3 6 9
```

Q6) List applications of arrays (6+M)

- Storing and accessing data: Arrays are used to store and retrieve data in a specific order. For example, an array can be used to store the scores of a group of students, or the temperatures recorded by a weather station.
- Sorting: Arrays can be used to sort data in ascending or descending order. Sorting algorithms such as bubble sort, mergesort and quicksort rely heavily on arrays.
- Searching: Arrays can be searched for specific elements using algorithms such as linear search and binary search.
- Matrices: Arrays are used to represent matrices in mathematical computations such as matrix multiplication, linear algebra, and image processing.

- Stacks and queues : Arrays are used as the underlying data structure for implementing stacks and queues, which are commonly used in algorithms and data structures.

- Graphs : Arrays can be used to represent graphs in computer science. Each element in the array represents a node in the graph, and the relationships between the nodes are represented by the values stored in the array.

Dec 2023 / Jan 2024

Q5 Discuss in detail the parts of a user-defined function (10M)

← already explained and written answers in the previous page →

Q5b Discuss storage classes in C (10M)

Q6. Define recursion. Write a C program to find the factorial of 'n' using recursion.

Recursion → it a technique that breaks a problem into one or more sub-problems that are similar to the original problem.

Any recursive functions can be categorized based on the following:

→ Whether the function calls itself directly or indirectly - (direct or indirect recursion)
→ Whether any operation is pending at each recursive call (tail recursion or not)
→ The structure of the calling pattern (linear or tree recursion)

27

1. Direct Recursion: A fn. is said to be directly recursive if it explicitly calls itself.

Example:

~~A fn. is said to be directly recursive if it~~

```
int Func(int n){
    if (n==0)
    return n;
    return (Func(n-1));
}
```

Here the fn. Func() calls itself for all positive value of n, so it is said to be a directly recursive fn.

2. Indirect Recursion: A fn. is said to be indirectly recursive if it contains a call to another fn. which ultimately calls it.

Example:

```
int Func1(int n){
    if(n==0)
    return n;
    return Func2(n);
}
int Func2(int x){
    return Func1(x-1);
}
```

3. Tail Recursion: A recursive fn. is said to be tail recursive if no operations are pending to be performed when the recursive function returns to its caller. When the called function returns the returned value is immediately returned from the calling function.

Ex:

```
int fact(int n){
    if(n==1)
    return 1;
    return (n*fact(n-1));
}
```

Example for non tail:

```
int Fact(n) {
    return Fact1(n, 1);
}
int Fact1(int n, int res) {
    if(n==1)
    return res;
    return Fact1(n-1, n*res);
}
```

→ PROGRAM CODE ☺ ←

```
#include<stdio.h>
int fact(int n);
int main() {
    int num;
    printf("\n enter the number: ");
    scanf("%d", &num);
    printf("result = %d", fact(num));
    return 0; }
int fact(int n) {
    if(n==0 || n==1)
    return 1;
    else
    return n * fact(n-1);
}
```

- Base case is when n=1 or n=0 because if n=1 or n=0 the result will be 1
- Recursive case of the factorial function will call itself but with smaller value of n, this case can given as

  n * fact(n-1)

Output:
Enter the number : 5
Result = 120

6b What is an array? Explain the declaration and initialization of 1-D arrays.

← Repeat (5M) →

6c Write a C program to perform matrix multiplication

← Repeat (8) → Refer pop lab program 6

June / July 2024

8a. Differentiate pass by value and pass by reference/address parameter passing techniques. (5M)

Pass by Value	Pass by reference/address
① When a fn/.. is called the values of variable are passed	When a fn/.. is called the address of variables are passed
② The type of formal parameters should be same as type of actual parameters	The type of formal parameters should be same as type of actual parameters but they have to be declared as pointers
③ Change of formal parameters in the function will not affect the actual parameters in the calling fn/..	The actual parameters are changed since the formal parameters indirectly manipulate the actual parameters
④ Execution is slower since all the values have to be copied into formal parameters	Execution is faster since only address are copied.
⑤ Transfer of info/.. is possible only in one direction i.e from calling fn/.. to the called fn/..	Even though only one value can be returned using return statement, through the parameters we can send the values to actual parameters. Thus, two way communication is possible.

30

6b → 2d Array – blah blah → already written previously

6c. Define recursion. Mention the properties of recursion fn. . Write a 'C' program to find GCD of 2 numbers using recursive fn.

fibanacci (Marked)

```
#include<stdio.h>
int fib (int n);
int main (){
    int i, n;
    printf (" Enter the number of terms: ");
    scanf ("%d", &n);
    for (i=0; i<n; i++) {
        printf ("%d", fib(i));
    }
    return 0;
}
int fib (int n) {
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Output:

```
Enter the number of terms : 5
0 1 1 2 3.
```

GCD

```
#include <stdio.h>
int main (){
    int a, b;
    printf (" Enter the value of a and b: ");
    scanf ("%d %d", &a, &b);
```

30

6b → 2d Array – blah blah → already written previously

6c. Define recursion. Mention the properties of recursion fn. . Write a 'C' program to find GCD of 2 numbers using recursive fn.

fibanacci (Marked)

```
#include<stdio.h>
int fib (int n);
int main (){
    int i, n;
    printf (" Enter the number of terms: ");
    scanf ("%d", &n);
    for (i=0; i<n; i++) {
        printf ("%d", fib(i));
    }
    return 0;
}
int fib (int n) {
    if (n==0)
        return 0;
    else if (n==1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

Output:

```
Enter the number of terms : 5
0 1 1 2 3.
```

GCD

```
#include <stdio.h>
int main (){
    int a, b;
    printf (" Enter the value of a and b: ");
    scanf ("%d %d", &a, &b);
```

```
printf("Result = %d\n", GCD(a,b));
return 0;
}

int GCD(int a, int b){
    if(a==0)
        return b;
    else if(b==0)
        return a;
    else
        return GCD(b, a%b);
}
```

Output:

Enter the value of a and b: 12 18
result = 6.

⟵ All most important Questions from MODULE-3 are Covered ⟶

Done ⟶ module-3 ⟶ All All ✌

⟵ Prepared by: Aleya ☺ Dept. CSE, BNMIT ✌

Module - 4 - POP

7a) Define string. Explain any four string manipulating functions with example. (10M)

→ A group of characters together is called string. String is always enclosed within double quotes (" ").
String always ends with delimiter (NULL, '\0').

Example:

String storage

Char str[] = "HELLO";

H	E	L	L	\0

↑ Beginning of string. ← end of string

Char str[] = "H";

H	\0

Here H is a string not a character. The string H requires two memory locations. One to store the character H and another to store the null character.

V/S

Character storage

char ch = 'H';

Here H is a character not a string. The character H requires only one memory location.

H

Char str[] = "";

\0

Empty string.

Although C permits empty string, it doesnot allow an empty character.

Declaration of a string: A string is declared like an array of character.

- Syntax: data type string name [size];
- Example: char str[10];

Initialization of a string:

Syntax: datatype stringname [size] = value;

Example:

1. char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};

Here compiler will automatically calculate the size based on the no. of elements initialized.

So, in this example 6 memory slots will be reserved to store the string variable.

str[0]	1000	H
str[1]	1001	E
str[2]	1002	L
str[3]	1003	L
str[4]	1004	O
str[5]	1005	\0

String Manipulation functions:

1. strcat() function:

Syntax: char * strcat (char *str1, const char *str2);

The strcat() function appends the string pointed by str2 to the end of the string pointed to by str1.

Ex:

```
#include <stdio.h>
#include <string.h>
int main(){
    char str1[50] = "Programming ";
    char str2[] = "in C";
    strcat(str1, str2);
    printf("\n str1 : %s", str1);
    return 0;
}
```

Output:
str1: Programming in C

2. strncat() function:

Syntax:

```
char *strncat(char *str1, const char *str2,
              size_t n);
```

This append the string pointed to by str2 to the end of the string pointed to by str1 upto n characters long.

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[50] = "programming";
    char str2[] = "in C";
    strncat(str1, str2, 2);
    printf("\n str1 : %s", str1);
    return 0;
}
```

output:
str1: programming in

3. Strchr() function:

Syntax:

```
char *strchr(const char *str, int c);
```

This function searches for the first occurence of the character c in the string pointed to by the argumented str. The function returns a pointer pointing to the character, or null if no match was found.

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[50] = "Programming in C";
    char *pos;
    pos = strchr(str, 'n');
    if (pos)
    printf("\n n is found in str at position %d",
            pos);
    else
    printf("\n n is not present in the string");
    return 0;
}
```

Output:

n is found in str at position 9

4. strchr() function:

Syntax:

```
char *strchr(const char *str, int c);
```

The strchr() function searches for the first occurence of the character c beginning at the rear end and working towards the front in the string pointed to by the argument str.

Example:

```
#include <stdio.h>
#include <string.h>
int main() {
    char str[50] = "programming in C";
    char *pos;
    if (pos)
    printf("\n the last position of n is %d", pos-str);
    else
```

printf("In n is not present in the string");
return 0;
}

output:
The last position of n is: 13.

7b. Write a C program to concatenate two strings without using built-in function strcat(). (5M)

Already written previously 😊

7c. Explain string unformatted input/output functions with example. (5M)

→ Unformatted I/O functions: These functions are used exclusively for character data types or character arrays/strings. They are designed for reading single inputs from the user at the console and for displaying values on the console.

→ Why "Unformatted"?: They are referred to as "Unformatted" I/O functions because they don't support format specifiers. Unlike formatted I/O functions like printf() and scanf(), you cannot use format specifiers to control the formatting of the data.

The following are unformatted I/O functions.

1. getch()
2. getche()
3. getchar()
4. putchar()
5. gets()
6. puts()
7. putch()
8. getch().

(6) Examples

1- getch() : In C, getch() reads a single character from the keyboard without displaying it on the console screen.

→: It immediately returns without requiring the user to press the enter key.

← This fn. is defined in the conio.h header file.

Ex: // C program to implement getch() function

```
#include <conio.h>
#include <stdio.h>
int main(){
    printf(" Enter any character: ");
    // Reads a character but not displays
    getch();
    return 0;
}
```

Output:

Enter any character

2. putch() : In C, putch() is used to display a single character provided by the user, and it prints the character at the current cursor location. This function is defined in the conio.h header file.

Ex:

```
#include <conio.h>
#include <stdio.h>
int main(){
    char ch;
    printf("Enter any character: \n");
    // Reads a character from the keyboard
    ch = getch();
    printf("\n Entered character is: ");
```

```
// Displays that character on the console.
putch(ch);
return 0;
}
```

output:

Enter any character: d
Entered character is: d

Q8a. Define pointer. Explain pointer variable declaration and initialization with suitable example. (08M)

A pointer is a variable that contains the address of a variable.
Pointer is indicated by * symbol.
It known as dereferencing operator.

Pointer declaration & initialization:

pointer declaration:
Declaration refers to the process of creating a pointer declaration of a pointer.

Syntax:

Data type *Ptr_name;
Here data type is the data type of the value that the pointer will point to.

Example:

```
int *p; // p is a pointer to an integer
float *pch;
char *pfnum;
```

→ pointer variable can hold only address of other variables.

(8)

→ The above three pointers are called dangling pointers because pointers are not specifically pointing to any particular variable.

Ex:
```
#include <stdio.h>
int main() {
printf ("\n the size of short integers is : %d",
    sizeof(short int));
printf ("\n the size of unsigned integers is : %d",
    sizeof (unsigned int));
printf ("\n the size of signed integers is : %d",
    sizeof (signed int));
return 0;
}
```

output:

The size of short integers is : 2
The size of unsigned integer is : 2
The size of signed integers is : 2

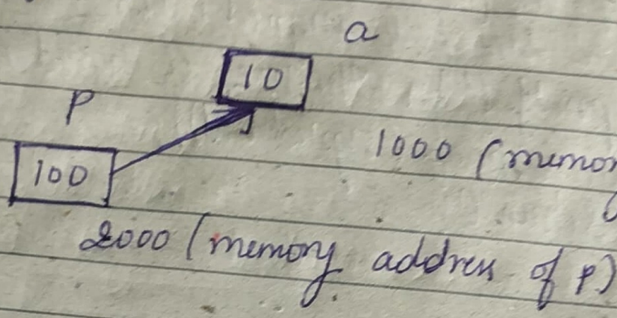
```
#include <stdio.h>
int main() {
  int num, *pnum;
  pnum = &num;
  printf ("\n enter the numbers : ");
  scanf ("%d", &num);
  printf ("\n the number that was entered is : %d",
  *pnum);
  printf ("\n the address of number in memory
  is : %p", &num);
  return 0;
}
```

output: Enter the number : 10
The ~~total~~ number that was entered is 10
The address of number in memory is : FFDC

Initialization:
process of assigning an address to the pointer

Example:
```
int a=10;
int *p;
p=&a;
```

memory representation:



Accessing variables through pointers

→ Accessing variable's values via pointers uses unary * operator.
→ This is called as indirection or de-referencing operator.

Q8b. Explain pass by value and pass by address with example (04 M)
← already covered in the previous module-3 →

Q8c. Write a C program using pointers to compute sum, mean, standard deviation of all elements stored in an array of n real numbers
← Refer the Lab program no 11 →

(10)

Study the lab program number 10. —

Q8 a. Write a program to copy and concatenate from one string to another. (8M)

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[100], str2[100], concatenated[200];

    // Input strings.
    printf("Enter the first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = '\0';
                // Remove newline character.

    printf("Enter the second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = '\0';
                // Remove newline character.

    // copy str1 to concatenated.
    strcpy(concatenated, str1);
    // concatenated str2 to concatenated.
    strcat(concatenated, str2);
    // Display results
    printf("First string: %s\n", str1);
    printf("Second string: %s\n", str2);
    printf("Concatenated string: %s\n",
                concatenated);

    return 0;
}
```

(N)

Explanation:

1. Copying : The strcpy() function is used to copy the contents of str1 into the concatenated array
2. Concatenation : The strcat() function appends the contents of str2 to the end of concatenated.
3. Input Handling : fgets() is used for safe input and strcspn() removes the trailing newline character.

Output:

Enter the first string : Indhu
Enter the second string : Meow
First string : Indhu
Second string : Meow
Concatenated string : IndhuMeow.

8b Explain any 6 String Manipulation functions (6M)

← Already covered in the previous page →

5. Strcpy() function:

char *strcpy(char *str1, const char *str2);

This function copies the string pointed to by str2 to str1 including the null character of str2. It returns the argument str1.

6. Strlen() function:

size_t strlen(const char *str);

This function calculates the length of the string str upto but not including the null character, i.e the function returns the number of characters in the string.