Module - 4

Function and Recursion

Function :-

Block of code that perform particular task

OR

Function is a set of statement that take input, do some specific computation and produce output

Take argument or input parameter → Do work → return result

Advantage :-

(1) It can be used multiple time
(2) increase ↑ code reusability
(3) reduce ↓ the size of the program
(4) make our code → clear, readable

disadvantage :

(1) It can't return multiple values.

imp page 192 (PPS)

Q6, Q7, Q8, Q9

user-defined functions	library functions
(1) These function are not predefined in the compiler.	(1) These function are predefined in the compiler of C languages
(2) user-defined function are not stored in library files	(2) library function are stored in library files
(3) These function are created by user as per their own requirements.	(3) These function are not created by users as their own.
(4) There is no such kind of requirement ⇓ add a particular library	(4) If the user want to use a particular library function ⇓ then the user has to add the particular library function ⇓ The Header file of the Program
(5) Example : sum(), fact() ... etc	(5) Ex :- printf(), scanf() sqrt() .... etc.

Explained ?

(1) How to declare function :-

Function prototype

- To declare a function in C

You specify the → return type, function name and parameters (if any)

Ex :-

Syntax

```
int add (int a , int b);
```

① declares a function name "add"

② That take two integer parameter

③ return an integer.

(2) Function Call :-

To call a function

↳ use it named followed with parentheses containing the arguments (value)

Ex :-

```
int main()
{
    result = add (3, 5);
}
```

That argument called as actual argument

- calls the "add" function
- with argument '3' and '5' → assign the result to a variable named 'result'

(3) **Function definition :**

- This is the function definition for "add"
- It specifies that the function
  - takes two integer (int a, int b) as parameters
- and return their sum.
- The actual implementation of the function
  - The addition operation 'return a+b;'

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

with prototype / declaration

```
#include<stdio.h>
// function prototype
int add(int a, int b);

int main(){
    int result = add(3, 5); // function call
    printf("result : %d\n", result);
    return 0;
}

// function definition
int add(int a, int b){
    return a+b;
}
```

without prototype / declaration

```
#include<stdio.h>
// function definition
int add(int a, int b){
    return a+b;
}

int main(){
    // function call
    int result = add(3,5);
    printf("result : %d\n", result);
    return 0;
}
```

Type of function

add two number with all categories:

(1) Function without arguments and without return value

```c
#include<stdio.h>
void add();   // function declaration
int main(){
    add();    // function call
    return 0;
}

void add(){   // function definition
    int a, b, sum;
    printf("Enter two number\n");
    scanf("%d %d", &a, &b);
    sum = a+b;
    printf("Addition =%d\n", sum);
}
```

Syntax :- (1) Function_name()
{ valid code;
}

No argument

Concept of No return direct go to definition

pura pura kaam main function me nhi kiya hum definition me kiya hai when no argument

(2) Function with arguments and No return value

```c
#include <stdio.h>
void add(int, int);   // function declaration
int main(){
    int a, b          // function call
    printf("Enter two numbers\n");
    scanf("%d %d", &a, &b);
    add(a, b);
    return 0;
}
void add(int a, int b){
    int sum;
    sum = a+b;
    printf("Addition =%d\n", sum);
}
```

no return

with argument

(2) Syntax: Function_name (argument1, argument2, ..., argumentn)
{ valid code;
}

Syntax :-
3) data_type function_name ( ) ;
{ valid c code ;
}

(3) Function with argument and with return value

```
#include<stdio.h>
int add ( ) ;          // function declaration
int main( ) {
int sum ;
sum = add ( ) ;
Printf ( " Addition = %d \n " , sum ) ;
return 0 ;
}
int add ( ) {
int a , b , c
printf ( " Enter two numbers \n " ) ;
Scanf ( " %d %d " , &a , &b )
c = a + b ;
return c ( ) ;
}
```

c value sum ko saara add ho ge

when return the value

Syntax :
4) data_type function_name (arg1 , arg2 , ... argn) ;
{ valide c code ;
}

(4) Function with argument and with return value

```
#include<stdio.h>
int add ( int , int ) ;
int main ( ) {
int a , b , sum ;
Printf ( " Enter the value two number \n " ) ;
Scanf ( " %d %d " , &a , &b ) ;
sum = add ( a , b ) ;
Printf ( " Addition = %d \n " , sum ) ;
return 0 ;
}
int add ( int a , int b ) {
{
int c ;
c = a + b ;
return c ( ) ;
}
```

then terminate the program

Imp

Call by value	Call by reference
(1) When we pass value as an argument to a function it is called call by value	(1) When we pass address as an argument to a function it is called call by reference
(2) example void add(int, int);	(2) Example void add(int*, int*); (pointer)
(3) when we call a function (using) ⇒ call by value, then the copy of variable is passed to the function So, if there is any change to the variable in that function, then the original value is not change	(3) when we call a function (using) ⇒ call by reference then the original value is passed to the function So, if there is any change there, it change the original value as well.
(4) Actual and formal argument are created at the different memory location	(4) Actual and formal argument are created at the same memory location

[2M]

Function Name :- Every function in C will have a name.

- using this name we can call the function and it has parameters

parameters :- When we call a function → to perform a task

- These are called actual parameter or argument we may pass the parameters
- The argument will have a data type
- The parameter that are passed to a function are optional.

When calling a function the parameter passed ~~if the function declaration~~

Actual Parameter

→ ① The parameter passed to a function

② declare in main function

e.g res = add(a, b)

Formal Parameter

→ ① The paramter receive by a function

② declare in function's define

e.g int add

The parameter passed in the ~~calling~~ function declaration

Swapping of two number using → Call by value and Call by reference

Call by value

```
#include<stdio.h>            -> function name swap
void swap (int , int );
int main(){
int a= 5 , b= 10;
printf("Before swapping - a= %d , b= %d\n"
     ,a,b);
swap(a,b); // function call
printf(" After swapping - a=%d , b= %d\n",a,b);
}
void swap (int x, int y) {
  int temp=0;
   temp = x ;
   x = y ;
   y = temp ;
   printf("x = %d , y = %d\n", x, y);
```

mhujhe return koi value nhi jaayeg bas two no. ko swap kar de

a=5 → x=5

b=10 → y=10

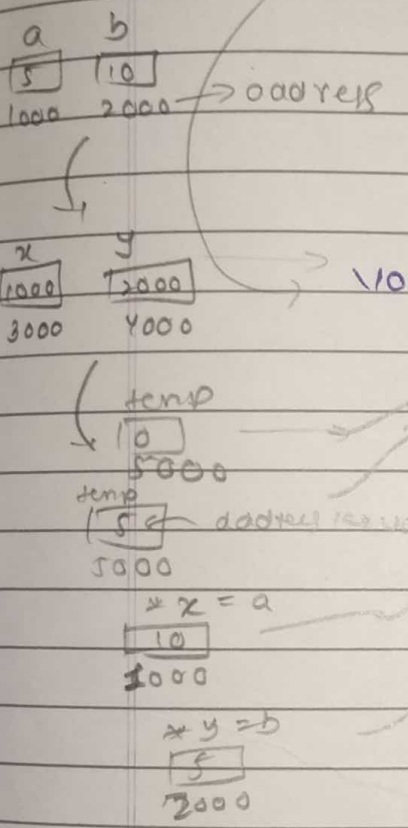
output :

Before swapping : a=5 b=10

After swapping : a=5 b=10

call by reference

```
#include <stdio.h>
void swap (int * , int * );
int main() {
int a=5 , b=10 ;
printf("Before swapping - a=%d , b=%d\n",a,b);
swap (&a,&b) ; // function call
printf("After swapping - a=%d, b=%d\n",a,b);
}
void swap ( int* x, int* y ) {
int temp=0;
temp = *x;
*x = *y;
*y = temp;
printf("x=%d, y=%d\n", x, y );
}
```



output :

Before Swapping : a=5 b=10

After swapping : a=10 b=5

⑤ It is used to solve mathemical, trignometric or any type of algebric problem

⑥ Reduce unnecessary calling of function

Recursion:

when a function call itself, then it's called recursion.

Syntax:

For example:-

```
int main(){
    recursion();
}
```
→ function call

function definition →
```
void recursion() {
    recursion(); /* Function calls itself */
}
```

Advantage: ① Recursion is very simple and easy to understand

② It required lesser number of program statement.

③ It is more useful in multi programming and multitasking environment

④ It is useful in soluing the data structure problem like, linked list, Queues, Stack

disadvantage:

① It consumes more storage space.

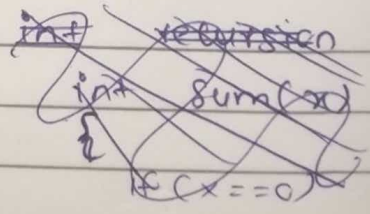
② If recursion is not a stop condition then it is creats an indefinite loop.

③ It is more time consuming process.

④ It is not efficient

recursion e.g

```
int fact(int n)
{
  if (n==0)
    return 1;
  else
    return n* factor(n-1);
}
```



through → Base case → Break the recursion

write a program to Find Factorial of number

```
# include <stdio.h>
int Factorial (int n) ;        // function declaration

int main() {
    int num, answer ;

    printf ("Enter a number : ")
    scanf("%d", &num);

    answer = Factorial(num);   // function call

    printf("Factorial of %d is : %d\n", answer);
    return 0;
}

int factorial (int n) {
if (n==0 || n==1) ;if (n==0) {      // Base case : factorial of 0 is 1
        return 1;
    } else {
// defined in terms itself    return n* Factorial(n-1);
    }
}
```

output

Enter a number : 5

Factorial of 5 is : 120

Next term = Sum two previous terms

term1
term2

fixed { 0 → 0+1=1
1 → 1+1=2
1 → 1+2=3
2 → 1+2=3
3
4
5

Fibonacci Series :

```
# include<stdio.h>
```

0 , 1 , 1 , 2 , 3 , 5

0th term, 1st term, 2nd term

Base case
last case

formula

fib(n) = fib(n-1) + fib(n-2);

Base case

```
if(n==0)
    return(0);
else if(n==1)
    return(1);
else
    return fib(n-1)+fib(n-2);
}
```

1st term fixed is one

0th term fixed is zero

1st term fixed is one

one = one

0 1 1 2 3 5

```
# include<stdio.h>
int Fib(int);
int main(){
int n, i, f;
Print("Enter value of n");
Scanf("%d", &n);
For(i=2; i<=n; i++){
    f = fib(i); // function call
Printf("%d\n", f);
    }
}
```

```
int Fib(int n){
if (n==0)  // Base case
    return 0;
else
if (n==1)  // Base case
    return 1;
else
return fib(n-1)+fib(n-2);
}
```

output Enter the value of n: 5

0 1 1 2 3

Sum of 1-N number :-

```
#include<stdio.h>
int sum(int);
int main(){
  int num, add;
  printf("Enter the value of num:");
  scanf("%d", &num);
  add = sum(num);
  printf("%d\n", add);
}
int sum(int n);
if (n==0){
    return 0;
}
else
    return n+sum(n-1);
}
```

output : Enter the value of num : 5
sum of n number: 15

sum(5) = 5 + 4 + 3 + 2 + 1

n + sum(n-1)

sum(5) = 5 + 4 + 3 + 2 + 1 = 5 + sum(4)

sum(4) = 4 + 3 + 2 + 1 = 4 + sum(3)

sum(3) = 3 + 2 + 1 = 3 + sum(2)

sum(2) = 2 + 1 = 2 + sum(1)

sum(1) = 1 + 0 = 1 + sum(0)

sum(0) = 0

```
if (n==0){
  return 0;
}else
  return (n+sum(n-1));
}
```

$LCM = \frac{n_1 * n_2}{GCD}$ Highest Common Factor

Greatest Common divisor HCF or GCD

Develop a code to perform GCD using recursive function using Euclid's method

e.g 12, 8

1st step = Highest input / lowest input ; lowest input

Remainder

$= \frac{12}{8}, 8$

Remainder 1st $= 4, 8$

2nd $= \frac{8}{4}, 4$

Remainder $= 0, 4$

that will be a GCD

e.g 6, 18

1, 2, 3, 6 → GCD

$6/6 = \underline{6} \times 1$

$18/6 = \underline{6} \times 3 \times 1$

e.g 2, 4

1, 2, 4 → GCD

$2/2 = \underline{2} \times 1$

$4/2 = \underline{2} \times \underline{2} \times 1$

(3) eg a=20, b=40

20 = 1, 2, 4, 5, 10, 20

40 = 1, 2, 4, 5, 10, 20, 40

20 = GCD

$n_1 > n_2$ → gcd (n_1 % n_2 , n_2);

$n_2 > n_1$ → gcd (n_1, n_2 % n_1);

(If n_1 == 0 then gcd = n_2;

If n_2 == 0 then gcd = n_1;)

```
#include<stdio.h>
int gcd(int, int);
int main(){
  int n1, n2;
  printf("Enter 2 positive integer no:\n");
  scanf("%d %d", &n1, &n2);
  printf("GCD is : %d\n", n1, n2,
         gcd(n1, n2));
}
```

lcm = n1 * n2 / gcd

```
int gcd(int N1, int N2){
  if(N1 == 0)
    return N2;
  if(N2 == 0)
    return N1;
  if(N1 > N2)
    return gcd(N1 % N2, N2);
  Else (N2 > N1)
    return gcd(N1, N2 % N1);
}
```

output :-

Enter the two the Integer no:

→ 5

→ 15

gcd is : 5

terminated when the base condition is fulfill

terminated when the loop condition is fail.

Recursion	Iteration
(1) Function call itself	(1) Function loops some part of code
(2) Recursion is selection structure	(2) Iteration is repetition structure
(3) Recursion terminates when a base case is recognized	(3) Iteration terminates when the loop-continuation condition fail
(4) recursion return value to the calling function	(4) Iteration doesn't return value
(5) Recursion makes code small	(5) Iteration makes code longer
(6) Recursion → is slow process	(6) Iteration → is Fast process

ex:-

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

output Enter the value 5
is 120

ex.

```
#include<stdio.h>
int main(){
    int i, n=5, fact=1;
for (i=1; i<=5; ++i)
    Fact = Fact * i;
Printf("factorial for 5 is %d", fact);
return 0;
}
```

output

Factorial 5 is 120.

Syntax :- int main() {
recursion();
}
void recursion() {
recursion();
}

Syntax for (initialization, condition, increment/decrement)
{ Code }

(5M)

Recursion

(1) implemented using function call

(2) terminated when the Base condition fail-filled

(3) Infinite loop is there if condition never fails

(4) Recursion is Selection Structure

(5) Make our code small

(6) is Slow Process

(7) it's require more Storage Space

(8) Syntax

```
#include <stdio.h>
void swap(
Void Fact(int);
int main(){
int Factorial, n
Printf("Enter the no. :");
Scanf("%d", &n);
Factorial = Fact(n)
printf("factorial is %d", factorial);
return 0;
}
void fact (int n)
int
if (n==0)
return 1;
else
return n* fact(n-1);
}
```

Iteration

(1) implemented using loop

(2) terminated when the loop condition fail

(3) Infinite recursion is there if no Base condition

(4) Recursion is Repetition structure

(5) make our code larger

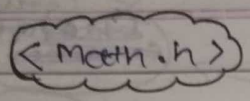
(6) is Fast process

(7) it's ~~decrease~~ require less Storage Space

```
#include <stdio.h>
void fact
int main()
{
int i, n=5, fact=1;
for(i=1; i<=5; i++)
{
fact= fact* i
Printf("factorial is %d", fact);
}
return 0;
}
```

most most Imp In ISE

Library function



- "math.h" → Header file supports all the mathematical related function in c language

- The <math.h> header file contain various method for performing mathematical operator.

  Sqrt(),
  Pow(),
  ceil(),
  floor() etc.

- Sqrt() - This function is used to Find the square root of a number
  Pow() - —11— to Find the power raised to that number.
  Fabs() - —11— to Find the absolute value of a no.
  log() - —11— to Find the logorithm value of no. with Base e
  Sine() - the trignometric function is used to Find the Sine value of number
  Cos() - —11— Find the cosine value of number
  tan() - —11— Find the tan value of number.

<ctype.h>

"ctype.h"

- This Header file is used for → function conversion and function character testing purpose

- Some of the function associated with <ctype.h> are :

- isalpha() - wont to check if the character is on alphabet or not
- isdigit() - used to check if the character is a digit or not
- isalnum() - used to check if the character is alphanumeric or not
- isupper() - used to check if the character is in uppercase or not
- islower() - wont to check if the character is in lowercase or not
- toupper() - wont to convert the character into uppercase
- tolower() - wont to convert the character into lowercase

Storage classes

- Storage classes in C programming are used to explain → the properties of the declared variables and function

- There are Four properties by which storge class of a variable can be recognized

These are Storage, default initial value, Scope and lifetime

- These properties help a programmer to know the lifetime and availability of a variable in a Scope.

C programming provides four type of storage classes.

(1) Automatic
(2) Register → Register
(3) Static
(4) External

RAM

Imp (1) Automatic storage classes mienticned:

(2) – All variables defined in a function or Block belong to the automatic storage classes.

(1) – Variables defined in function or block with automatic specifier belong to the automatic storage classes.

By default.

– Features of automatic storage classes

Declaration : Auto int a; or int a; (auto variable / local variable)

Default Value : Garbage value — If int a=10; int a; ✗ value when not initialized

Storage : primary memory

Scope : local to Block

An automatic/local variable can access within the Block in which variable is declared

{ access within the Block local variable }

Life time : local to Block

e.g

```
#include<stdio.h>
int main()
{
    auto int a=1;
    {
        auto int b=2;
        {
            auto int c=3;
            printf("\n%d", c);
        }
        printf("\n%d", b);
    }
    printf("\n%d", a);
}
```

output :
3
2
1

(2) Register storage classes : required to compiler agar koi CPU me free register present hai to hum apne variable ko main memory NO storage kar ke kaise CPU register storage karde ku ke kam time me hum variable ko jaldi access kar sake

- A register storage class tell the compiler to store a variable in such a way that access to it as fast as possible

- register specifer declares the variable of the register storage class

- variable belonging to the register storage classes

- Feauture of Register storage classes

Declaration :	register int a;
Default value :	Garbage value
Storge :	cpu register
Scope :	local to Block
life time :	local to Block

A ~~Register~~ ~~static~~ variable can access within the block in which variable is declared

e.g

```
# include <stdio.h>
int main()
{
register int a; // variable (a) is allocated memory in the CPU register.
                initial default value is (zero).
print("%d", a);
}
```

output : 0

(3) Static Storage class :

- The static storage class instructs
  → The compiler to keep a local variable existence during the life-time of the program.

therefore,

- making local variables static allows them to maintain their value b/w function call.

- The static modifier may also be applied to global variables

- Feature of static storage class :

Declaration : static int a;

Default Value : 0

Storage : primary memory

Scope : local to Block. A static variable can access within the block in which variable is declared.

Lifetime : Entire program

e.g

```
#include <stdio.h>
void staticDemo();
int main(){
    staticDemo();
    staticDemo();
}
void staticDemo()
{
    int a=1;
    static int b=1;
    printf("%d %d", a, b);
    a++;  b++;
}
```

1st time function call

① a=1 ✗ b=1

② destroy the value one and again because variable (a) not use static

③ a=1 ✗ (not use static) b=2 (variable is static)

④ a=1 ✗ b=3

output

a=1 , b=3

(4) External storage class :

- extern is used to declare → global variable or function → in another file
- the extern modifier is most commonly used when there are two or more file sharing → the same global variable or function
- Feature of ~~static~~ extern storage class

Declaration : extern int a;

Default value : 0

Storage : Primary memory

Scope : Global . extern variable can be access anywhere in ~~any~~ file

life time : Entire program

e.g

File 1 : main.c

```
#include<stdio.h>
int a;
extern void fun();
void main()
{
    a=15;
    fun();
}
```

File 2 : Support.c

```
#include <stdio.h>
extern int a;
void fun();
void fun()
{
printf("%d\n", a);
}
```

global variable

Big program ko different different file me divide kar deta hai

abad memory

datatype variable_name;

(4) Syntax :- auto datatype variable_name;

auto int a;

request karna compiler ko

auto :-

(1) a variable defined in function or Block with automatic specifiers

Belong

to the automatic storage class

e.g int a;

(2) If NO storage classes is mentioned

all variable defined in function or Block

Belong

feature of automatic storage

declaration : auto int a;
int a; to the automatic storage class

Default value : Garbage value By Default variable.

(3) automatic variable are simply local variable

(3) destroyed when the block is exited

e.g
```
#include <stdio.h>
int main()
{
    auto int a=1;
    {
        auto int b=2;
        {
            auto int c=3;
            printf("\n%d", c);
        }
        printf("\n%d", b);
    }
    printf("\n%d", a);
}
```

output = 321

Storage : RAM, Primary memory

Scope :
local to Block
an automatic / local variable
can access within a Block
in which variable is declared.

Life time :
local to Block

private

request to the compiler, if possible then make variable as a register variable

register :- (1) A register storage class

Tell the compiler

to store the variable in such a way that access as it fast as possible

(2) destroyed when the block is exited

(3) a variable defined in Block with ~~automc~~ register specifier.

Belong to the register storage class

are local in the Block in which they are define

(4) Syntax : register datatype variable_name;

e.g register int a;

feature

declaration : register int a;

default value : Garbage value

storage : CPU

scope :

local to Block

A register variable can access within the Block in which variable is declared

life time :

local to Block

e.g

```
#include<stdio.h>
int main()
{
    register int a;  // variable (a) is allocated memory in the CPU register.
printf("%d",a);
}
```

output : 0. the initial default value of (a) is 0

(3) **Static** :

① Static modifier ↑ may be also is applied to Global variable

② Static Storage class

↓

instruct the compiler

↓

to keep the local variable

↓

existence during life-time program.

③ they are created only at once throughout the program remain in till end of the program.

Syntax :- Static data type variable name

Features

declaration : static int a;

default value : 0

Storage : primary memory / RAM

Scope : local to Block

A static variable can access within the blocks in which variable is declared

life time :

Entire program

e.g

```
#include<stdio.h>
void fun();
void main()
{
    fun();
    fun();
    fun();
}
void fun()
{
    int a=1;
    static int b=1;
    printf("%d %d", a, b);
    a++; b++;
}
```

- static int b create memory only once and use it entire the program

- during the program int a exist within B

But

- static int b exist entire the program

not for local variable

only use for global variable

External

↳ used to Tell the complier

to that variable is

declare some where else

feature

declaration : extern int b;

default value : 0

Storage : RAM

Scope : global

life time :

entire program

e.g

```
int x; // global
void f(){
    x++;
    printf("%d", x);
}
void g(x){
    x++;
    printf("%d", x);
}
void main()
{
++x;
printf("%d", x);
}
```

e.g

```
#include <stdio.h>
extern int x;
int main()
{
Printf("x: %d", x);
}
```