Date. No.

Why is C-programming language called structured programming languge?

Ans:

C- programming language is called structured programming language because C- programming gives emphasis to process and it has basic control structures which control the flow of execution of program.

Why is C-programming language called modular programming language?

Ans:

C- programming language is also called modular programming language because a large program can be divided into semi-independent modules ie, logical function modules using function procedure each of which can be called when required.

Function:

Number of statements grouped into single logical unit to perform specific task is called function.

In C- programming, main() is the function first executed and all other functions are first executed from main.

Date. No.

(*) Types:

Functions are of two types: built-in function and ~~library function~~. user-defined function.

a) Built-in function:

→ Library functions that are provided by the system

→ This is read by compiles directly and executed.

b) User-defined function:

→ functions created by the user.

(*) main ( ) is user-defined function but is it is standard declaration.

(*) Advantages of Function:

(i): It helps us to reduce redundancy in the program.
(ii): It makes debugging programs easy.
(iii): It is efficient and helps in better memory utilization.

(*) General Syntax:

returntype functionname (~~argument~~ parameters); → declaring function.

returntype functionname (parameters)
{
function body
return statement
} → defining function

functionname ~~name~~ (arguments); } → calling functions.

Date. No.

(*) Parts of a function:

A function is divided into three parts: function declaration, function definition and function call.

(a): function declaration:

Syntax: returntype functionname (parameters);

→ declaring a function helps compiles know the type of function.
→ Mostly, the functions are declared outside main to increase its scope.

While declaring, it is not necessary to give name of parameters but their datatype should be specified.
Eg: void fun (void);
void fun (int, int);

While declaring or defining a function, if no returntype is provided, it is default fixed to int.

So, if we want some value returned, we must declare a function.

→ semi-colon end is required while declaring function.

Date No.

(b) function definition:

Syntax: returntype functionname (parameters)
{
function body
return statement
}

A function is defined in this way.
Based on the returntype of the function, we have to place the return statement.

For ~~eit~~ function with no return type, closing brackes acts as return.

While defining function, both datatype of parameters and their name must be written.
Eg: void fun(void);
int fun (int a, int b).

(*) The ~~data~~ parameters passed during declaring and defining are called formal parameters.

(C): function call:
Syntax: function name (arguments);

Here, the name of arguments in declared order must be written. and there is no need to specify datatype while calling.

Date No.

We can call a function by value or by reference.
In calling by reference, we pass the address of values.

(*) Classification of functions:
Functions are classified into four types. They are as follows:
(i) No argument without returntype
(ii) No argument with return type
(iii) Argument without returntype
(iv) Argument with returntype.

(i) No argument without returntype:
Syntax:
```
void fun (void);
void fun ( )
{
  function body
}
fun( );
```
Here, the function takes no arguments ie, void and also returns no value.
Here, for a function, the closing braces acts as return statement.

(ii) No argument with return type:
Syntax:
```
int fun (void);
int fun( )
{
  function body
  return statement }
fun( );
```
Here, the function takes no arguments ie, void but returns integer value as specified.
Here, return statement is used.

Date. No.

(iii) Argument without Returntype:

Syntax: void fun(int , int);
void fun (int a, int b);
{
function body }
fun(x,y);

Here, the function returns no function but we are passing two integers values as arguments.

(iv) Argument with returntype:

Syntax: int fun (int , int);
int fun (int a, int b);
{
function body
return statement
}
fun(x,y);

Here, the function returns integer value and we have also called a function by passing two integers values.

Recursion in C:

When a function calls itself, it is called recursion.

Eg: void add ()
{
add();
}

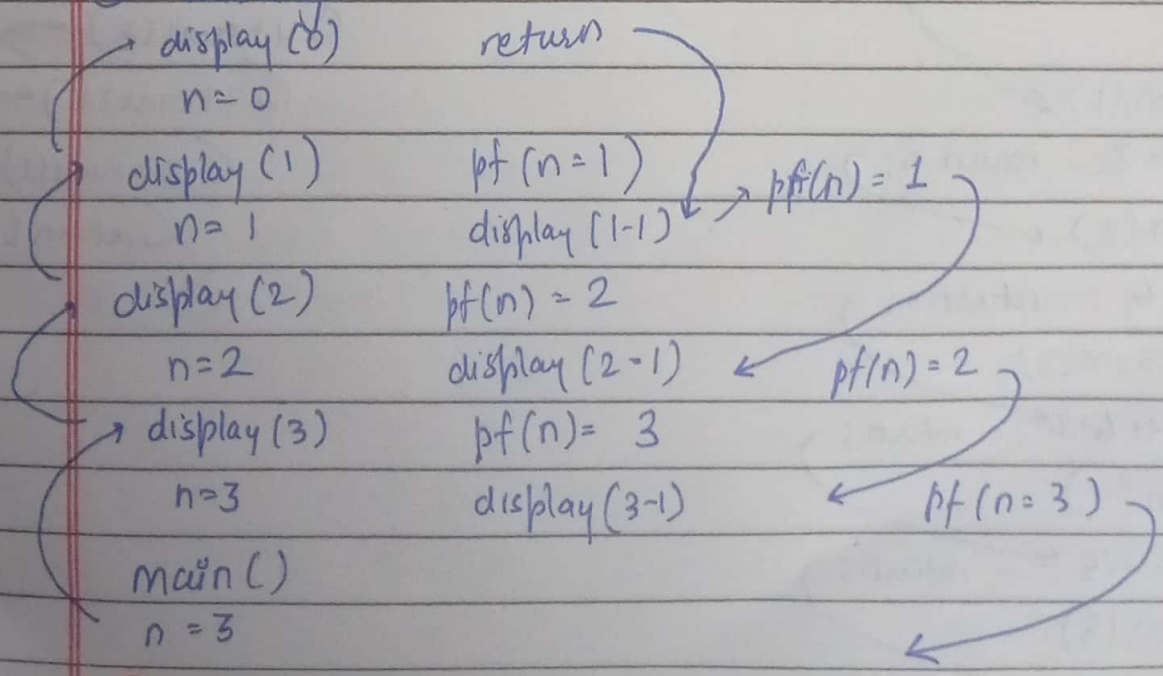
(*) Base condition: Condition to terminate ~~base~~ recursion is base condition. If base condition is not properly specified, it causes stack overflow.

Date. No.

Eg: (i):
```
#include <stdio.h>
void display(int );
void main()
{
 int n
 scanf("%d", &n);
  display(n);
}
void display(int n);
{
 if (n<1) return;
 else
 { printf("%d", n)
  display(n-1);
  printf("%d", n)
   }
}
```

* Output:
3 2 1 1 2 3

(*) Working:



Date. No.

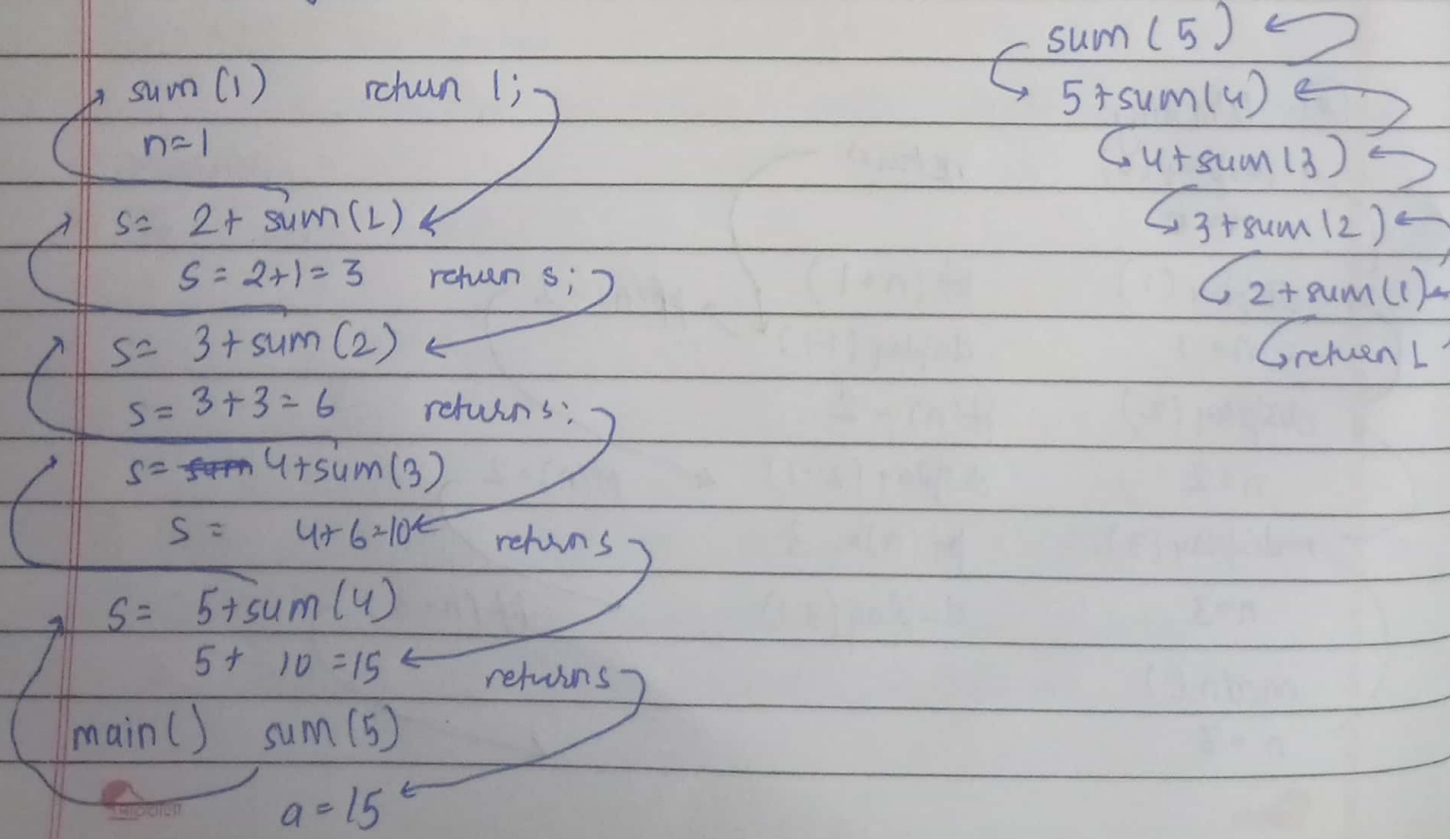
(ii): #include <stdio.h>

```
int sum (int x);
void main()
{
  int a; int n=5
  a= sum(n)
  printf("%d", a);
}
int sum (int n)
{
  int s=0;
  if (n==1) return n;
  else  s = n+sum(n-1);
  return s;
}
```

(*) Output:

15

(*) Working:



Date. No.

(*) Types of Recursion:

(i): Direct recursion: A function that calls itself directly is called direct recursion.

Syntax:
```
fun()
{ ....
  fun()
  .... }
}
```
Here, fun calls fun directly.

(ii) Indirect recursion: A function that calls itself indirectly through other functions is called indirect recursion.

Syntax:
```
fun1()
{ ....
  fun2(); ....
}
fun2()
{ ....
  fun1(); ....;
}
```
Here, fun1 calls fun2 which again calls fun1.

(iii) Tail recursion:
The function having no task after recursive call.

Syntax:
```
fun()
{ ....
  ....
  fun();
}
```

(iv) Non-tail recursion
→ The function having tasks after recursive call.

Syntax:
```
fun()
{ ....
  fun();
  ....
}.
```