

✔ UNIT – II : Top-Down Design & Pointers in C

1. Top-Down Design with Functions

Introduction

Top-Down Design is a **programming approach** where a large problem is divided into **smaller sub-problems (functions)**.

Each function performs a **specific task**, making the program **easy to understand, test, and maintain**.

Advantages

- Reduces program complexity
 - Easy debugging
 - Improves readability
 - Code reusability
 - Modular programming
-

2. Building Programs from Existing Information

Meaning

Programs can be developed using:

- Previously written logic
- Library functions
- User-defined functions

This saves **time and effort**.

Example

Using printf() instead of writing your own display function.

3. Library Functions

Definition

Library functions are **predefined functions** available in header files.

Common Examples

Function Header File Purpose

printf()	stdio.h	Output
scanf()	stdio.h	Input
strlen()	string.h	Length of string
sqrt()	math.h	Square root

Example

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

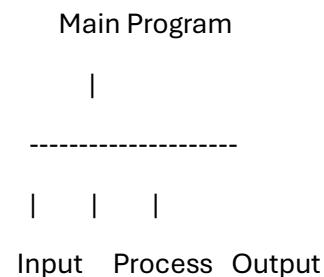
```
int main() {  
    printf("Hello World");  
    return 0;  
}
```

4. Top-Down Design and Structure Charts

Top-Down Design

The main problem is broken into **smaller modules**.

Structure Chart (Text Diagram)



Advantages

- Easy debugging
- Better planning
- Code reuse

5. Functions without Arguments

Definition

Functions that **do not receive values** from calling function.

Syntax

```
void functionName()  
{  
    statements;  
}
```

Example

```
#include <stdio.h>  
  
void display() {  
    printf("Hello World");  
}  
  
int main() {  
    display();  
    return 0;  
}
```

6. Functions with Input Arguments

Definition

Functions that **receive values** from the calling function.

Syntax

```
void add(int a, int b)  
{  
    printf("%d", a + b);  
}
```

Example

```
int main() {  
    add(10, 20);  
    return 0;  
}
```

7. Pointers and Indirection Operator

Pointer

A pointer is a variable that stores the **address of another variable**.

Declaration

```
int *p;
```

Indirection Operator (*)

Used to access the value at an address.

Example

```
int a = 10;
```

```
int *p = &a;
```

```
printf("%d", *p); // prints 10
```

8. Functions with Output Parameters

Definition

Functions can return values using **pointers**.

Example

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

```
int main() {  
    int result;  
    add(5, 10, &result);  
    printf("%d", result);  
}
```

9. Multiple Calls to a Function with Input / Output Parameters

Example

```
void square(int x, int *result) {  
    *result = x * x;
```

```
}
```

```
int main() {  
    int res;  
    square(2, &res);  
    printf("%d\n", res);  
  
    square(4, &res);  
    printf("%d", res);  
}
```

10. Scope of Names

Definition

Scope defines **where a variable can be accessed**.

Types

1. Local variables
2. Global variables

Example

```
int x = 10; // Global
```

```
void fun() {  
    int y = 5; // Local  
}
```

11. Formal Output Parameters as Actual Arguments

Explanation

- Formal parameters → parameters in function definition
- Actual parameters → values passed during function call

Example

```
void change(int *a) {  
    *a = 20;
```

```
}
```

```
int main() {  
    int x = 10;  
    change(&x);  
    printf("%d", x);  
}
```

Text Diagram: Function Call Flow

Main Function

|

v

Function Call

|

Parameters Passed

|

Function Executes

|

Result Returned

Important Keywords (For Exams)

- Modular programming
- Top-down approach
- Function prototype
- Actual parameters
- Formal parameters
- Pointer
- Indirection operator
- Scope of variables