Format Specifiers:

Format specifiers are used in `printf()` and `scanf()` to tell the compiler the type of data to be printed or read.

Specifier	Meaning	Example Code	Output
`%d`	Integer	`printf("Age = %d", 23);`	Age = 23
`%f`	Float	`printf("PI = %f", 3.14);`	PI = 3.140000
`%lf`	Double	`printf("Value = %lf", 12.3456);`	Value = 12.345600
`%c`	Character	`printf("Grade = %c", 'A');`	Grade = A
`%s`	String	`printf("Name = %s", "Neeraj");`	Name = Neeraj
`%u`	Unsigned int	`printf("%u", 25);`	25
`%x`	Hexadecimal	`printf("%x", 15);`	f
`%%`	Percent sign	`printf("Discount = 50%%");`	Discount = 50%

Escape Sequence Characters:

Escape sequences are used inside strings (`""`) or characters (`''`) to represent special characters.

Escape Sequence	Meaning	Example Code	Output
`\n`	New line	`printf("Hello\nWorld");`	Hello World
`\t`	Tab space	`printf("A\tB\tC");`	A B C
`\\`	Backslash	`printf("C:\\Program Files");`	C:\Program Files
`\"`	Double quote	`printf("He said \"Hello\"");`	He said "Hello"
`\'`	Single quote	`printf("It\'s OK");`	It's OK
`\0`	Null character	Used in strings → `"Hello\0World"`	Prints only → Hello

Control Statements in C Language

1. **If – Else Statements**
2. **Switch case statements**

If-Else Statements

1. If Statement

Used to execute a block of code **only if the condition is true**.

Example:

```
int age = 20;
if (age >= 18) {
    printf("You are eligible to vote.");
}
```

Output:

```
You are eligible to vote.
```

2. If – Else Statement

Executes one block if condition is true, otherwise executes another block.

Syntax:

```
if (condition) {
    // code if true
} else {
    // code if false
}
```

Example:

```
int marks = 35;
if (marks >= 40) {
    printf("Pass");
} else {
    printf("Fail");
}
```

Output:

```
Fail
```

3. If – Else If – Else Ladder

Used when we have **multiple conditions**.
Conditions are checked **top to bottom**, and the first true condition executes.

Example:

```
int marks = 75;

if (marks >= 90) {
    printf("Grade A");
} else if (marks >= 75) {
    printf("Grade B");
} else if (marks >= 50) {
    printf("Grade C");
} else {
    printf("Fail");
}
```

Output:

```
Grade B
```

4. Nested If Statement

An `if` statement **inside another if**.

Example:

```
int age = 20;
int citizen = 1;  // 1 = Indian, 0 = Other

if (age >= 18) {
    if (citizen == 1) {
        printf("Eligible for voting in India.");
    } else {
        printf("Not an Indian citizen.");
    }
} else {
    printf("Not eligible (age below 18).");
}
```

Output:

```
Eligible for voting in India.
```

Switch Case Statement

- The **switch statement** is used when we need to execute **one block of code out of many choices**.
- It works with **int, char, enum** types (not with float/double or strings).
- It is an **alternative to multiple if–else if** statements.

Important Points

1. expression is evaluated **once**.
2. The value is compared with each `case`.
3. If a match is found → that block runs.
4. `break;` stops execution and exits the switch.
5. If no case matches → `default` runs (optional).

Example 1: Simple Switch

```
#include <stdio.h>
int main() {
    int day = 3;

    switch(day) {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
            break;
        default:
```

```
            printf("Invalid Day");
    }
    return 0;
}
```

Output:

```
Wednesday
```

Types of Loops in C

1. **while loop**
2. **do…while loop**
3. **for loop**

1. While Loop

Condition is checked **before** executing the code.

Example:

```
int i = 1;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

Output:

```
1 2 3 4 5
```

2. Do…While Loop

Code executes **at least once**, even if the condition is false.

Example:

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
```

Output:

```
1 2 3 4 5
```

3. For Loop

Used when the **number of repetitions is known**.

Example 1: Normal For Loop

```
for (int i = 1; i <= 5; i++) {
    printf("%d ", i);
}
```

Output:

```
1 2 3 4 5
```

Example 2: For Loop with Two Expressions
Initialization can have multiple variables.

```
for (int i = 1, j = 5; i <= 5 && j >= 1; i++, j--) {
    printf("i = %d, j = %d\n", i, j);
}
```

Output:

```
i = 1, j = 5
i = 2, j = 4
i = 3, j = 3
i = 4, j = 2
i = 5, j = 1
```

Break, Continue & Goto Statements

1. break Statement

Used to **terminate** a loop or switch immediately.
Control moves outside the loop/switch.

Use Case:

- When you want to **stop execution** once a certain condition is met.

Example (Loop):

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        break;   // loop will stop when i = 3
    }
    printf("%d ", i);
}
```

Output:

```
1 2
```

2. continue Statement

Used to **skip the current iteration** of a loop.
Control jumps to the **next iteration** of the loop.

Use Case:

- When you want to **ignore specific values** but continue looping.

Example:

```
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;   // skips printing 3
    }
    printf("%d ", i);
}
```

Output:

```
1 2 4 5
```

3. goto Statement

Transfers control to a **label** in the program.
Not recommended (bad practice) because it makes code hard to read, but sometimes used in **error handling** or to **exit nested loops**.

Use Case:

- Breaking out of **multiple nested loops**.
- Handling **errors/exceptions** in old C programs.

Example 1: Simple goto

```
int x = 1;
if (x == 1) {
    goto label;   // jump to label
}
printf("This will be skipped\n");

label:
printf("Goto executed!");
```

Output:

```
Goto executed!
```

Example 2: Exiting Nested Loops with goto

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            goto end;   // exit both loops
        }
        printf("%d %d\n", i, j);
    }
}
end:
printf("Exited from nested loops.");
```

Output:

```
1 1
1 2
1 3
2 1
Exited from nested loops.
```

Summary:

- **break** → stops loop/switch immediately.
- **continue** → skips current iteration, goes to next.
- **goto** → jumps to a labeled statement (use rarely).

What is Typecasting?

Typecasting means **converting one data type into another**.
In C, there are two types:

1. **Implicit Typecasting (Type Conversion)** → done automatically by compiler
2. **Explicit Typecasting (Type Casting)** → done manually by programmer

1. Implicit Typecasting (Type Conversion)

Also called **Type Promotion**.
Compiler automatically converts **smaller type → larger type** to prevent data loss.
Order of promotion:
`char → int → float → double`

Example:

```
int x = 10;
float y = x;   // int automatically converted to float
printf("%f", y);
```

Output:

```
10.000000
```

2. Explicit Typecasting (Type Casting)

Programmer manually converts one type into another using **(type)** operator.

Example 1: int to float

```
int a = 5, b = 2;
float result = (float)a / b;   // forcefully convert int to float
printf("%f", result);
```

Output:

```
2.500000
```

Use Cases of Typecasting

1. When we want **accurate division** (`5/2 → 2` but `(float)5/2 → 2.5`).
2. When handling **mixed type expressions** (int + float).
3. To **control data conversion** manually.

What is a Function?

A function is a **block of code** that performs a specific task.
It helps in **code reusability, modularity, and readability**.

Advantages of Functions:

1. Code reusability → one function can be called multiple times.
2. Easy debugging → errors can be found in a smaller block.
3. Modularity → program is divided into smaller parts.
4. Better readability.

Types of Functions:

1. **Library Functions** (predefined functions)
   - Already defined in header files.
   - Example:
     - `printf()`, `scanf()` → in `<stdio.h>`
     - `sqrt()`, `pow()` → in `<math.h>`
     - `strlen()`, `strcpy()` → in `<string.h>`
2. **User-defined Functions** (created by programmer)
   - Declared and defined by the user.

Parts of a User-defined Function:

1. **Function Declaration (Prototype)** → tells compiler about function name, return type, and parameters.
2. **Function Definition** → actual body (code) of the function.
3. **Function Call** → to execute the function.

General Syntax of a Function:

```
return_type function_name(parameter_list) {
    // function body
}
```

Examples:

Example 1: Function without parameters and without return value

```
#include <stdio.h>
void greet() {              // function definition
    printf("Hello, World!");
}
```

```
int main() {
    greet();                // function call
    return 0;
}
```

Output:

```
Hello, World!
```

Example 2: Function with parameters and with return value

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

int main() {
    int sum = add(5, 10);   // function call
    printf("Sum = %d", sum);
    return 0;
}
```

Output:

```
Sum = 15
```

Example 3: Function with parameters and no return value

```
#include <stdio.h>
void printSquare(int n) {
    printf("Square = %d", n*n);
}

int main() {
    printSquare(6);
    return 0;
}
```

Output:

```
Square = 36
```

Types of User-defined Functions Based on Arguments/Return:

1. No arguments, no return value
2. Arguments, no return value
3. No arguments, return value
4. Arguments, return value