C PROGRAMMING

1. What is C Programming?

C is a general-purpose programming language used to write instructions for a computer. A program can be thought of as a recipe: input is the information supplied to the program, processing is the work performed on that information, and output is the result.

Example: calculating a patient's average temperature or checking whether a biological measurement is within a given range.

```
#include <stdio.h>

int main()
{
    float temperature;

    printf("Enter temperature: ");
    scanf("%f", &temperature);

    printf("Temperature = %.2f", temperature);

    return 0;
}
```

2. Basic Structure of a C Program

```
#include <stdio.h>

int main()
{
    // Declaration
    int age;

    // Executable statements
    printf("Enter age: ");
    scanf("%d", &age);
```

```
    printf("Age = %d", age);

    return 0;
}
```

- Header file: #include <stdio.h> provides standard input/output functions such as printf() and scanf().
- main(): execution of a C program starts from main().
- Declaration section: variables are declared before use.
- Executable statements: instructions that the computer executes.
- return 0: normally indicates successful completion of the program.

3. Comments in C

Comments are notes written for humans. The compiler does not execute them.

```
// This is a single-line comment

/*
   This is a
   multi-line comment
*/
```

4. Header Files

A header file contains declarations needed to use library functions.

- stdio.h: input/output functions such as printf() and scanf().
- math.h: mathematical functions such as sqrt(), pow(), sin() and cos().

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

int main()
{
    printf("%.2f", sqrt(25));
    return 0;
}
```

Output: 5.00

5. Statements and Semicolon

Most C statements end with a semicolon (;).

```
int age;
age = 20;
printf("%d", age);
```

Do not put a semicolon after the condition of an if statement when using a block.

6. printf() Function

printf() is used to display output.

```
printf("Hello");

int age = 20;
printf("%d", age);
```

7. Important Format Specifiers

Data type	Format specifier	Example
int	%d	printf("%d", age);
float	%f	printf("%f", temperature);
double	%f in printf	printf("%f", value);
char	%c	printf("%c", grade);
String	%s	printf("%s", name);

Note: for scanf(), %lf is used for a double.

8. Newline (\n)

\n moves the cursor to the next line.

```
printf("Hello\n");
printf("World");
```

Output:

```
Hello
World
```

9. Formatted Output

```
float amount = 123.45678;
printf("%.2f", amount);
```

Output: 123.46

%.2f means that the number is displayed with two digits after the decimal point. A field width such as %5.2f is a minimum width, not necessarily the total number of characters.

10. scanf() Function

scanf() is used to take input from the user.

```
int age;
scanf("%d", &age);
```

If the user enters 20, the variable age stores 20.

11. Why do we use & in scanf()?

&age gives the memory address of age. scanf() needs that address so it knows where to store the input.

```
int age;
scanf("%d", &age);
```

For multiple values:

```
int age;
float height;

scanf("%d %f", &age, &height);
```

12. printf() vs scanf()

printf()	scanf()
Output function	Input function
Displays data	Reads data
printf("%d", age);	scanf("%d", &age);
Usually does not use & before a normal variable	Usually uses & for ordinary scalar variables

13. Variables

A variable is a named memory location used to store data.

```
int age = 20;
```

- int = data type
- age = variable name
- 20 = stored value

14. Data Types

Type	Purpose	Example
int	Whole numbers	int age = 20;
float	Decimal numbers	float temperature = 37.5;
double	Higher-precision decimal numbers	double value = 12345.6789;
char	One character	char grade = 'A';

A character uses single quotes, e.g. 'A'. A string uses double quotes, e.g. "Hello".

15. Operators in C

Operators are symbols used to perform operations. Examples include +, -, *, /, %, >, <, ==, &, | and ?:.

16. Bitwise Operators

Bitwise operators work on the individual binary bits of integer values. A bit is either 0 or 1.

Operator	Name
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift

17. Bitwise AND (&)

AND produces 1 only when both corresponding bits are 1.

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

```
10 = 1010
 4 = 0100
     ----
     0000

10 & 4 = 0
```

18. Bitwise OR (|)

OR produces 0 only when both corresponding bits are 0.

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

```
10 = 1010
 4 = 0100
     ----
     1110

10 | 4 = 14
```

19. Bitwise XOR (^)

XOR produces 1 when the two corresponding bits are different.

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

Memory trick: XOR = same → 0, different → 1.

```
10 = 1010
 4 = 0100
     ----
     1110

10 ^ 4 = 14
```

20. Bitwise NOT (~)

NOT flips every bit: 0 becomes 1 and 1 becomes 0.

```
int a = 10;
printf("%d", ~a);
```

For the usual signed two's-complement representation, ~10 = -11. A useful identity is ~A = -(A + 1).

21. Left Shift (<<)

Left shift moves bits to the left. For suitable non-negative values, shifting left by one position corresponds to multiplying by 2.

```
10 = 1010
10 << 1 = 10100 = 20
```

22. Right Shift (>>)

Right shift moves bits to the right. For suitable non-negative values, shifting right by one position corresponds to integer division by 2.

```
10 = 1010
10 >> 1 = 0101 = 5
```

23. Complete Bitwise Example

```c
#include <stdio.h>

int main()
{
    int a = 10, b = 4;

    printf("AND = %d\n", a & b);
    printf("OR = %d\n", a | b);
    printf("XOR = %d\n", a ^ b);
    printf("NOT = %d\n", ~a);
    printf("Left Shift = %d\n", a << 1);
    printf("Right Shift = %d\n", a >> 1);

    return 0;
}
```

24. Ternary / Conditional Operator

The conditional operator is called ternary because it uses three operands.

```
condition ? expression1 : expression2;
```

If the condition is true, expression1 is used; otherwise expression2 is used.

25. Simple Ternary Example

```
int age = 20;

age >= 18 ? printf("Adult") : printf("Minor");
```

Since 20 >= 18 is true, the output is Adult.

26. Larger Number Using Ternary

```
#include <stdio.h>

int main()
{
    int a = 11, b = 25, c;

    c = (a > b) ? a : b;

    printf("The larger number is: %d", c);

    return 0;
}
```

27. Ternary vs if-else

```
if (a > b)
{
    c = a;
}
else
{
    c = b;
}
c = (a > b) ? a : b;
```

Use the ternary operator for short, simple decisions. Use if-else when the logic is more complex or needs multiple statements.

28. Miscellaneous Operators

Operator	Name / Purpose
sizeof()	Finds size of a type or variable
(type)	Type casting
,	Comma operator
&	Address-of operator
*	Dereference operator

29. sizeof() Operator

sizeof() gives the size in bytes of a data type or object.

```
int a;
printf("%zu", sizeof(a));
```

Common sizes on many systems are char = 1 byte, int = 4 bytes, float = 4 bytes and double = 8 bytes. The exact size of a type is implementation-dependent.

```
#include <stdio.h>

int main()
{
    char c;
    int i;
    float f;
    double d;

    printf("Size of c = %zu\n", sizeof(c));
    printf("Size of i = %zu\n", sizeof(i));
    printf("Size of f = %zu\n", sizeof(f));
    printf("Size of d = %zu\n", sizeof(d));

    return 0;
}
```

30. Type Casting

Type casting means explicitly converting a value from one data type to another.

```
float x = 5.26;
int y = (int)x;
```

Now y stores 5 because the fractional part is discarded.

```
int a = 5;
int b = 2;

printf("%d", a / b);           // 2
printf("%f", (float)a / b);    // 2.500000
```

31. Comma Operator

The comma operator evaluates expressions from left to right, and the value of the complete comma expression is the value of the last expression.

```
int a;
a = (10, 20, 30);

printf("%d", a);
```

Output: 30.

32. Address-of Operator (&)

& gives the memory address of a variable.

```
int age = 20;
printf("%p", (void*)&age);
```

The actual address changes between systems and executions.

33. Dereference Operator (*)

When used with a pointer, * accesses the value stored at the address held by that pointer.

```
int age = 20;
int *p = &age;

printf("%d", *p);
```

- &age → address of age
- p → stores that address
- *p → value stored at that address

34. Decision-Making Statements

Decision-making statements allow a program to choose which statements should execute.

- if
- if-else
- if-else-if ladder
- nested if
- switch

35. Conditions in C

A condition is an expression whose result determines which path the program takes.

```
age > 18
temperature <= 37.5
num == 10
```

36. Relational Operators

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to

!=	Not equal to

37. Very Important: = vs ==

Symbol	Meaning	Example
=	Assignment	age = 20;
==	Equality comparison	age == 20

Memory trick: = means assign; == means compare.

38. if Statement

An if statement executes its block only when the condition is true.

```
if (age >= 18)
{
    printf("Adult");
}
```

39. Absolute Value Example

```
#include <stdio.h>

int main()
{
    int number;

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

    if (number < 0)
    {
        number = -number;
    }

    printf("Absolute value = %d", number);

    return 0;
}
```

If the input is -14, number < 0 is true, so number becomes -(-14) = 14.

40. if-else Statement

if-else is used when there are two possible paths. Exactly one of the two blocks executes.

```
if (age >= 18)
{
    printf("Adult");
}
else
{
    printf("Minor");
}
```

41. Even or Odd Example

```
#include <stdio.h>

int main()
{
    int number;

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

    if (number % 2 == 0)
    {
        printf("%d is an Even Number", number);
    }
    else
    {
        printf("%d is an Odd Number", number);
    }

    return 0;
}
```

The % operator gives the remainder. If number % 2 is 0, the number is even.

42. if-else-if Ladder

Use an if-else-if ladder when there are multiple alternative conditions. C checks from top to bottom and executes the first true block.

```
if (condition1)
{
    // code
}
else if (condition2)
{
    // code
}
else if (condition3)
{
    // code
}
else
{
    // code when all are false
}
```

43. Positive, Negative or Zero

```
#include <stdio.h>

int main()
{
    int num;

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

    if (num > 0)
    {
        printf("Positive number");
    }
    else if (num < 0)
    {
        printf("Negative number");
    }
    else
```

```
    {
        printf("Zero");
    }

    return 0;
}
```

44. Nested if

A nested if is an if statement inside another if statement. The inner if is reached only when the outer condition allows execution to enter its block.

```
if (condition1)
{
    // statement

    if (condition2)
    {
        // statement
    }
}
```

45. Nested if Example

```
#include <stdio.h>

int main()
{
    int num;

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

    if (num >= 0)
    {
        if (num == 0)
        {
            printf("You entered zero.");
        }
        else
        {
            printf("You entered a positive number.");
```

```
        }
    }
    else
    {
        printf("You entered a negative number.");
    }

    return 0;
}
```

46. if-else-if vs Nested if

if-else-if ladder	Nested if
Used for several alternative conditions	Used when one decision depends on another
Conditions are checked in sequence	Inner condition is checked only after the outer condition allows it
Example: grading by marks	Example: check non-negative, then distinguish zero/positive

47. switch Statement

switch is useful when one expression must be compared against multiple fixed integral choices.

```
switch (expression)
{
    case value1:
        // code
        break;

    case value2:
        // code
        break;

    default:
        // code
}
```

48. How switch Works

```
int choice = 2;

switch(choice)
{
    case 1:
        printf("One");
        break;

    case 2:
        printf("Two");
        break;

    case 3:
        printf("Three");
        break;

    default:
        printf("Invalid");
}
```

Since choice is 2, case 2 executes and prints Two.

49. Why is break Used?

break terminates the switch statement. Without it, execution can continue into the next case, which is called fall-through.

```
int n = 1;

switch(n)
{
    case 1:
        printf("One");

    case 2:
        printf("Two");
}
```

Output: OneTwo, because there is no break after case 1.

50. default in switch

default executes when none of the case labels match.

```
int choice = 5;

switch(choice)
{
    case 1:
        printf("Apple");
        break;

    case 2:
        printf("Banana");
        break;

    default:
        printf("Invalid choice");
}
```

51. What Can Be Used in switch?

- For introductory C, switch is used with integral types such as int and char.
- float and double are not valid switch expression types.
- case labels must be constant integral values.
- Logical expressions such as case x > 5: are not how switch cases work.

52. Vowel Example Using switch

```
#include <stdio.h>

int main()
{
    char ch;

    printf("Enter a character: ");
    scanf(" %c", &ch);

    switch(ch)
    {
        case 'a':
```

```
            printf("Vowel");
            break;

        case 'e':
            printf("Vowel");
            break;

        case 'i':
            printf("Vowel");
            break;

        case 'o':
            printf("Vowel");
            break;

        case 'u':
            printf("Vowel");
            break;

        default:
            printf("Not a vowel");
    }

    return 0;
}
```

The space before %c in scanf(" %c", &ch) helps skip whitespace left in the input stream.

53. Better Vowel Example with Fall-through

```
switch(ch)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        printf("Vowel");
        break;

    default:
```

```
        printf("Not a vowel");
}
```

Several cases can intentionally fall through to the same statement. Once a matching case is found, execution continues until the break.

54. switch vs if-else-if

if-else-if	**switch**
Can test ranges such as marks >= 90	Best for fixed choices such as 1, 2, 3
Can use complex conditions	Matches the switch expression against case values
Flexible	Usually clearer for menu-like choices

55. Important Symbols You Must Know

Symbol	**Meaning**
=	Assignment
==	Equal comparison
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
%	Remainder
&	Bitwise AND or address-of, depending on context
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Left shift
>>	Right shift
?:	Ternary/conditional operator
*	Multiplication or dereference, depending on context

,	Comma operator or separator, depending on context

56. Same Symbol, Different Meaning

C reuses some symbols in different contexts.

Symbol	Context	Meaning
&	a & b	Bitwise AND
&	&age	Address of age
*	a * b	Multiplication
*	*p	Dereference pointer p