
BCA
Semester - 1st

C – Programming

Notes - 1

Contents

Structures of 'C' Programming Language, Elements of C Programming Algorithms and flowcharts (Real Life Examples), Exercises, C Tokens , Keywords, Identifiers, Variables, Constant, Data Types, Operators, Types of operators

1. Structure of C Programming Language

A C program follows a specific structure — a defined way of writing code so that the compiler can understand and execute it properly.

The structure of a C program is divided into several sections:

Basic Structure of a C Program

```c
#include <stdio.h>     // Header file section

// Function declaration
int add(int, int);

int main() {           // Main function
    int a, b, sum;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    sum = add(a, b);   // Function call
    printf("Sum = %d", sum);
    return 0;          // Return statement
}

// Function definition
int add(int x, int y) {
    return x + y;
}
```

Sections in a C Program

1. Documentation Section
   - Contains comments that describe the purpose of the program.
   - Example:

     ```c
     // Program to add two numbers
     ```

2. Link Section
   - Includes header files using `#include`.
   - Example: `#include <stdio.h>`
3. Definition Section
   - Defines constants using `#define`.
   - Example:

     ```c
     #define PI 3.14
     ```

4. **Global Declaration Section**
   - Declares global variables and functions that are used throughout the program.
   - Example:

```c
int total;
```

5. **Main Function Section**
   - The entry point of every C program. Execution starts from `main()`.
   - Example:

```c
int main() {
    // body of program
}
```

6. **Subprogram Section (User-defined Functions)**
   - Contains function definitions that perform specific tasks.
   - Example:

```c
void display() {
    printf("Hello!");
}
```

---

2. Elements of C Programming

The basic building blocks or elements of C language are:

1. Character Set

- C programs are made up of characters like:
  - **Letters:** A–Z, a–z
  - **Digits:** 0–9
  - **Special symbols:** +, -, *, /, %, =, #, etc.
  - **Whitespace:** space, tab, newline

2. Keywords

- Predefined words with special meanings in C.
- Example: `int`, `float`, `if`, `else`, `for`, `while`, `return`
- C has 32 keywords.

3. Identifiers

- Names given to variables, functions, and arrays.
- Example: `age`, `totalMarks`, `calculateSum`

4. Constants

- Fixed values that do not change during program execution.
- Example:

```c
const float PI = 3.14;
```

5. Variables

- Named memory locations used to store data.
- Example:

```c
int age = 20;
```

6. Data Types

- Define the type of data a variable can hold.
  - `int` → integer
  - `float` → decimal
  - `char` → single character
  - `double` → large floating point

7. Operators

- Symbols used to perform operations on data.
- Example:
  - Arithmetic: `+`, `-`, `*`, `/`, `%`
  - Relational: `>`, `<`, `==`, `!=`
  - Logical: `&&`, `||`, `!`

8. Expressions

- Combination of variables, constants, and operators that produce a result.
- Example:

```c
total = marks1 + marks2;
```

9. Statements

- Instructions given to the computer.
- Example:

```c
printf("Hello World");
```

---

3. Algorithms and Flowcharts

What is an Algorithm?

An **algorithm** is a **step-by-step procedure** to solve a problem.

Example: Algorithm to add two numbers

1. Start
2. Read two numbers: A and B
3. Calculate sum = A + B
4. Display sum
5. Stop

Real-Life Example (Making Tea):

1. Start
2. Boil water
3. Add tea leaves
4. Add sugar and milk
5. Boil again
6. Strain tea into a cup
7. Stop

This is also an **algorithm** — it solves a problem (making tea) step-by-step.

What is a Flowchart?

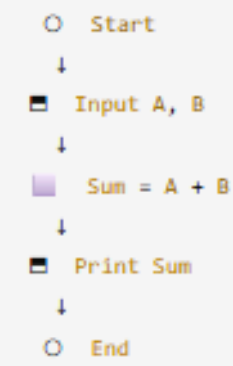
A **flowchart** is a **diagram** that shows the steps of an algorithm using **symbols**.

Symbol	Meaning
○	Start/End
▭	Process (e.g., calculation)
♦	Decision (Yes/No)
⬒	Input/Output

Example: Flowchart to Add Two Numbers

mathematica

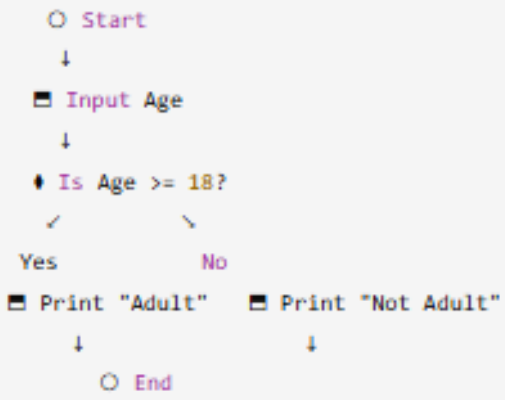
Copy code



Real-Life Example: Flowchart for "Check if Person is Adult"

pgsql

Copy code



---

4. Exercises (For Practice)

Q1. Write an algorithm and flowchart to find the largest of two numbers.

Algorithm:

1. Start
2. Input A, B
3. If A > B → Print "A is largest"
   Else → Print "B is largest"
4. Stop

Flowchart:

```
css                                                    Copy code

    ○ Start
     ↓
   ▭ Input A, B
     ↓
   ◆ A > B ?
    ↙        ↘
Yes           No
▭ Print A    ▭ Print B
     ↓          ↓
      ○ End
```

Q2. Write an algorithm to calculate the area of a rectangle.

1. Start
2. Input length (L) and breadth (B)
3. Area = L × B
4. Display Area
5. Stop

Q3. Algorithm for Checking Even or Odd

1. Start
2. Input number N
3. If N % 2 == 0 → Print "Even"
   Else → Print "Odd"
4. Stop

1. C Tokens

Meaning:

- Tokens are the smallest units or building blocks of a C program.
- A C compiler reads the program and breaks it into tokens before processing it.

Example:

```c
int sum = a + b;
```

Here the tokens are:

```markdown
1. int
2. sum
3. =
4. a
5. +
6. b
7. ;
```

Types of Tokens in C:

Type	Example	Description
1. Keywords	`int`, `if`, `return`	Predefined reserved words
2. Identifiers	`sum`, `total`, `main`	Names given by programmer
3. Constants	`10`, `'A'`, `3.14`	Fixed values
4. Strings	`"Hello"`	Sequence of characters
5. Operators	`+`, `-`, `*`, `/`, `=`	Symbols that perform operations
6. Special Symbols	`{}`, `()`, `;`, `#`	Used in syntax and structure

✅ In short:

> Tokens are the basic components of a program — like words in a sentence.

2. Keywords

Meaning:

- Keywords are predefined or reserved words in C language.
- They have special meanings and cannot be used as variable names.

Example:

```c
int main() {
    int age = 20;
    return 0;
}
```

Here `int`, `main`, and `return` are keywords.

List of Common C Keywords (Total 32):

```cpp
auto        double      int         struct
break       else        long        switch
case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

✅ **Note:**

Keywords are always written in lowercase and have special meaning to the compiler.

3. Identifiers

Meaning:

- Identifiers are names given by the programmer to identify variables, functions, arrays, etc.
- They help in naming and referring to data or functions in a program.

Example:

```c
int marks;
float average;
void calculateTotal();
```

Here `marks`, `average`, and `calculateTotal` are identifiers.

Rules for Naming Identifiers:

1. Can contain letters (A–Z, a–z), digits (0–9), and underscore (_).
2. Must begin with a letter or underscore.
3. No spaces or special symbols allowed.
4. Case-sensitive (`Age` and `age` are different).
5. Cannot be a keyword (e.g., `int`, `return`).

✅ Examples of Valid Identifiers:

```python
totalMarks, _sum, age1
```

❌ Invalid Identifiers:

```css
1total, total-marks, float
```

4. Variables

Meaning:

- A **variable** is a **named memory location** used to store a value that can change during program execution.
- Think of a variable like a **container** that holds data.

Example:

```c
int age = 20;
float salary = 35000.50;
```

Here:

- `age` is a variable of type `int`
- `salary` is a variable of type `float`

Rules for Declaring Variables:

1. Must begin with a letter or underscore.
2. Should not be a keyword.
3. Must be declared before use.
4. Can store values according to their **data type**.

Syntax:

```c
data_type variable_name = value;
```

Example:

```c
int num = 10;
char grade = 'A';
float percentage = 92.5;
```

Types of Variables:

Type	Description	Example
Local Variable	Declared inside a function	`int x;` inside main()
Global Variable	Declared outside all functions	`int total;` before main()
Static Variable	Keeps its value between function calls	`static int count = 0;`
Automatic Variable	Default variable inside function	`auto int num;`
External Variable	Declared in another file using `extern`	`extern int a;`

✅ **Example Program:**

c

Copy code

```c
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    int sum = a + b;
    printf("Sum = %d", sum);
    return 0;
}
```

---

5. Constants

Meaning:

- Constants are fixed values that do not change during the execution of the program.

Example:

c

Copy code

```c
const float PI = 3.14;
```

Types of Constants:

Type	Example	Description
1. Integer Constant	`10`, `-25`, `0`	Whole numbers
2. Floating Constant	`3.14`, `-0.25`	Decimal numbers
3. Character Constant	`'A'`, `'b'`, `'9'`	Single character enclosed in single quotes
4. String Constant	`"Hello"`, `"C Program"`	Set of characters enclosed in double quotes
5. Symbolic Constant	`#define PI 3.14`	Defined using `#define` keyword
6. Constant Variables	`const int a = 10;`	Declared using `const` keyword

Examples:

c

Copy code

```c
#include <stdio.h>
#define PI 3.14   // symbolic constant

int main() {
    const int DAYS = 7;  // constant variable
    float radius = 2.0;
    float area = PI * radius * radius;
    printf("Area = %.2f", area);
    return 0;
}
```

🧩 Data Types in C Programming

Meaning:

A **data type** in C tells the **compiler** what kind of data a variable can store and **how much memory** it will occupy.

👉 Example:

```c
int age = 21;
float price = 250.75;
char grade = 'A';
```

Here,

- `int` → integer type (stores whole numbers)
- `float` → floating-point type (stores decimal numbers)
- `char` → character type (stores single characters)

🧠 Why Data Types Are Important

1. They define **the type of data** a variable can hold.
2. They help the compiler **allocate memory** properly.
3. They ensure **data accuracy** and **type safety** in operations.

⚙️ Types of Data Types in C

C provides **four main categories** of data types:

Category	Description	Example
1. Basic Data Types	Fundamental types built into C	`int`, `float`, `char`, `double`
2. Derived Data Types	Formed using basic types	`arrays`, `pointers`, `structures`, `unions`
3. Enumeration Data Type	User-defined type for named constants	`enum`
4. Void Data Type	Represents "no value"	`void`

1️⃣ Basic Data Types

Data Type	Size (in Bytes)*	Range (Approx.)	Example
int	2 or 4	-32,768 to 32,767 (or higher)	`int age = 20;`
float	4	3.4×10^{-38} to 3.4×10^{38}	`float marks = 85.5;`
double	8	1.7×10^{-308} to 1.7×10^{308}	`double salary = 35000.75;`
char	1	-128 to 127	`char grade = 'A';`

* Size may vary depending on the compiler and system (16-bit or 32-bit).

📘 Example Program:

c

Copy code

```c
#include <stdio.h>

int main() {
    int age = 20;
    float height = 5.9;
    char grade = 'A';
    double salary = 25000.75;

    printf("Age: %d\n", age);
    printf("Height: %.1f\n", height);
    printf("Grade: %c\n", grade);
    printf("Salary: %.2lf\n", salary);

    return 0;
}
```

Output:

makefile

Copy code

```
Age: 20
Height: 5.9
Grade: A
Salary: 25000.75
```

2️⃣ Derived Data Types

Derived data types are created from basic types to handle more complex data.

Type	Meaning	Example
Array	Collection of same type elements	`int marks[5];`
Pointer	Stores memory address of a variable	`int *ptr;`
Structure	Groups different data types	`struct Student { int roll; char name[20]; };`
Union	Similar to structure but shares memory	`union Data { int i; float f; };`

🧩 Example:

c

Copy code

```c
struct Student {
    int roll;
    char name[20];
    float marks;
};
```

Here, `Student` is a structure containing different data types together.

3️⃣ Enumeration Data Type (enum)

Meaning:

- It is a **user-defined data type** used to assign names to integer constants — makes program easy to read.

Example:

c

Copy code

```c
#include <stdio.h>

enum Days { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main() {
    enum Days today = Monday;
    printf("Day number: %d", today);
    return 0;
}
```

Output:

sql

Copy code

```sql
Day number: 1
```

> By default, enumeration constants start from 0 (Sunday = 0, Monday = 1, etc.).

🔽 Void Data Type

Meaning:

- Represents no value or empty type.
- Commonly used in functions that do not return a value.

Example:

```c
#include <stdio.h>

void greet() {
    printf("Hello, welcome to C programming!");
}

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

Here,
`void` indicates that the function `greet()` does not return any value.

🏷️ Type Modifiers in C

C provides type modifiers to change the size and range of basic data types.

Modifier	Used With	Meaning
short	int	Smaller range integer
long	int, double	Larger range integer or double
signed	int, char	Can store both + and – values
unsigned	int, char	Only + values (doubles the positive range)

Example:

```c
short int a;
long int b;
unsigned int c;
signed char d;
```

Example Table:

Type	Size	Range
short int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
long int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	0 to 4,294,967,295

💡 Example Program (Type Modifiers)

c Copy code

```c
#include <stdio.h>
int main() {
    short int a = 10;
    long int b = 123456;
    unsigned int c = 50;

    printf("Short: %d\n", a);
    printf("Long: %ld\n", b);
    printf("Unsigned: %u", c);
    return 0;
}
```

Output:

makefile Copy code

```
Short: 10
Long: 123456
Unsigned: 50
```

✅ Summary (For Revision)

Type	Description	Example
Basic	Built-in types like int, char, float, double	`int a = 5;`
Derived	Formed using basic types	`int arr[10];`
Enumeration	User-defined named constants	`enum Color {red, blue};`
Void	No value or return type	`void function()`

⚙️ 1. Operators in C Programming

Meaning:

An operator is a symbol that tells the compiler to perform a specific operation on data or variables.

👉 In simple words:
Operators are symbols used to manipulate data or perform calculations.

Example:

```c
int a = 10, b = 5;
int sum = a + b;
```

Here,

- `+` is an operator
- `a` and `b` are operands
- `sum = a + b` is an expression

✅ Expression = Operands + Operator

🧩 2. Types of Operators in C

C language provides several types of operators, grouped as follows:

Type	Example	Description
1. Arithmetic Operators	`+`, `-`, `*`, `/`, `%`	Perform mathematical operations
2. Relational Operators	`>`, `<`, `>=`, `<=`, `==`, `!=`	Compare two values
3. Logical Operators	`&&`, `	
4. Assignment Operators	`=`, `+=`, `-=`, `*=`, `/=`, `%=`	Assign values to variables
5. Increment and Decrement Operators	`++`, `--`	Increase or decrease value by 1
6. Conditional (Ternary) Operator	`?:`	Acts as shorthand for if-else
7. Bitwise Operators	`&`, `	`, `^`, `~`, `<<`, `>>`
8. Special Operators	`sizeof`, `,`, `&`, `*`, `->`, `.`	Special purposes in C

Let's understand each one clearly 👇

🧮 1. Arithmetic Operators

Used to perform basic mathematical operations.

Operator	Description	Example	Result
`+`	Addition	`a + b`	Sum of a and b
`-`	Subtraction	`a - b`	Difference
`*`	Multiplication	`a * b`	Product
`/`	Division	`a / b`	Quotient
`%`	Modulus	`a % b`	Remainder

Example:

```c
int a = 10, b = 3;
printf("%d", a % b);  // Output: 1
```

⚖️ 2. Relational Operators

Used to compare two values.

They return true (1) or false (0).

Operator	Meaning	Example	Result
`>`	Greater than	`a > b`	true if a is greater
`<`	Less than	`a < b`	true if a is smaller
`>=`	Greater or equal	`a >= b`	true if a ≥ b
`<=`	Less or equal	`a <= b`	true if a ≤ b
`==`	Equal to	`a == b`	true if equal
`!=`	Not equal to	`a != b`	true if not equal

Example:

```c
int x = 5, y = 10;
if (x < y)
    printf("x is smaller");
```

🧠 3. Logical Operators

Used to combine or invert conditions in decision-making statements.

Operator	Meaning	Example	Result
`&&`	Logical AND	`(a > b) && (a > c)`	True if both conditions true
`		`	`	Logical OR
`!`	Logical NOT	`!(a > b)`	Reverses condition

Example:

```c
int age = 20;
if (age >= 18 && age <= 60)
    printf("Eligible to work");
```

📝 4. Assignment Operators

Used to assign values to variables.

Operator	Example	Same As
`=`	`a = 10`	Assign 10 to a
`+=`	`a += 5`	`a = a + 5`
`-=`	`a -= 3`	`a = a - 3`
`*=`	`a *= 2`	`a = a * 2`
`/=`	`a /= 4`	`a = a / 4`
`%=`	`a %= 2`	`a = a % 2`

Example:

```c
int x = 10;
x += 5;   // x = x + 5 → 15
```

🔼 5. Increment and Decrement Operators

Used to increase or decrease variable's value by 1.

Operator	Meaning	Example	Result
`++`	Increment	`a++` or `++a`	Increases a by 1
`--`	Decrement	`a--` or `--a`	Decreases a by 1

Example:

```c
int a = 5;
printf("%d", ++a); // Output: 6
printf("%d", a--); // Output: 6 (then a becomes 5)
```

- Prefix (`++a`) → Increments before using value
- Postfix (`a++`) → Increments after using value

🎯 6. Conditional (Ternary) Operator

Used as a shortcut for if-else statement.

Syntax:

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

Example:

```c
int a = 10, b = 20;
int max = (a > b) ? a : b;
printf("Largest = %d", max);
```

Output:

```ini
Largest = 20
```

🔢 7. Bitwise Operators

Operate directly on bits (binary form) of numbers.

Operator	Meaning	Example	Result (if a=5, b=3)
`&`	Bitwise AND	`a & b`	1
`	`	`	Bitwise OR	`a
`^`	Bitwise XOR	`a ^ b`	6
`~`	Bitwise NOT	`~a`	-6
`<<`	Left Shift	`a << 1`	10
`>>`	Right Shift	`a >> 1`	2

Example:

```c
int a = 5, b = 3;
printf("%d", a & b); // Output: 1
```

🔗 8. Special Operators

These have special uses in C programming.

Operator	Name	Purpose
`sizeof`	Size of operator	Finds size of data type
`,`	Comma operator	Separates multiple expressions
`&`	Address of operator	Gives memory address
`*`	Pointer operator	Accesses value at an address
`->`	Arrow operator	Access structure member via pointer
`.`	Dot operator	Access structure member

Examples:

```c
printf("%lu", sizeof(int));   // sizeof
int a = 5, b = 10;
int c = (a++, b++);           // comma
printf("%d", c);              // Output: 10
```

✅ Summary Table

Type	Operators	Example	Purpose
Arithmetic	+ - * / %	`a + b`	Math operations
Relational	> < >= <= == !=	`a > b`	Compare values
Logical	&& || !	`(a>0 && b>0)`	Combine conditions
Assignment	= += -= *= /= %=	`a += 5`	Assign value
Increment/Decrement	++ --	`a++`	Increase/decrease by 1
Conditional	?:	`(a>b)?a:b`	Short if-else
Bitwise	& | ^ ~ << >>	`a & b`	Work on bits
Special	sizeof , & * . ->	`sizeof(a)`	Special use

🧠 Example Program (All Operators Combined)

c

Copy code

```c
#include <stdio.h>

int main() {
    int a = 10, b = 5, c;

    // Arithmetic
    c = a + b;
    printf("Sum = %d\n", c);

    // Relational
    printf("a > b = %d\n", a > b);

    // Logical
    printf("(a > b && b > 0) = %d\n", (a > b && b > 0));

    // Assignment
    a += 5;
    printf("a after += 5: %d\n", a);

    // Increment
    b++;
    printf("b after increment: %d\n", b);

    // Conditional
    int max = (a > b) ? a : b;
    printf("Max = %d", max);

    return 0;
}
```

✍️ In Short (For Exam Writing):

> Operators in C are special symbols used to perform operations on variables and values.
> There are 8 main types of operators — Arithmetic, Relational, Logical, Assignment, Increment/Decrement, Conditional, Bitwise, and Special Operators.

Practice Questions

1. What are the main structures of C programming language? Explain its basic components.
2. What are the elements of C programming? Describe each element briefly.
3. What is an algorithm? Explain with a real-life example.
4. What is a flowchart? Draw and explain a flowchart for finding the largest of two numbers.
5. What are C tokens? List and explain different types of tokens in C.
6. What are keywords in C? Give examples and explain their purpose.
7. What are identifiers, variables, and constants? Explain with examples.
8. What are data types in C? Explain different types of data types with examples.
9. What are operators in C language?
10. Explain the different types of operators in C with examples (Arithmetic, Relational, Logical, etc.).

Check the answer in the Practice Questions section on our website.

Visit Website