* **C basics: C character set, Identifiers and keywords, Data types, constants, variables and arrays, declarations, expressions statements, symbolic constants, compound statements, arithmetic operators, unary operators, relational and logical operators, assignment operators, conditional operators, bit operators.**

**1. C Character Set**

The C character set includes:

* **Letters:** A to Z (uppercase), a to z (lowercase)
* **Digits**: 0 to 9
* **Special Characters:** Symbols like!, @, #, $, %, ^, &, \*, (, ), {, }, [, ], ;, :, ,, ., <, >, /, ?, \, |, ~, `, etc.
* **White Spaces:** Space, tab, newline, etc.

**2. Identifiers and Keywords**

* **Identifiers:** Names used to identify variables, functions, arrays, etc. in a program. Identifiers must start with a letter (uppercase or lowercase) or an underscore (\_), followed by letters, digits, or underscores.
  + **Examples:** sum, totalAmount, \_index
* **Case-sensitive**: Variable and variable are different identifiers.
* **No spaces or special characters**: Only letters, digits, and underscores are allowed.
* **Cannot use keywords** as identifiers (e.g., int, return).
* **Keywords:** Reserved words in C that have a special meaning and cannot be used as **identifiers.** Keywords are always written in lowercase.

Examples include int, return, if, else, while, for, char, etc.

**3. Data Types**

C provides several data types to define the nature of the data that can be stored in a variable:

* **Basic Data Types:**
  + **int**: Integer numbers (e.g., 5, -23)
* **size** 4 bytes (on most systems)
* **Format specifier** %d or %i
* **Range** -2147,483,648 to 2147,483,647
  + **char**: Character type (e.g., 'a', '1')
* **size** 1 byte
* **Format specifier** %c
* **Range** -128 to 127 (for signed char) or 0

to 255 (for unsigned char)

* + **float**: Single-precision floating-point numbers (e.g., 3.14)
* **size** 4 byte
* **Format specifier** %f or %g
* **Range** Approximately 3.4E-38 to

3.4E+38 (precision of about 6-7

decimal digits)

* + **double**: Double-precision floating-

point numbers (e.g., 3.14159)

* **size** 8 byte
* **Format specifier** %lf
* **Range** Approximately 1.7E-308 to

1.7E+308(precision of about 15-16

decimal digits)

* **Derived Data Types:**
  + Arrays, pointers, structures, unions
* **Void Type:** Represents the absence of type.

**4. Constants**

Constants are fixed values that do not change during the execution of a program. They can be of several types:

* **Integer Constants:** E.g., 123, -456
* **Floating-point Constants:** E.g., 3.14,

-0.001

* **Character Constants:** E.g., 'a', '9'

**Size** Typically, a character constant 1 byte of Memory (depends on the system and The Programming language)

* **String Constants: E.g.,** "Hello, World!"

**Size** A string Constant occupies Multiple Bytes in Memory, One for each Character plus an actional byte for the null Terminator \0 (in languages like c/c++)

**5. Variables and Arrays**

* **Variables**: Named storage locations in memory to store data. They must be declared before use, specifying the type of data they will hold.

- **Declaration:** `int x;` , `float y;`

- **Initialization:** `int x = 5;` ,`float y = 3.14;`

* **Arrays**: A collection of variables of the same type, stored in contiguous memory locations.

**1. Variable Naming Rules:**

**- Start with a letter or underscore (`\_`):**

**- Valid:** `int age;`, `int \_count;`

**- Invalid:** `int 2nd Place;`

**- Case-sensitive:**

**- No special characters:**

- **Valid:** `float price;` ,

**- Invalid:** `float price$;`

**- No spaces allowed:**.

**- Valid:** `int firstName;`

**- Invalid:** `int first name;`

**- Reserved keywords:** You cannot use C's reserved keywords (like `int`, `return`, `if`, `else`, etc.) as variable names.

**- Invalid:** `int return;`

**Variable Declaration:**

- Example: `int age;` declares an integer

variable named `age`.

**Variable Initialization:**

If not initialized, they might contain garbage values (especially for local variables).

- Example: `int age = 25;` initializes `age`

with the value `25`.

**6. Declarations**

In C, a variable must be declared before it is used, specifying its data type and optionally initializing it:

c

int count; // Declaration

int total = 100; // Declaration and

Initialization

**7. Expressions and Statements**

* **Expressions**: Combinations of variables, constants, and operators that are evaluated to produce a value.
  + Example: a + b, 3 \* x + y
* **Statements**: Instructions executed by the compiler. Statements usually end with a semicolon (;).
  + Example: x = 5;
* **8. Symbolic Constants**

Symbolic constants are constants given a name, using the #define preprocessor directive, making code more readable and maintainable.

c

* #define PI 3.14

### 9. ****Compound Statements****

Compound statements or blocks are a group of statements enclosed within braces {}. They are used in functions, loops, conditionals, etc.

c

{

int a = 10;

int b = 20;

printf("%d", a + b);

}

**10. Operators**

**1. Arithmetic Operators**

* Used for basic mathematical operations.
  + **+ :** Addition (e.g., a + b)
  + **- :** Subtraction (e.g., a - b)
  + **\* :** Multiplication (e.g., a \* b)
  + **/ :** Division (e.g., a / b)
  + **% :** Modulus (remainder of division,

e.g., a % b)

**2. Unary Operators**

* Operate on a single operand.
  + **- :** Negation (e.g., -a)
  + **++ :** Increment (increases value by 1,

e.g., a++ or ++a)

* + **-- :** Decrement (decreases value by 1,

e.g., a-- or --a)

**3. Relational Operators**

* Used for comparisons, return true or false.
  + **==** : Equal to (e.g., a == b)
  + **!=** : Not equal to (e.g., a != b)
  + **>** : Greater than (e.g., a > b)
  + **<** : Less than (e.g., a < b)
  + **>=** : Greater than or equal to (e.g., a >=

b)

* + **<= :** Less than or equal to (e.g., a <= b)

**4. Logical Operators**

* Used to combine conditional expressions.
  + **&& :** Logical AND (true if both are

true, e.g., a && b)

* + **|| :** Logical OR (true if at least one is

true, e.g., a || b)

* + **! :** Logical NOT (inverts truth value,

e.g., !a)

**5. Assignment Operators**

* Used to assign values to variables.
  + **= :** Simple assignment (e.g., a = 5)
  + **+= :** Add and assign (e.g., a += 5 is

equivalent to a = a + 5)

* + **-= :** Subtract and assign (e.g., a -= 5)
  + **\*= :** Multiply and assign (e.g., a \*= 5)
  + **/= :** Divide and assign (e.g., a /= 5)
  + **%= :** Modulus and assign (e.g., a %= 5)

**6. Conditional Operator**

* Ternary operator for short if-else conditions.
  + **? :** : Conditional (e.g., condition ?

value\_if\_true : value\_if\_false)

**7. Bitwise Operators**

* Operate on bits, used for bit-level manipulation.
  + **& :** AND (e.g., a & b)
  + **| :** OR (e.g., a | b)
  + **^ :** XOR (exclusive OR, e.g., a ^ b)
  + **~ :** NOT (bitwise negation, e.g., ~a)
  + **<< :** Left shift (e.g., a << 2 shifts bits of

a left by 2)

* + **>> :** Right shift (e.g., a >> 2 shifts bits

of a right by 2)

**A flowchart** is a diagram that represents the sequence of steps in a process or a program. It uses different shapes to show different types of actions or steps, connected by arrows that indicate the flow of control.

* **Oval**: Represents the start or end of the program. It is used to signify where the flow begins and where it terminates.
* **Rectangle**: Represents a process or an operation, such as a calculation or an assignment in a program.
* **Parallelogram**: Represents input or output operations, like reading data from a user or displaying results on the screen.
* **Diamond**: Represents a decision point that branches the flow based on a condition, such as an if statement in programming.
* **Arrows**: Show the direction of flow, indicating the order in which the steps are executed.

By visualizing these steps, flowcharts help in understanding and planning the logic of a program before writing actual code.