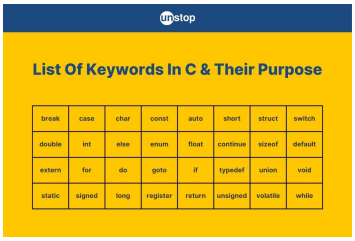
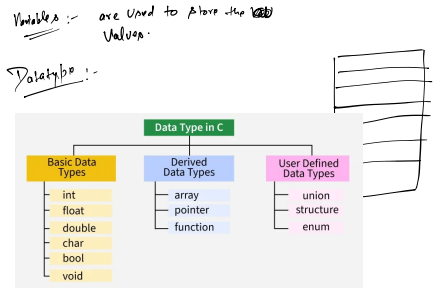
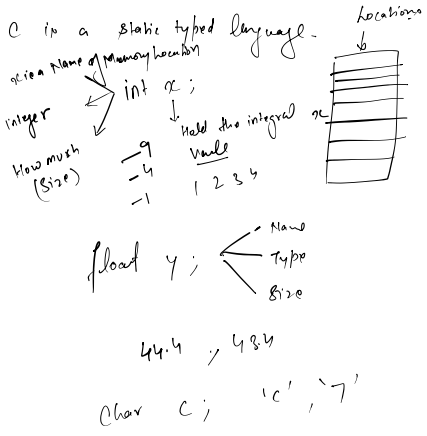
Section C 02

Thursday, 11 September 2025 10:54 AM

https://github.com/TeachToTech-in/Rahul_Trainer







Rules for Naming Identifiers in C:

There are specific rules you must follow when creating identifiers in C:

1. **Characters Allowed:**
   - They can consist of uppercase letters (A-Z).
   - They can consist of lowercase letters (a-z).
   - They can consist of digits (0-9).
   - They can include the underscore character (_).
2. **Starting Character:**
   - An identifier **must** begin with an alphabet (A-Z or a-z) or an underscore (_).
   - It **cannot** start with a digit.
3. **Case-Sensitivity:**
   - C is a **case-sensitive** language. This means myVariable, myvariable, and MyVariable are all considered different identifiers.
4. **Reserved Keywords (Keywords):**
   - You **cannot** use C's reserved keywords (also known as keywords) as identifiers. These keywords have special meanings to the compiler (e.g., int, if, else, while, for, return, void, etc.).
5. **No Special Characters (Except Underscore):**
   - You cannot use any other special characters (like !, @, #, $, %, ^, &, *, (,), -, +, =, {, }, [,], |, \, :, ;, ', ", <, >, /, ?, ., ,) in identifiers.
6. **No Spaces:**
   - Identifiers cannot contain spaces. If you need a name with multiple words, use underscores (e.g., total_sum) or camelCase (e.g., totalSum).
7. **Length (Compiler Dependent):**
   - While there's no strict limit defined by the C standard for the length of an identifier, compilers typically support identifiers up to a certain length (often 31 characters for external identifiers and more for internal ones). It's good practice to keep them reasonably concise but descriptive.

Examples of Valid and Invalid Identifiers:

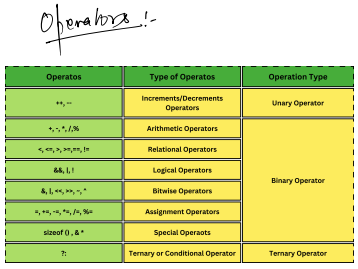
Valid Identifiers	Invalid Identifiers	Reason for Invalidity
myVariable	1variable	Starts with a digit
count	my variable	Contains a space
_temp	int	int is a reserved keyword
calculateSum	total#value	Contains a special character (#)
MAX_SIZE	for	for is a reserved keyword
data_entry_point	first-name	Contains a special character (-)

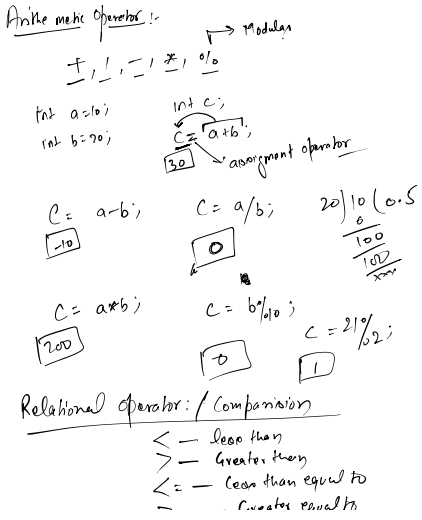
Export to Sheets

Good Practices for Naming Identifiers:

While the rules define what's *allowed*, good programming practices suggest how to make your identifiers *readable* and *maintainable*:

- **Descriptive Names:** Choose names that clearly indicate the purpose or content of the identifier. For example, studentAge is better than sa.
- **Consistency:** Stick to a consistent naming convention throughout your codebase. Common conventions include:
  - **camelCase:** myVariableName, calculateTotalSum (first letter of first word lowercase, subsequent words start with uppercase). Often used for variables and functions.
  - **PascalCase (or UpperCamelCase):** MyStructName, ClassName (first letter of every word uppercase). Often used for structures, unions, and sometimes function names.
  - **snake_case:** my_variable_name, calculate_total_sum (words separated by underscores). Often used for variables, functions, and macros.
  - **MACRO_CASE (or SCREAMING_SNAKE_CASE):** MAX_VALUE, PI (all uppercase with underscores). Commonly used for symbolic constants defined with #define.
- **Avoid Single-Letter Names (unless context is clear):** While i for a loop counter is common, avoid x or y for more significant variables.
- **Be Mindful of Scope:** In larger programs, consider adding prefixes or suffixes to avoid name clashes





(though namespaces in C++ address this more directly).

In C programming, a **data type** is a classification that tells the compiler how the programmer intends to use a variable. Specifically, it determines:

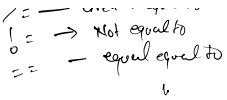
1. **The amount of memory (storage size)** to be allocated for a variable.
2. **The type of values** that can be stored in that memory location (e.g., whole numbers, decimal numbers, characters).
3. **The range of values** that the variable can hold.
4. **The operations** that can be performed on the data stored in the variable.

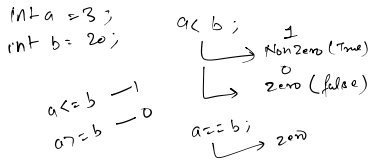
Think of data types as blueprints for memory. When you declare a variable with a specific data type, you're essentially telling the compiler, "Hey, set aside this much space, and I'm going to put this *kind* of data in it."

C data types can be broadly categorized into:

1. **Basic (or Primitive) Data Types:** These are the fundamental data types built into the C language.
2. **Derived Data Types:** These are constructed from the basic data types.
3. **User-Defined Data Types:** These are custom data types created by the programmer.
4. **Void Type:** A special type indicating the absence of a type.

Let's explore each category:





Logical Operators:- are used to combine the two or more conditions

AND OR NOT

&& || !

&& → if any input is zero o/p is zero (false) otherwise one (True)

Truth Stmt

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

OR - if any of the Input is true o/p is true otherwise false

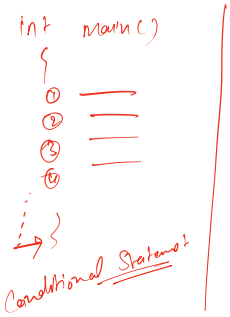
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

! → It alters the Value

A	Y
0	1
1	0

Plan

1. Registers → Leetcode → Gfg.
2. Google forms
3. Revise
4. if | if/else Stmt
5. loops while | do while | for
6. Memory Layout in C.




```
int main()
{
    int a, b;
    printf("Enter the two Value to
            Compare \n");
    &canf("%d %d", &a, &b);
    if (a > b){
        printf("%d is Greater than %d", a, b);
    }
    printf("Exit");
    return 0;
}
```