Identifiers in C Language: Full Guide

What Are Identifiers in C Language?

In C programming language, an identifier is a name given to a variable, function, or any other user-defined item.

It is used to identify and refer to the item in the program.

Identifiers in C language can be made up of a sequence of letters, digits, and underscores, and the first character must be a letter or underscore.

There are certain rules for identifiers in C, including:

- An identifier must be unique within the scope they are defined in.
- Identifiers must start with a letter or an underscore. They cannot start with a digit.
- Identifiers are case-sensitive, so "myVariable" and "MyVariable" are two different identifiers.
- Identifiers can include letters, digits, and underscores, but no other special characters.
- Identifiers must not be C keywords. You must know that keywords and identifiers in C are different.

Character Set Allowed As Identifiers in C

There are 63 total alphanumeric characters on how an identifier can be represented, consisting of:

- 26 lowercase letters (a, b, c, d…….. x, y, z)
- 26 uppercase letters (A, B, C, D……. X, Y, Z)
- 10 numeric digits from 0 to 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Underscore (_)

Examples

Here are some easy examples of identifiers in C:

- Example: 1

```
int count;
float total_price;
char first_name;
```

void print_message();

In the above example, "*count*", "*total_price*", "*first_name*", and "*print_message*" are all identifiers that have been declared to represent a variable or function within the program.

Whereas, *int*, *float*, *char*, and *void* are keywords in C program.

- Example: 2

void displayMessage();
float calculateAverage(int array[], int size);
int getLargest(int a, int b);

In this example, the identifiers are:

- ***displayMessage***: Representing a function which takes no arguments and returns no value. It has a return type of void.
- ***calculateAverage***: Representing a function that takes two arguments: an integer array and its size. It returns a float value representing the average of the elements in the array.
- ***array***: Representing an integer array passed as an argument to the *calculateAverage* function.
- ***size***: Representing the size of the array passed as an argument to the *calculateAverage* function.
- ***getLargest***: Representing a function that takes two integer arguments and returns an integer value representing the larger of the two values.
- ***a***: Representing the first integer argument passed to the getLargest function.
- ***b***: Representing the second integer argument passed to the getLargest function.

Rules for Naming Identifiers in C

Some important rules for naming identifiers in C are:

1. Identifiers must begin with a letter or an underscore.

This means the first character of an identifier must be a letter (uppercase or lowercase) or an underscore.

It cannot start with a digit or a special character, such as +, -, *, /, or %.

For example, *1variable* or *$name* are not valid identifiers in C.

2. Identifiers can contain letters, digits, and underscores.

After the first character, an identifier can contain any combination of letters, digits, and underscores.

No spaces or other special characters are allowed in an identifier.

For example, *my variable* or *my-variable* are not valid identifiers in C.

3. Identifiers are case-sensitive.

This means that uppercase and lowercase letters are considered different.

For example, *myVariable* and *MyVariable* are two different identifiers in C.

4. Identifiers cannot be C keywords.

Keywords in C are reserved words that have a special meaning in the language.

For example, if, else, while, for, int, float, double, char, and void are all C keywords.

Identifiers cannot have the same name as a keyword, or they will cause a compilation error.

Overall, it's important to choose meaningful and descriptive names for your identifiers in C. Using descriptive names will make your code more readable and easier to understand for both yourself and others who may need to work with your code in the future.

What Are Valid Identifiers in C?

In C programming language, a valid identifier is a name given to a variable, function, constant, label, or any other user-defined item that follows the rules for naming identifiers in C.

Here are some examples of valid identifiers in C:

- *myVariable*
- *_temp*
- *MAX_VALUE*
- *function123*
- *a1_b2_c3*

What Are Invalid Identifiers in C?

An identifier in C is considered invalid if it:

- Starts with a digit.
- Contains special characters other than underscore, such as !, @, #, $, %, ^, &, *, (,), +, =, {, }, [,], |, , :, ;, ", ', <, >, ?, ,, ., /.

- Exceeds the maximum allowed length of 31 characters.

Here are some examples of invalid identifiers in C:

- *123var*
- *my$var*
- *for*
- *longIdentifierThatIsWayTooLongToBeAValidIdentifier*

It's important to use valid identifiers in your C programs, as invalid identifiers will result in compile errors.

Different Types of Identifiers in C

There are two types of identifiers in C language: **internal** and **external**.

1. Internal Identifiers

Internal identifiers are those that are declared within a function or block of code and have local scope.

They are only visible and accessible within the function or block where they are declared. Once the function or block is exited, the internal identifier is no longer accessible.

2. External Identifiers

External identifiers are those that are declared outside any function or block of code and have global scope.

They are visible and accessible to all functions within the same file. If an external identifier is declared with the keyword extern, it can also be accessed by functions in other files.

Internal vs External Identifiers in C Programming

Here are the differences between internal and external identifiers in C with tabular comparison:

Feature	**Internal Identifier**	**External Identifier**
Scope	Local	Global
Visibility	Only visible within the function or block where it is declared	Visible to all functions within the same file

Accessibility	Only accessible within the function or block where it is declared	Accessible by any function within the same file, or in other files if declared with extern
Lifetime	Exists only while the function or block is being executed	Exists throughout the execution of the program
Naming Convention	No special naming convention required	Should begin with a letter or underscore, and may contain letters, digits, and underscores
Usefulness	Used for variables or functions that are only needed within a specific function or block	Used for variables or functions that are needed across multiple functions or files
Potential Issues	May lead to variable name conflicts if multiple functions or blocks use the same name	May lead to naming collisions with other global variables or functions, and can make code harder to understand and maintain
Best Practices	Use internal identifiers whenever possible to minimize conflicts and improve code clarity	Limit the use of external identifiers as much as possible, and use appropriate naming conventions to avoid collisions

Useful Tips to Follow While Naming Identifiers in C Programming Language

1. Keep name of Identifiers descriptive and meaningful.

This means that the name of an identifier should give an idea of what the variable, function, or other item represents or does.

For example, instead of using a generic name like "var1" or "foo", it's better to use a more descriptive name like "age" or "calculateAverage".

2. Follow a consistent naming convention.

This means that you should choose a consistent way to name your identifiers across your program or project.

Some common naming conventions include using camelCase, snake_case, or PascalCase for different types of identifiers.

For example, camelCase is commonly used for variable names, while PascalCase is commonly used for function names.

3. Keep identifiers as short as possible, but as long as necessary.

This means that you should try to choose a short and concise name for your identifier, while still making it descriptive and meaningful.

For example, a variable that represents a person's age can be named "age" instead of "personAge" or "theAgeOfThePerson".

These tips will help you to make your code more readable, understandable, and maintainable, especially as your programs and projects become more complex.

Difference Between Keywords and Identifiers in C Language

Here are some of the key differences between keywords and identifiers:

Sr .No.	Keywords	Identifiers
1.	These are predefined and can not be used or defined according to the user.	These are user-defined for several operations.
2.	Lowercase letters only	Either lowercase or uppercase
3.	Alphabetic characters	Alphanumeric characters
4.	Identifies a value that is already in the compiler	Identifies a value that is user-defined.
5.	Examples: int, char, if, do	Examples: count1, Test, high_speed, etc.
6.	Identify the whole class of entity	Identifies a particular entity

Here is an example of keyword and identifier:

int myteam = “tutorialsfreak”

Here *int* is the keyword and *myteam* is the identifier that stores the variable “tutorialsfreak”. In this program, wherever a user writes *myteam*, it will translate to “tutorialsfreak”.