C Identifiers

In C programming, identifiers are the names used to identify variables, functions, arrays, structures, or any other user-defined items. It is a name that uniquely identifies a program element and can be used to refer to it later in the program.

Example:

```
// Creating a variable
int val = 10;

// Creating a function
void func() {}
```

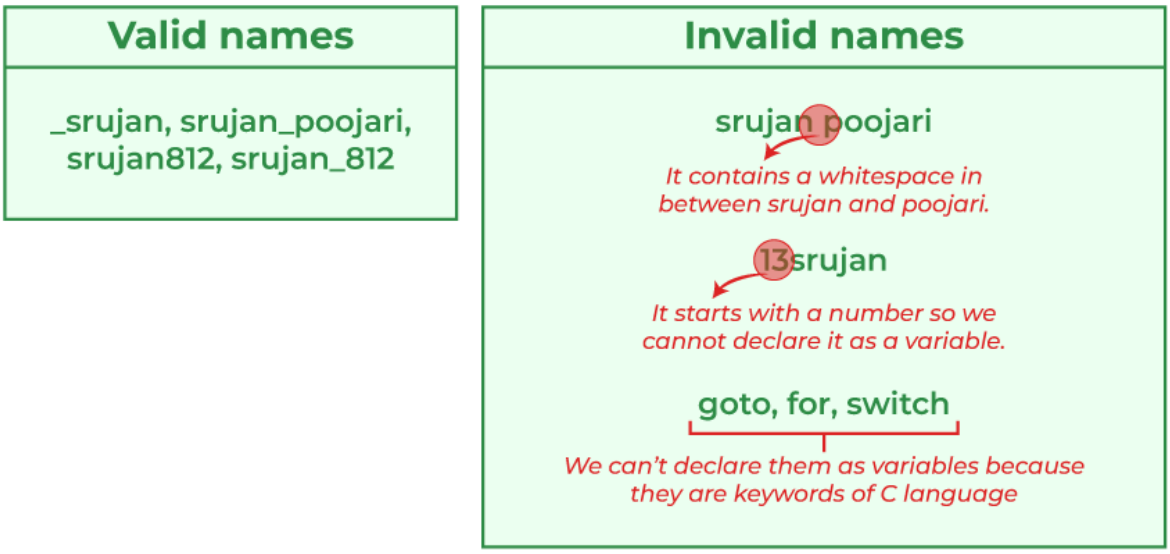
In the above code snippet, "**val**" and "**func**" are identifiers.

Rules for Naming Identifiers in C

A programmer must follow a set of rules to create an identifier in C:

- Identifier can contain following characters:
    - Uppercase (A-Z) and lowercase (a-z) alphabets.
    - Numeric digits (0-9).
    - Underscore (_).
- The first character of an identifier must be a letter or an underscore.
- Identifiers are case-sensitive.
- Identifiers cannot be keywords in C (such as int, return, if, while etc.).

The below image and table show some **valid** and **invalid** identifiers in C language.



Example

The following code examples demonstrate the creation and usage of identifiers in C:

Creating an Identifier for a Variable

```
{...}
```

```
    // Creating an integer variable and
     // assign it the identifier 'var'
     int var;

     // Assigning value to the variable
     // using assigned name
     var = 10;
```

```
    // Referring to same variable using
    // assigned name
    printf("%d", var);
```

```
{...}
```

Output

```
10
```

If you are not familiar with variables and functions, don't worry! We will discuss them in the later sections.

Creating an Identifier for a Function

```
{...}
```

```
// Function declaration which contains user
// defined identifier as it name
int sum(int a, int b) {
    return a + b;
}

int main() {

    // Calling the function using its name
    printf("%d", sum(10, 20));
```

```
{...}
```

Output

```
30
```

Naming Conventions

In C programming, naming conventions are not strict rules but are commonly followed suggestions by the programming community for identifiers to improve readability and understanding of code. Below are some conventions that are commonly used:

For Variables:

- Use camelCase for variable names (e.g., frequencyCount, personName).
- Constants can use UPPER_SNAKE_CASE (e.g., MAX_SIZE, PI).
- Start variable names with a lowercase letter.
- Use descriptive and meaningful names.

For Functions:

- Use camelCase for function names (e.g., getName(), countFrequency()).
- Function names should generally be verbs or verb phrases that describe the action.

For Structures:

- Use PascalCase for structure names (e.g., Car, Person).
- Structure names should be nouns or noun phrases.

Keywords vs Identifiers

Here's a table that highlights the differences between keywords and identifiers in C:

Feature	Keyword	Identifier
Definition	A keyword is a reserved word with a special meaning in C that cannot be used as an identifier.	An identifier is a user-defined name used to refer to variables, functions, arrays, etc.
Usage	Keywords are predefined in the C language and are used to define the structure and control	Identifiers are used by programmers to name variables, functions, and other user-defined

	flow of the program (e.g., int, if, while).	elements (e.g., age, sum, main).
Example	int, return, for, if, while	totalAmount, studentAge, calculateTotal
Modification	Keywords cannot be modified or used for any other purpose.	Identifiers can be created and used as per the programmer's needs.
Position in Code	Keywords are part of the syntax of C and are used to structure the program.	Identifiers are used for variable names, function names, and more throughout the code.
Case Sensitivity	Keywords are case-sensitive (e.g., int and Int are different).	Identifiers are also case-sensitive (e.g., age and Age are different).

What happens if we use a keyword as an Identifier in C?

In the below code, we have used const as an identifier which is a keyword in C. This will result in an error in the output.

```
#include <stdio.h>

int main() {



   // used keyword as an identifier
    int const = 90;


{...}
```

Output

```
./Solution.c: In function 'main':
./Solution.c:5:14: error: expected identifier or '(' before '=' token
    int const = 90;
              ^
```

C Variables

In C, **variable** is a name given to the memory location to easily store data and access it when required. It allows us to use the memory without having to memorize the exact memory address. A variable name can be used anywhere as a substitute in place of the value it stores.

Create Variables

To create a variable in C, we have to specify its **name** and the **type of data** it is going to store. This is called **variable declaration.**

```
data_type name;
```

We can also create multiple variables of same in a single statement by separating them using comma:

```
data_type name1, name2, name3, ...;
```

C provides a set of different data type that are able to store almost all kind of data. For example, integers are stored as **int** type, decimal numbers are stored by **float** and **double** data types, characters are stored as **char** data type.

Rules for Naming Variables

We can assign any name to the variable as long as it follows the following rules:

- A variable name must only contain **alphabets**, **digits**, and **underscore**.
- A variable name must **start with an alphabet** or an **underscore** only. It cannot start with a digit.
- **No white space** is allowed within the variable name.
- A variable name must **not** be any reserved word or **keyword**.

Initialization

No useful data is stored inside a variable that is just declared. We have to initialize it to store some meaningful value to the variable. It is done using assignment operator =.

```
int n;

n = 3;
```

We can also initialize a variable with declaration.

```
int n = 3;
```

Note: *It is compulsory that the values assigned to the variables should be of the same data type as specified in the declaration.*

Accessing

The data stored inside a variable can be easily accessed by using the variable's name. **For example:**

```
{...}


   // Access the value stored in
    // variable
    printf("%d", n);


{...}
```

Output

```
3
```

Updating

We can also update the value of a variable with new value whenever needed by using the assignment operator =.

Example:

```
#include <stdio.h>

int main() {




    // Create integer variable
    int n = 3;

    // Change the stored data
    n = 22;

    // Access the value stored in
    // variable
    printf("%d", n);
    return 0;
}
```

Output

```
22
```

How to use variable?

Variables act as name for memory locations that stores some value. It is valid to use the variable wherever it is valid to use its value. For **example**, an integer variable can be used in a mathematical expression in place of numeric values.

```c
#include <stdio.h>


int main() {


    // Expression that uses values
    int sum1 = 20 + 40;


    // Defining variables
    int a = 20, b = 40;


    // Expression that uses variables
    int sum2 = a + b;


    printf("%d\n%d", sum1, sum2);
    return 0;
}
```

Output

```
60

60
```

Memory Allocation of Variables

When a variable is **declared**, the compiler is told that the variable with the given name and type exists in the program. But no memory is allocated to it yet. Memory is allocated when the variable is **defined**.

Most programming languages like C generally declare and define a variable in the single step. For example, in the above part where we create a variable, variable is declared and defined in a single statement.

The size of memory assigned for variables depends on the type of variable. We can check the size of the variables using **sizeof operator.**

Example:

```
{...}


   int num = 22;


    // Finding size of num

    printf("%d bytes", sizeof(22));
```

```
{...}
```

Output

```
4 bytes
```

Variables are also stored in different parts of the memory based on their storage classes.

Scope of Variable

We have told that a variable can be accessed anywhere once it is declared, but it is partially true. A variable can be accessed using its name anywhere in a specific region of the program called its scope. It is the region of the program where the name assigned to the variable is valid.

A scope is generally the area inside the **{} curly braces.**

Example:

```
// num cannot be accessed here

int main() {

    // num cannot be accessed here
```

```
    {
        // Variable declaration
        int num;
    }

    // Cannot be accessed here either
    return 0;
}
```

Constants

We know that we can change the value stored by any variable anytime, but C also provides some variables whose value cannot be changed. These variables are called **constants** and are created simply by prefixing **const** keyword in variable declaration.

Syntax:

```
const data_type name = value;
```

Constants must be initialized at the time of declaration.