Variables

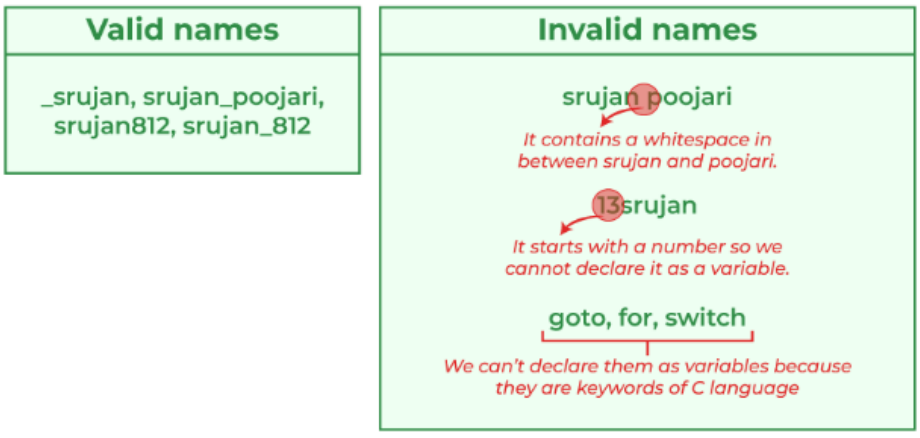
A **variable in C language** is the name associated with some memory location to store data of different types. There are many types of variables in C depending on the scope, storage class, lifetime, type of data they store, etc. A variable is the basic building block of a C program that can be used in expressions as a substitute in place of the value it stores.

- A variable is nothing, but a **name given to a storage area** that our programs can manipulate.
- Each **variable has a specific type**, which determines the **size**.
- the **range of values** that can be **stored within that memory**.
- the set of operations that can be applied to the variable.

Rules for Naming Variables in C

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

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No whitespace is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.

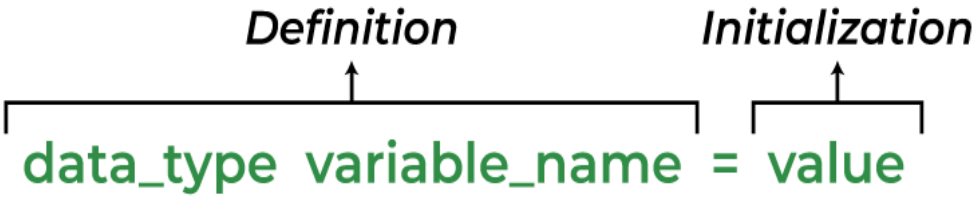


Syntax:

data_type **variable_name = value;** // defining single variable

or

data_type variable_name1, variable_name2; // defining multiple variable



Here,

- **data_type:** Type of data that a variable can store.
- **variable_name:** Name of the variable given by the user.
- **value:** value assigned to the variable by the user.

Example

```
int var;    // integer variable
char a;     // character variable
float fff;  // float variables
```

Note: C is a strongly typed language so all the variables types must be specified before using them.

```
// Student data
int studentID = 15;
int studentAge = 23;
float studentFee = 75.25;
char studentGrade = 'B';

// Print variables
printf("Student id: %d\n", studentID);
printf("Student age: %d\n", studentAge);
printf("Student fee: %f\n", studentFee);
printf("Student grade: %c", studentGrade);
```

There are 3 aspects of defining a variable:

1. Variable Declaration
2. Variable Definition

3. Variable Initialization

C Variable Declaration

Variable declaration in C tells the compiler about the existence of the variable with the given name and data type. When the variable is declared compiler automatically allocates the memory for it.

C Variable Definition

In the definition of a C variable, the compiler allocates some memory and some value to it.
A defined variable will contain some random garbage value till it is not initialized.

Example

```
int var;
char var2;
```

C Variable Initialization

Initialization of a variable is the process where the user assigns some meaningful value to the variable.

Example

```
int var; // variable definition
var = 10; // initialization
      or
int var = 10; // variable declaration and definition
```

Example Program:

```
// C program to demonstrate the
// declaration, definition and
// initialization
#include <stdio.h>

int main()
{
    // declaration with definition
    int defined_var;
     printf("Defined_var: %d\n", defined_var);
```

```
    // initialization
   defined_var = 12;
    // declaration + definition + initialization
   int ini_var = 25;
    printf("Value of defined_var after initialization: %d\n",defined_var);
   printf("Value of ini_var: %d", ini_var);
    return 0;
}
```

C Variable Types

The C variables can be classified into the following types:

1. **Local Variables**
2. **Global Variables**
3. **Static Variables**
4. **Automatic Variables**

Local Variables in C

A **Local variable in C** is a variable that is declared inside a function or a block of code. Its scope is limited to the block or function in which it is declared.

Example of Local Variable in C

```
// C program to declare and print local variable inside a function

#include <stdio.h>
void function()
{
        int x = 10; // local variable
        printf("%d", x);
}
int main()
 {
 function();
}
```

OP:- 10//

Global Variables in C

A **Global variable in C** is a variable that is declared outside the function or a block of code. Its scope is the whole program i.e. we can access the global variable anywhere in the C program after it is declared.

Example of Global Variable in C

```
// C program to demonstrate use of global variable
#include <stdio.h>

int x = 20; // global variable

void function1() { printf("Function 1: %d\n", x); }

void function2() { printf("Function 2: %d\n", x); }

int main()

{


        function1();
        function2();
        return 0;
}
```

Output

Function 1: 20

Function 2: 20

In the above code, both functions can use the global variable as global variables are accessible by all the functions.

Note: When we have same name for local and global variable, local variable will be given preference over the global variable by the compiler.

Static Variables in C

A **static variable in C** is a variable that is defined using the **static** keyword. It can be defined only once in a C program and its scope depends upon the region where it is declared (can be **global or local**).
The **default value** of static variables is **zero.**

Syntax of Static Variable in C

```
static data_type variable_name = initial_value;
```

As its lifetime is till the end of the program, it can retain its value for multiple function calls as shown in the example.

Example of Static Variable in C

```
// C program to demonstrate use of static variable
#include <stdio.h>
```

```
void function()
{
	int x = 20; // local variable
	static int y = 30; // static variable
	x = x + 10;
	y = y + 10;
	printf("\tLocal: %d\n\tStatic: %d\n", x, y);
}

int main()
{
	printf("First Call\n");
	function();
	printf("Second Call\n");
	function();
	printf("Third Call\n");
	function();
	return 0;
}
```

Output

First Call

Local: 30

Static: 40

Second Call

Local: 30

Static: 50

Third Call

Local: 30

Static: 60

In the above example, we can see that the local variable will always print the same value whenever the function will be called whereas the static variable will print the incremented value in each function call.

Automatic Variable in C

All the **local** variables are **automatic** variables **by default**. They are also known as auto variables.

Their scope is **local** and their lifetime is till the end of the **block.** If we need, we can use the **auto** keyword to define the auto variables.

The default value of the auto variables is a garbage value.

Syntax of Auto Variable in C

```
auto data_type variable_name;
      or
data_type variable_name;    (in local scope)
```

Example of auto Variable in C

```
// C program to demonstrate use of automatic variable
#include <stdio.h>
void function()
{
        int x = 10; // local variable (also automatic)
        auto int y = 20; // automatic variable
        printf("Auto Variable: %d", y);
}
int main()
{

        function();
        return 0;
}
```

Output

```
Auto Variable: 20
```

In the above example, both x and y are automatic variables. The only difference is that variable y is explicitly declared with the **auto** keyword.