C++ Variables

Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program.

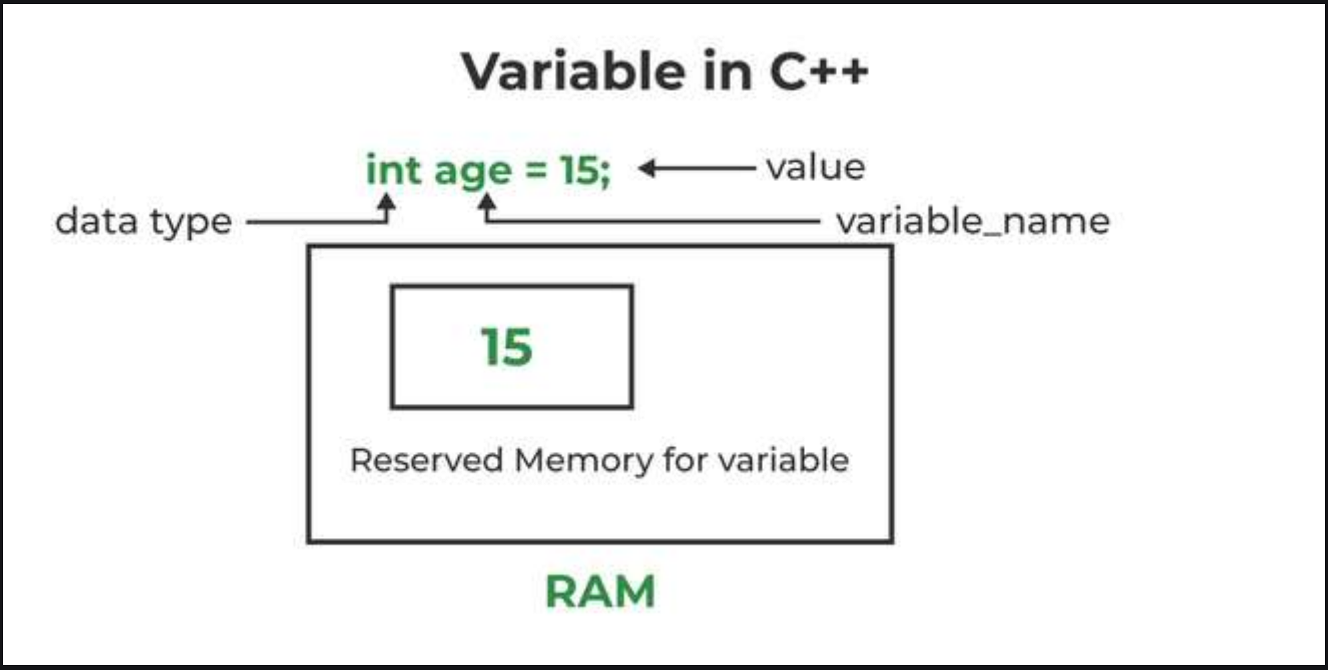
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In C++, all the variables must be declared before use.

How to Declare Variables?

A typical variable declaration is of the form:

```
// Declaring a single variable
type variable_name;


// Declaring multiple variables:
type variable1_name, variable2_name, variable3_name;
```

A variable name can consist of alphabets (both upper and lower case), numbers, and the underscore '_' character. However, the name must not start with a number.



Initialization of a variable in C++

In the above diagram,

***datatype**: Type of data that can be stored in this variable.*
***variable_name**: Name given to the variable.*
***value**: It is the initial value stored in the variable.*

Examples:

```
// Declaring float variable
float simpleInterest;
```

```
// Declaring integer variable
int time, speed;


// Declaring character variable
char var;
```

We can also provide values while declaring the variables as given below:

```
int a=50,b=100;  //declaring 2 variable of integer type
float f=50.8;  //declaring 1 variable of float type
char c='Z';    //declaring 1 variable of char type
```

Rules For Declaring Variable

- The name of the variable contains letters, digits, and underscores.
- The name of the variable is case sensitive (ex Arr and arr both are different variables).
- The name of the variable does not contain any whitespace and special characters (ex #,$,%,*, etc).
- All the variable names must begin with a letter of the alphabet or an underscore(_).
- We cannot used C++ keyword(ex float,double,class)as a variable name.

Valid variable names:

```
int x;   //can be letters
int _yz; //can be underscores
int z40;//can be letters
```

Invalid variable names:

```
int 89; Should not be a number
int a b; //Should not contain any whitespace
int double;// C++ keyword CAN NOT BE USED
```

Difference Between Variable Declaration and Definition

The **variable declaration** refers to the part where a variable is first declared or introduced before its first use. A **variable definition** is a part where the variable is assigned a memory location and a value. Most of the time, variable declaration and definition are done together.
See the following C++ program for better clarification:

- C++

```
// C++ program to show difference between
// definition and declaration of a
```

```cpp
// variable
#include <iostream>
using namespace std;

int main()
{
    // this is declaration of variable a
    int a;

    // this is initialisation of a
    a = 10;

    // this is definition = declaration + initialisation
    int b = 20;

    // declaration and definition
    // of variable 'a123'
    char a123 = 'a';

    // This is also both declaration and definition
    // as 'c' is allocated memory and
    // assigned some garbage value.
    float c;

    // multiple declarations and definitions
    int _c, _d45, e;

    // Let us print a variable
    cout << a123 << endl;

    return 0;
}
```

Output

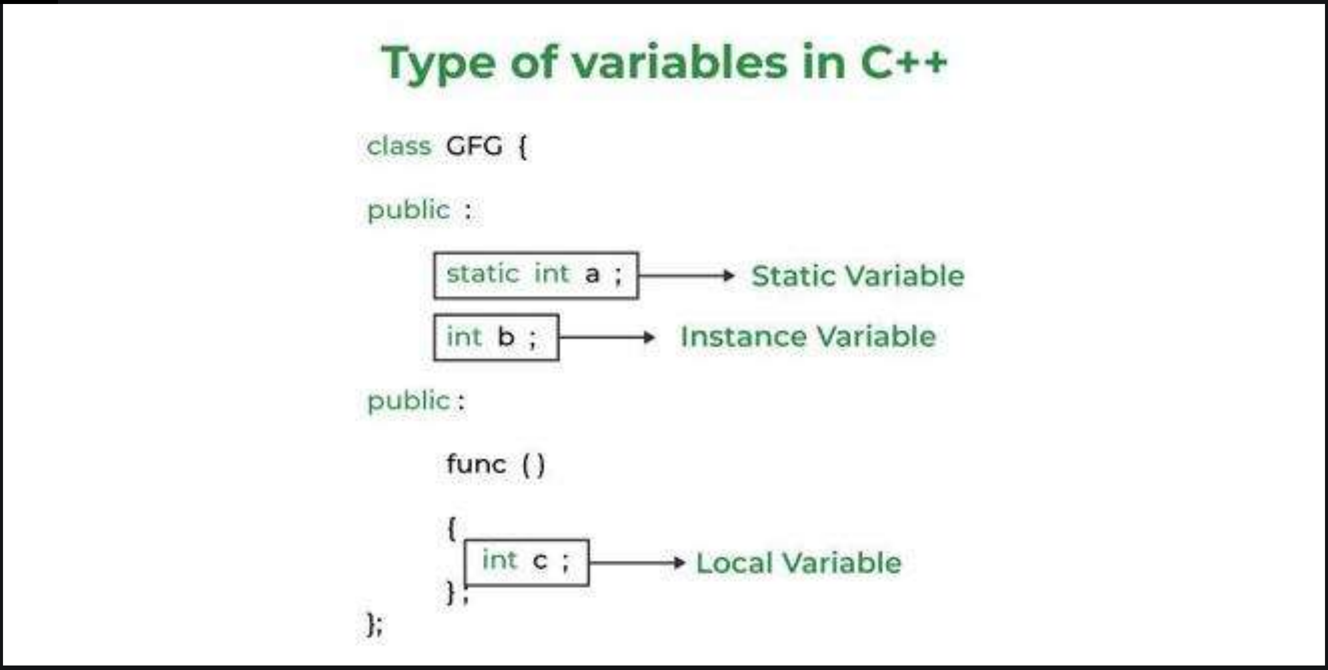
a

Time Complexity: O(1)
Space Complexity: O(1)

Types of Variables

There are three types of variables based on the scope of variables in C++

- **Local Variables**
- **Instance Variables**
- **Static Variables**



Types of Variables in C++

Let us now learn about each one of these variables in detail.

1. **Local Variables**: A variable defined within a block or method or constructor is called a local variable.
    - These variables are created when entered into the block or the function is called and destroyed after exiting from the block or when the call returns from the function.
    - The scope of these variables exists only within the block in which the variable is declared. i.e. we can access this variable only within that block.
    - Initialization of Local Variable is Mandatory.
2. **Instance Variables**: Instance variables are non-static variables and are declared in a class outside any method, constructor, or block.
    - As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.

- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
- Initialization of Instance Variable is not Mandatory.
- Instance Variable can be accessed only by creating objects.

3. **Static Variables**: Static variables are also known as Class variables.
    - These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
    - Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
    - Static variables are created at the start of program execution and destroyed automatically when execution ends.
    - Initialization of Static Variable is not Mandatory. Its default value is 0
    - If we access the static variable like the Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name with the class name automatically.
    - If we access the static variable without the class name, the Compiler will automatically append the class name.

Instance Variable Vs Static Variable

- Each object will have its **own copy** of the instance variable whereas We can only have **one copy** of a static variable per class irrespective of how many objects we create.
- Changes made in an instance variable using one object will **not be reflected** in other objects as each object has its own copy of the instance variable. In the case of static, changes **will be reflected** in other objects as static variables are common to all objects of a class.
- We can access instance variables **through object references** and Static Variables can be accessed **directly using the class name.**
- The syntax for static and instance variables:

```
class Example
{
    static int a; // static variable
    int b;        // instance variable
}
```

Constants in C

The constants in C are the read-only variables whose values cannot be modified once they are declared in the C program. The type of constant can be an integer constant, a floating pointer constant, a string constant, or a character constant. In C language, the **const** keyword is used to define the constants.

What is a constant in C?

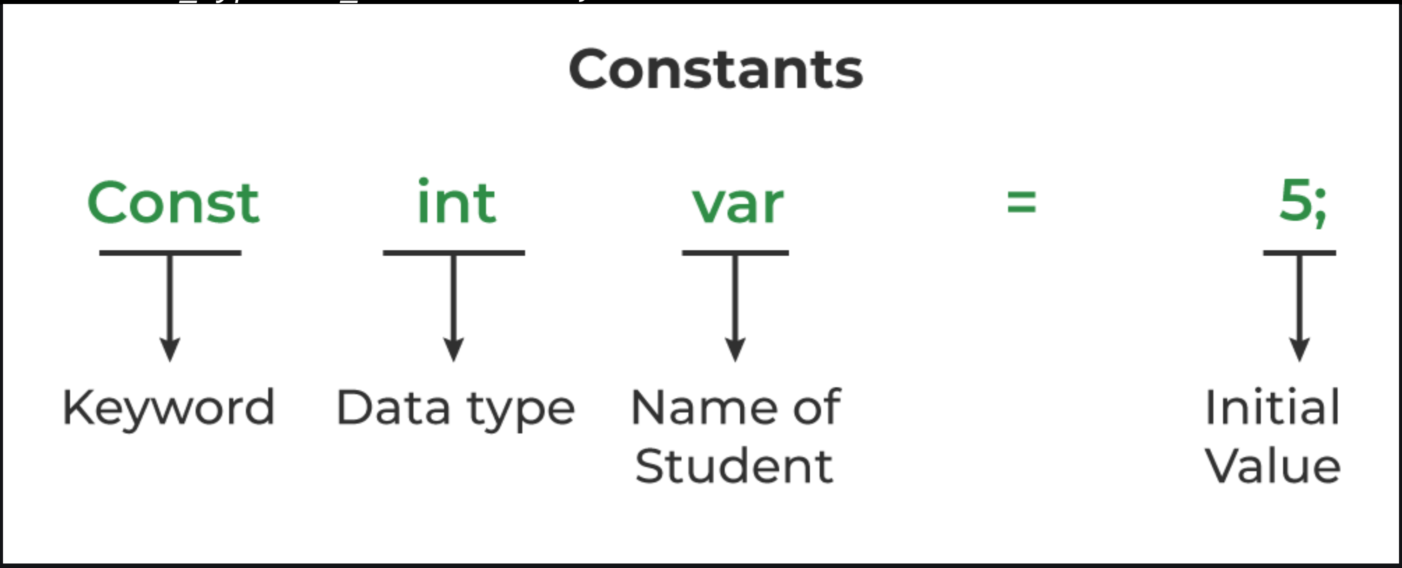
As the name suggests, a constant in C is a variable that cannot be modified once it is declared in the program. We can not make any change in the value of the constant variables after they are defined.

How to define a constant in C?

We define a constant in C language using the **const** keyword. Also known as a const type qualifier, the const keyword is placed at the start of the variable declaration to declare that variable as a constant.

Syntax to Define Constant

```
const data_type var_name = value;
```



Example of Constants in C

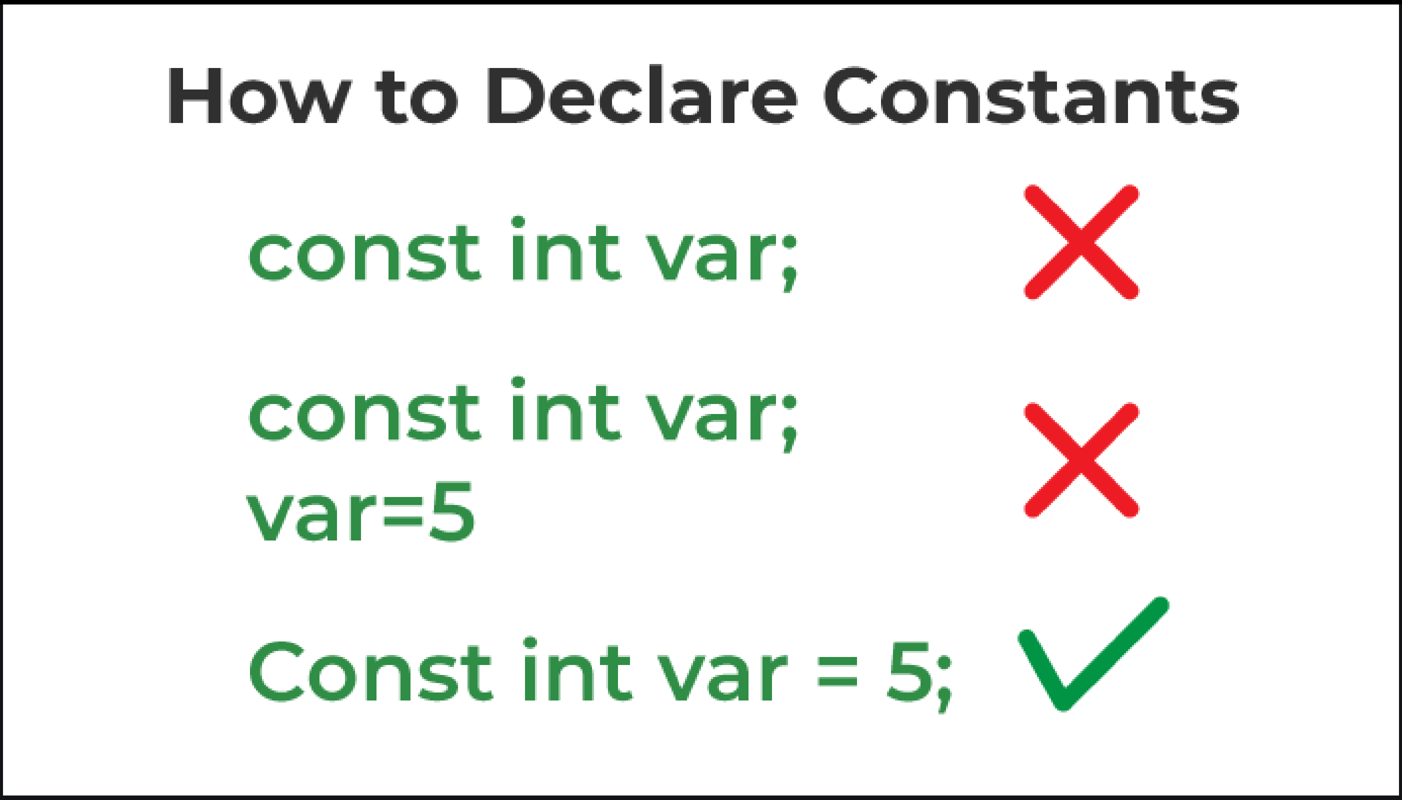
- C

```
// C program to illustrate constant variable definition

#include <stdio.h>


int main()

{
    // defining integer constant using const keyword
    const int int_const = 25;

    // defining character constant using const keyword
    const char char_const = 'A';

    // defining float constant using const keyword
    const float float_const = 15.66;

    printf("Printing value of Integer Constant: %d\n",
           int_const);
    printf("Printing value of Character Constant: %c\n",
           char_const);
    printf("Printing value of Float Constant: %f",
           float_const);

    return 0;
}
```

Output

```
Printing value of Integer Constant: 25
Printing value of Character Constant: A
Printing value of Float Constant: 15.660000
```

One thing to note here is that we have to **initialize the constant variables at declaration**. Otherwise, the variable will store some garbage value and we won't be able to change it. The following image describes examples of incorrect and correct variable definitions.



Types of Constants in C

The type of the constant is the same as the data type of the variables. Following is the list of the types of constants

- Integer Constant
- Character Constant
- Floating Point Constant
- Double Precision Floating Point Constant
- Array Constant
- Structure Constant

We just have to add the const keyword at the start of the variable declaration.

Properties of Constant in C

The important properties of constant variables in C defined using the const keyword are as follows:

1. Initialization with Declaration

We can only initialize the constant variable in C at the time of its declaration. Otherwise, it will store the garbage value.

2. Immutability

The constant variables in c are immutable after its definition, i.e., they can be initialized only once in the whole program. After that, we cannot modify the value stored inside that variable.

- C

```c
// C Program to demonstrate the behaviour of constant
// variable
#include <stdio.h>

int main()
{
    // declaring a constant variable
    const int var;
    // initializing constant variable var after declaration
    var = 20;

    printf("Value of var: %d", var);
    return 0;
}
```

Output

```
In function 'main':
10:9: error: assignment of read-only variable 'var'
10 |     var = 20;
   |         ^
```

Difference between Constants and Literals

The constant and literals are often confused as the same. But in C language, they are different entities and have different semantics. The following table list the differences between the constants and literals in C:

Constant	Literals
Constants are variables that cannot be modified once declared.	Literals are the fixed values that define themselves.

Constant	Literals
Constants are defined by using the const keyword in C. They store literal values in themselves.	They themselves are the values that are assigned to the variables or constants.
We can determine the address of constants.	We cannot determine the address of a literal except string literal.
They are lvalues.	They are rvalues.
Example: const int c = 20.	Example: 24,15.5, ‘a’, “Geeks”, etc.

Defining Constant using #define Preprocessor

We can also define a constant in C using #define preprocessor. The constant defined using #define are macros that behaves like a constant. These constants are not handled by the compiler, they are handled by the preprocessor and are replaced by their value before complication.

Syntax of Constant in C using #define

```
#define const_name value
```

Example

- C

```c
// C Program to define a constant using #define
#include <stdio.h>
#define pi 3.14

int main()
{

    printf("The value of pi: %.2f", pi);
    return 0;
}
```

Output

```
The value of pi: 3.14
```

FAQs on C Constants

1. Define C Constants.

Constants in C are the immutable variables whose values cannot be modified once they are declared in the C program.

2. What is the use of the const keyword?

The const keyword is the qualifier that is used to declare the constant variable in C language.

3. Can we initialize the constant variable after the declaration?

No, we cannot initialize the constant variable once it is declared.

4. What is the right way to declare the constant in C?

The right way to declare a constant in C is to always initialize the constant variable when we declare.

5. What is the difference between constant defined using const qualifier and #define?

The following table list the differences between the constants defined using const qualifier and #define in C:

Constants using const	Constants using #define
They are the variables that are immutable	They are the macros that are replaced by their value.
They are handled by the compiler.	They are handled by the preprocessor.
Syntax: **const *type name = value*;**	Syntax: **#define *name value***

6. Is there any way to change the value of a constant variable in C?

Yes, we can take advantage of the loophole created by pointers to change the value of a variable declared as a constant in C. The below C program demonstrates how to do it.

- C

```
// C Program to change the value of constant variable
#include <stdio.h>


int main()
{


    // defining an integer constant
    const int var = 10;


    printf("Initial Value of Constant: %d\n", var);


    // defining a pointer to that const variable
    int* ptr = &var;
    // changing value
    *ptr = 500;
    printf("Final Value of Constant: %d", var);
    return 0;
}
```

Output

```
Initial Value of Constant: 10
Final Value of Constant: 500
```