## Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

1. type variable_list;

The example of declaring the variable is given below:

1. **int** a;
2. **float** b;
3. **char** c;

Here, a, b, c are variables. The int, float, char are the data types.We can also provide values while declaring the variables as given below:

1. **int** a=10,b=20;        //declaring 2 variable of integer type
2. **float** f=20.8;
3. **char** c='A';

## Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

**Valid variable names:**

1. **int** a;
2. **int** _ab;
3. **int** a=30;
4. **char** c=’a’;

**Invalid variable names:**

```
int 2;
int a b;
int long;
```

**Types of Variables in C**

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

**Local Variable**

A variable that is declared inside the function or block is called a local variable.It must be declared at the start of the block.

```
void function1(){
int x=10;//local variable
}
```

You must have to initialize the local variable before it is used.

**Global Variable**

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
int value=20;//global variable
void function1(){
int x=10;//local variable
}
```

**Static Variable**

A variable that is declared with the static keyword is called static variable.It retains its value between multiple function calls.

```
void function1(){
int x=10;//local variable
static int y=10;//static variable
x=x+1;
y=y+1;
printf("%d,%d",x,y);
}
```

If you call this function many times, the **local variable will print the same value** for each function call, e.g, 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13 and so on.

**Automatic Variable**

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```
void main(){
int x=10;//local variable (also automatic)
auto int y=20;//automatic variable
}
```

**External Variable**

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

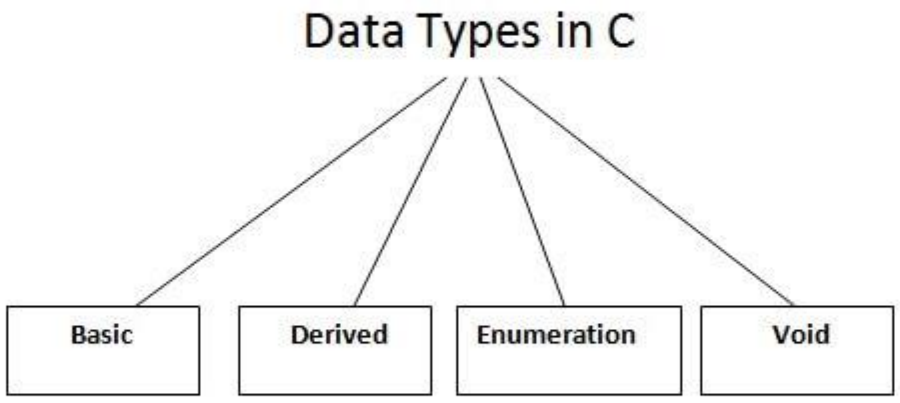
```
extern int x=10;//external variable (also global)
```

program1.c

```
#include "myfile.h"
#include <stdio.h>
void printValue(){
    printf("Global variable: %d", global_variable);
}
```

**Data types in c:**

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



There are the following data types in C language.

| Types | Data Types |
|---|---|
| Basic Data Type | int, char, float, double |
| Derived Data Type | array, pointer, structure, union |
| Enumeration Data Type | enum |
| Void Data Type | void |

**Basic Data Types**

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

| Data Types | Memory Size | Range |
|---|---|---|
| **char** | 1 byte | −128 to 127 |
| signed char | 1 byte | −128 to 127 |
| unsigned char | 1 byte | 0 to 255 |
| **short** | 2 byte | −32,768 to 32,767 |
| signed short | 2 byte | −32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 65,535 |
| **int** | 2 byte | −32,768 to 32,767 |
| signed int | 2 byte | −32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 65,535 |
| **short int** | 2 byte | −32,768 to 32,767 |
| signed short int | 2 byte | −32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 65,535 |
| **long int** | 4 byte | -2,147,483,648 to 2,147,483,647 |
| signed long int | 4 byte | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 byte | 0 to 4,294,967,295 |
| **float** | 4 byte | |
| **double** | 8 byte | |
| **long double** | 10 byte | |

**Keywords in C**

A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

| | | | | | | |
|---|---|---|---|---|---|---|
| auto | break | case | char | const | continue | default |
| double | else | enum | extern | float | for | goto |
| int | long | register | return | short | signed | sizeof |
| struct | switch | typedef | union | unsigned | void | volatile |

**C Identifiers**

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore. If the identifier is not used in the external linkage, then it is called as an internal identifier. If the identifier is used in the external linkage, then it is called as an external identifier.

We can say that an identifier is a collection of alphanumeric characters that begins either with an alphabetical character or an underscore, which are used to represent various programming elements such as variables, functions, arrays, structures, unions, labels, etc. There are 52 alphabetical characters (uppercase and lowercase), underscore character, and ten numerical digits (0-9) that represent the identifiers. There is a total of 63 alphanumerical characters that represent the identifiers.

## Rules for constructing C identifiers

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.

- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

**Example of valid identifiers**

total, sum, average, _m _, sum_1, etc.

**Example of invalid identifiers**

1. 2sum (starts with a numerical digit)
2. **int** (reserved word)
3. **char** (reserved word)
4. m+n (special character, i.e., '+')

**Types of identifiers**

- Internal identifier
- External identifier

**Internal Identifier**

If the identifier is not used in the external linkage, then it is known as an internal identifier. The internal identifiers can be local variables.

**External Identifier**

If the identifier is used in the external linkage, then it is known as an external identifier. The external identifiers can be function names, global variables.

**Differences between Keyword and Identifier**

| Keyword | Identifier |
|---|---|
| Keyword is a pre-defined word. | The identifier is a user-defined word |
| It must be written in a lowercase letter. | It can be written in both lowercase and uppercase letters. |
| Its meaning is pre-defined in the c compiler. | Its meaning is not defined in the c compiler. |
| It is a combination of alphabetical characters. | It is a combination of alphanumeric characters. |
| It does not contain the underscore character. | It can contain the underscore character. |

**Let's understand through an example.**

```
int main()
{
    int a=10;
    int A=20;
    printf("Value of a is : %d",a);
    printf("\nValue of A is :%d",A);
    return 0;
}
```

**Output**

```
Value of a is : 10
Value of A is :20
```

The above output shows that the values of both the variables, 'a' and 'A' are different. Therefore, we conclude that the identifiers are case sensitive.

## C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator
- Misc Operator

## Precedence of Operators in C:

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

**int** value=10+20*10;

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

## Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

## Single Line Comments:

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

```
#include<stdio.h>
int main(){
   //printing information
   printf("Hello C");
return 0;
}
```

## Multi-Line Comments:

Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax:

```
/*
code
to be commented
*/
```

Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main(){
   /*printing information
     Multi-Line Comment*/
   printf("Hello C");
return 0;
}
```

## C Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

**%d**

```
int main()
{
 int b=6;
 int c=8;
 printf("Value of b is:%d", b);
 printf("\nValue of c is:%d",c);

   return 0;
}
```

In the above code, we are printing the integer value of b and c by using the %d specifier.

- **%u**

```
int main()
{
 int b=10;
 int c= -10;
 printf("Value of b is:%u", b);
 printf("\nValue of c is:%u",c);

   return 0;
}
```

In the above program, we are displaying the value of b and c by using an unsigned format specifier, i.e., %u. The value of b is positive, so %u specifier prints the exact value of b, but it does not print the value of c as c contains the negative value.

- **%o**

```
int main()
{
  int a=0100;
  printf("Octal value of a is: %o", a);
  printf("\nInteger value of a is: %d",a);
  return 0;
}
```

In the above code, we are displaying the octal value and integer value of a.

- **%x and %X**

```
int main()
{
  int y=0xA;
  printf("Hexadecimal value of y is: %x", y);
  printf("\nHexadecimal value of y is: %X",y);
  printf("\nInteger value of y is: %d",y);
    return 0;
```

In the above code, y contains the hexadecimal value 'A'. We display the hexadecimal value of y in two formats. We use %x and %X to print the hexadecimal value where %x displays the value in small letters, i.e., 'a' and %X displays the value in a capital letter, i.e., 'A'.

- **%f**

```
int main()
{
  float y=3.4;
  printf("Floating point value of y is: %f", y);
  return 0;
}
```

The above code prints the floating value of y.

- **%e**

```
int main()
{
  float y=3;
  printf("Exponential value of y is: %e", y);
  return 0;
}
```

- **%E**

```
int main()
{
  float y=3;
  printf("Exponential value of y is: %E", y);
  return 0;
}
```

- **%g**

```
int main()
{
  float y=3.8;
  printf("Float value of y is: %g", y);
  return 0;
}
```

In the above code, we are displaying the floating value of y by using %g specifier. The %g specifier displays the output same as the input with a same precision.

- **%p**

```
int main()
{
  int y=5;
  printf("Address value of y in hexadecimal form is: %p", &y);
  return 0;
}
```

- 
- **%c**

```c
int main()
{
  char a='c';
  printf("Value of a is: %c", a);
  return 0;
}
```

- **%s**

```c
int main()
{
  printf("%s", "javaTpoint");
  return 0;
}
```

Minimum Field Width Specifier

Suppose we want to display an output that occupies a minimum number of spaces on the screen. You can achieve this by displaying an integer number after the percent sign of the format specifier.

```c
int main()
{
 int x=900;
  printf("%8d", x);
  printf("\n%-8d",x);
  return 0;
}
```

In the above program, %8d specifier displays the value after 8 spaces while %-8d specifier will make a value left-aligned.

**Now we will see how to fill the empty spaces. It is shown in the below code:**

```c
int main()
{
 int x=12;
  printf("%08d", x);
  return 0;
}
```

In the above program, %08d means that the empty space is filled with zeroes.

**Escape Sequence in C**

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

## List of Escape Sequences in C:

| Escape Sequence | Meaning |
|---|---|
| \a | Alarm or Beep |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab (Horizontal) |
| \v | Vertical Tab |
| \\ | Backslash |
| \' | Single Quote |
| \" | Double Quote |
| \? | Question Mark |

| | |
|---|---|
| \nnn | octal number |
| \xhh | hexadecimal number |
| \0 | Null |

**Escape Sequence Example:**

```
#include<stdio.h>
int main(){
     int number=50;
    printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
return 0;
}
```

**Output:**

```
You
are
learning
'c' language
"Do you know C language"
```

**Constants in C**

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

## List of Constants in C:

| **Constant** | **Example** |
|---|---|

| | |
|---|---|
| Decimal Constant | 10, 20, 450 etc. |
| Real or Floating-point Constant | 10.3, 20.2, 450.6 etc. |
| Octal Constant | 021, 033, 046 etc. |
| Hexadecimal Constant | 0x2a, 0x7b, 0xaa etc. |
| Character Constant | 'a', 'b', 'x' etc. |
| String Constant | "c", "c program", "c in javatpoint" etc. |

## 2 ways to define constant in C:

There are two ways to define constant in C programming.

1. const keyword
2. #define preprocessor

1) **C const keyword**

The const keyword is used to define constant in C programming.

```
const float PI=3.14;
```

Now, the value of PI variable can't be changed.

```
#include<stdio.h>
int main(){
    const float PI=3.14;
    printf("The value of PI is: %f",PI);
    return 0;
}
```

**Output:**

```
The value of PI is: 3.140000
```

If you try to change the the value of PI, it will render compile time error.

```
#include<stdio.h>
int main(){
const float PI=3.14;
PI=4.5;
printf("The value of PI is: %f",PI);
    return 0;
}
```

**Output:**

```
Compile Time Error: Cannot modify a const object
```

2) **C #define preprocessor:**

The #define preprocessor is also used to define constant.

**C #define**

The #define preprocessor directive is used to define constant or micro substitution. It can use any basic data type.

**Syntax:**

```
#define token value
```

```
 #include <stdio.h>
 #define PI 3.14
 main()
{
        printf("%f",PI);
}
```

**Output:**

```
3.140000
```

```
#include <stdio.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
void main() {
    printf("Minimum between 10 and 20 is: %d\n", MIN(10,20));
}
```

**Output:**

```
Minimum between 10 and 20 is: 10
```

## Tokens in C

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

### Classification of tokens in C

Tokens in C language can be divided into the following categories:

- Keywords in C
- Identifiers in C
- Strings in C
- Operators in C
- Constant in C
- Special Characters in C

### Keywords in C

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names. If the keywords are used as the variable names, it means that we are assigning a different meaning to the keyword, which is not allowed. C language supports 32 keywords given below:

| | | | |
|---|---|---|---|
| **auto** | **double** | **int** | **struct** |

| | | | |
|---|---|---|---|
| **break** | **else** | **long** | **switch** |
| **case** | **enum** | **register** | **typedef** |
| **char** | **extern** | **return** | **union** |
| **const** | **float** | **short** | **unsigned** |
| **continue** | **for** | **signed** | **void** |
| **default** | **goto** | **sizeof** | **volatile** |
| **do** | **if** | **static** | **while** |

## Identifiers in C

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

## Strings in C

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

char a[10] = "javatpoint"; // The compiler allocates the 10 bytes to the 'a' array.

char a[] = "javatpoint"; // The compiler allocates the memory at the run time.

char a[10] = {'j','a','v','a','t','p','o','i','n','t','\0'}; // String is represented in the form of characters.

**Operators in C**

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

**Unary Operator**

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

**Binary Operator**

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Conditional Operators
- Assignment Operator
- Misc Operator

**Constants in C**

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor

**Types of constants in C**

| **Constant** | **Example** |
|---|---|
| **Integer constant** | **10, 11, 34, etc.** |
| **Floating-point constant** | **45.6, 67.8, 11.2, etc.** |
| **Octal constant** | **011, 088, 022, etc.** |
| **Hexadecimal constant** | **0x1a, 0x4b, 0x6b, etc.** |
| **Character constant** | **'a', 'b', 'c', etc.** |
| **String constant** | **"java", "c++", ".net", etc.** |

**Special characters in C**

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.

- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.