Numeric Data Type:

They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.

a) **Integer:** In C programming, **int** stands for **integer**, or a non-decimal numeric value. For example, -38, 15 and 0 are all *int* values. An *int* type is stored as 2 or 4 bytes. If you really need to keep it to 2 bytes and use the lower range, you can use a **short** data type. And if you want to be sure you get the value up to 2 million-plus, use the **long** data type.
The following table provides the details of standard integer types with their storage sizes and value ranges –

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

b) **Floating Point:** A **float** type is used to store a floating-point number; that is a number with a decimal.
The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Non-Numeric Data Type:

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

String: Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'. The C language does not provide an inbuilt data type for strings but it has an access specifier "**%s**" which can be used to directly print and read strings.

A string is a data type used in programming, such as an integer and floating-point unit, but is used to represent text rather than numbers. It is comprised of a set of characters that can also contain spaces and numbers. For example, the word "hamburger" and the phrase "I ate 3 hamburgers" are both strings. Even "12345" could be considered a string, if specified correctly. Typically, programmers must enclose strings in quotation marks for the data to recognized as a string and not a number or variable name.

Example:

We can use the Sizeof() operator to check the size of a variable. See the following C program for the usage of the various data types:

```
#include <stdio.h>
int main()
{
   int a = 1;
   char b = 'G';
   double c = 3.14;
   printf("Hello World!\n");

   // printing the variables defined
   // above along with their sizes
   printf("Hello! I am a character. My value is %c and "
         "my size is %lu byte.\n",
         b, sizeof(char));
   // can use sizeof(b) above as well

   printf("Hello! I am an integer. My value is %d and "
         "my size is %lu  bytes.\n",
```

```
        a, sizeof(int));
    // can use sizeof(a) above as well

    printf("Hello! I am a double floating-point variable."
        " My value is %lf and my size is %lu bytes.\n",
        c, sizeof(double));
    // can use sizeof(c) above as well


    return 0;
}
```

Output:

Hello World!

Hello! I am a character. My value is G and my size is 1 byte.

Hello! I am an integer. My value is 1 and my size is 4 bytes.

Hello! I am a double floating-point variable. My value is 3.140000 and my size is 8 bytes.

Character value : H

Integer value : 90150

Float value : 3.400000

Keywords

Keywords are predefined, reserved words in C language and each of which is associated with specific features. These words help us to use the functionality of C language. They have special meaning to the compilers.

There are total 32 keywords in C.

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

→ Introduction to C → Dennis Ritchie
→ Structure of the C program ⇓ 1972
→ Input and Output Statements
→ Variables and identifiers
→ Constants, Keywords
→ Values, Name, Scope, Binding.
→ Storage classes → Numeric Data types : Integer, floating point

Dr. Uma

Structure of C Program

Header	#include <stdio.h>
main()	int main() {
Variable declaration	int a = 10;
Body	printf("%d ", a);
Return	return 0; }

Return Statement: The last part of any C program is the return statement. The return statement refers to the returning of the values from a function. This return statement and return value depend upon the return type of the function. For example, if the return type is void, then there will be no return statement. In any other case, there will be a return statement and the return value will be of the type of the specified return type.

Example:

```
int main()
{
   int a;

   printf("%d", a);

   return 0;
}
```

What is meant by Case Sensitivity in C Language?

The answer to the question is c language case sensitive is yes. The C language is case sensitive. This means that all language keywords, identifiers, function names, and other variables must be entered with consistent letter capitalization.

- **C language is case sensitive language.**
- **Case sensitivity in C language helps to compile the C programs faster.**
- **Case sensitivity means both upper case and lower case characters are treated as different.**
- Let us try to print "Hello World" using a C program. We can make use of the printf() function to output in the C language. The syntax of the print function in C is printf(). Changing the capitalization of the syntax will result in an error. This can be seen in the below example.
- **Code:**

```
#include <stdio.h>
int main()
{
    Printf("Hello World"); \\Here the p in printf is in
uppercase
    return 0;
}
```

- **Output:**

```
main.c: In function 'main':
main.c:5:5: warning: implicit declaration of function
'Printf'; did you mean 'printf'? [-Wimplicit-function-
declaration]
    5 |     Printf("Hello World");
      |     ^~~~~~
      |     printf
/usr/bin/ld: /tmp/ccVL48i9.o: in function `main':
main.c:(.text+0x15): undefined reference to `Printf'
collect2: error: ld returned 1 exit status
```

- **Explanation:** If we observe the error shown by the compiler, we can see that in line 5 there is an error, compiler suggests using printf instead of Printf. This example explains C is a case sensitive language. This gives the answer to the question is c language case sensitive.

C Input Output (I/O) functions: Printf ( ) and scanf ( )

- scanf() function to take input from the user, and
- printf() function to display/print output to the user.

C Output

In C programming, `printf()` is one of the main output function. The function sends formatted output to the screen. For example,

Example 1: C Output

```
#include <stdio.h>
int main()
{
    // Displays the string inside quotations
    printf("C Programming");
    return 0;
}
```

Output

```
C Programming
```

How does this program work?

- All valid C programs must contain the `main()` function.
- The code execution begins from the start of the `main()` function.

Programiz
Compiler
run without int also & with return 0.

- The `printf()` is a library function to send formatted output to the screen. The function prints the string inside quotations.
- To use `printf()` in our program, we need to include `stdio.h` header file using the `#include <stdio.h>` statement.
- The `return 0;` statement inside the `main()` function is the "Exit status" of the program. It's optional.

Example 2: Integer Output

Run this code
1. without int
2. without return

```
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

We use `%d` format specifier to print `int` types. Here, the `%d` inside the quotations will be replaced by the value of `testInteger`.

Example 3: float and double Output

```
#include <stdio.h>
int main()
{
    float number1 = 13.5;
    double number2 = 12.4;

    printf("number1 = %f\n", number1);
    printf("number2 = %lf", number2);
    return 0;
}
```

Output

```
number1 = 13.500000
number2 = 12.400000
```

To print `float`, we use `%f` format specifier. Similarly, we use `%lf` to print `double` values.

Example 4: Print Characters

```
#include <stdio.h>
int main()
{
    char chr = 'a';
    printf("character = %c", chr);
    return 0;
}
```

Output

```
character = a
```

To print `char`, we use `%c` format specifier.

C Input

In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

Example 5: Integer Input/Output

```
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

take int input from the user.

Output

```
Enter an integer: 4
Number = 4
```

Here, we have used `%d` format specifier inside the `scanf()` function to take `int` input from the user. When the user enters an integer, it is stored in the `testInteger` variable.

Notice, that we have used `&testInteger` inside `scanf()`. It is because `&testInteger` gets the address of `testInteger`, and the value entered by the user is stored in that address.

Example 6: Float and Double Input/Output

```
#include <stdio.h>
int main()
{
    float num1;
    double num2;

    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);

    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);

    return 0;
}
```

Output

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

We use `%f` and `%lf` format specifier for `float` and `double` respectively.

I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.

```c
#include <stdio.h>
int main()
{
    int a;
    float b;

    printf("Enter integer and then a float: ");

    // Taking multiple inputs
    scanf("%d%f", &a, &b);

    printf("You entered %d and %f", a, b);
    return 0;
}
```

Output

```
Enter integer and then a float: -3
3.4
You entered -3 and 3.400000
```

Format Specifiers for I/O

As you can see from the above examples, we use

- `%d` for `int`
- `%f` for `float`
- `%lf` for `double`
- `%c` for `char`

Here's a list of commonly used C data types and their format specifiers.

Data Type	Format Specifier
int	%d
char	%c
float	%f
double	%lf
short int	%hd
unsigned int	%u
long int	%li
long long int	%lli
unsigned long int	%lu
unsigned long long int	%llu
signed char	%c
unsigned char	%c
long double	%lf

Variable in C

- C variable is a named location in a memory where a program can manipulate the data. This location is used to hold the value of the variable.
- The value of the C variable may get change in the program.
- C variable might be belonging to any of the data type like int, float, char etc

RULES FOR NAMING C VARIABLE:

1. Variable name must begin with letter or underscore.
2. Variables are case sensitive
3. They can be constructed with digits, letters.
4. No special symbols are allowed other than underscore.
5. sum, height, _value are some examples for variable name

DECLARING & INITIALIZING C VARIABLE:

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration. It happens only on variable definition.
- Variable initialization means assigning a value to the variable.

Type	Syntax
Variable declaration	data_type variable_name; Example: int x, y, z; char flat, ch;
Variable initialization	data_type variable_name = value; Example: int x = 50, y = 30; char flag = 'x', ch='l';

```
#include<stdio.h>

int main()

{

  int m = 22, n = 44;


  printf("\nvalues : m = %d and n = %d", m, n);


}
```

C Identifiers

All C **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program. For example:

```
int money;
double accountBalance;
```

Here, `money` and `accountBalance` are identifiers.

Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

```
/ Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The **general rules** for naming variables are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (such as int) cannot be used as names

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.

3. You cannot use keywords like `int`, `while` etc. as identifiers.

Constants

- To define a variable whose value cannot be changed,
- use the `const` keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

Notice, we have added keyword `const`.

Here, `PI` is a symbolic constant; its value cannot be changed.

```
const double PI = 3.14;
PI = 2.9; //Error
```

- C Constants are also like normal variables. But, only difference is, their values can not be modified by the program once they are defined.
- Constants refer to fixed values. They are also called as literals
- Constants may be belonging to any of the data type.

Syntax:

const data_type variable_name; (or) const data_type *variable_name;

TYPES OF C CONSTANT:

1. **Integer constants**
2. **Real or Floating point constants**
3. **Octal & Hexadecimal constants**

4. Character constants
5. String constants
6. Backslash character constants

HOW TO USE CONSTANTS IN A C PROGRAM?

We can define constants in a C program in the following ways:

1. By "const" keyword
2. By "#define" preprocessor directive

```
1  #include <stdio.h>
2  void main()
3  {
4  const int height = 100; /*int constant*/
5  const float number = 3.14; /*Real constant*/
6  const char letter = 'A'; /*char constant*/
7  const char letter_sequence[10] = "ABC"; /*string constant*/
8  const char backslash_char = '\?'; /*special char cnst*/
9  printf("value of height :%d \n", height );
10 printf("value of number : %f \n", number );
11 printf("value of letter : %c \n", letter );
12 printf("value of letter_sequence : %s \n", letter_sequence);
13 printf("value of backslash_char : %c \n", backslash_char);
```

```
14 }
```

EXAMPLE PROGRAM USING #DEFINE PREPROCESSOR DIRECTIVE IN C:

```
1  #include <stdio.h>
2  #define height 100
3  #define number 3.14
4  #define letter 'A'
5  #define letter_sequence "ABC"
6  #define backslash_char '\?'
7  void main()
8  {
9  printf("value of height : %d \n", height );
10 printf("value of number : %f \n", number );
11 printf("value of letter : %c \n", letter );
12 printf("value of letter_sequence : %s \n",letter_sequence);
13 printf("value of backslash_char : %c \n",backslash_char);
14 }
```

C Comments

In programming, comments are hints that a programmer can add to make their code easier to read and understand. For example,

```
#include <stdio.h>
int main() {

  // print Hello World to the screen
  printf("Hello World");
  return 0;
}
```

Output

```
Hello World
```

Here, `// print Hello World to the screen` is a comment in C programming. Comments are completely ignored by C compilers.

Types of Comments

There are two ways to add comments in C:

1. `//` - Single Line Comment
2. `/*...*/` - Multi-line Comment

1. Single-line Comments in C

In C, a single line comment starts with `//`. It starts and ends in the same line. For example,

```
#include <stdio.h>
int main() {

  // create integer variable
  int age = 25;

  // print the age variable
  printf("Age: %d", age);

  return 0;
}
```

Output

```
Age: 25
```

In the above example, `// create integer variable` and `// print the age variable` are two single line comments.

We can also use the single line comment along with the code. For example,

```
int age = 25;  // create integer variable
```

Here, code before `//` are executed and code after `//` are ignored by the compiler.

---

2. Multi-line Comments in C

In C programming, there is another type of comment that allows us to comment on multiple lines at once, they are multi-line comments.

To write multi-line comments, we use the `/*....*/` symbol. For example,

```
/* This program takes age input from the user
It stores it in the age variable
And, print the value using printf() */
#include <stdio.h>
int main() {

  int age;

  printf("Enter the age: ");
  scanf("%d", &age);

  printf("Age = %d", age);
```

```
    return 0;
}
```

Output

```
Enter the age: 24
Age = 24
```

In this type of comment, the C compiler ignores everything from `/*` to `*/`.

Note: Remember the keyboard shortcut to use comments:

- Single Line comment: ctrl + / (windows) and cmd + / (mac)
- Multi line comment: ctrl + shift + / (windows) and cmd + shift + / (mac)

Use of Comments in C

1. Make Code Easier to Understand

If we write comments on our code, it will be easier to understand the code in the future. Otherwise you will end up spending a lot of time looking at our own code and trying to understand it.

Comments are even more important if you are working in a group. It makes it easier for other developers to understand and use your code.