Constants, Variables and Data Types

A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of processing of data is accomplished by executing a sequence of precise instructions called a program. These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules (or grammar). Every program instruction must obey these rules.

C Tokens

The smallest individual unit of a C program is known as a C token.
Example: int a = 6; This is a valid C program statement. It is assigning a value of 6 to an integer type variable called a. Here, we find 5 tokens - int, a, =, 6 and ;

C has six types of tokens. All C programs are written using these tokens and the syntax of the language.

1) Keywords - Keywords serve as basic building blocks for program statements. All keywords have fixed meanings and these meanings cannot be changed. All keywords must be written in lowercase. C has a total of 32 keywords:

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

2) Identifiers - Identifiers refer to names of variables, functions and arrays. These are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. They basically "identify" something. For example: My name, Soumyajit, is an identifier because people are able to identify me using it. Both uppercase (A-Z) and lowercase (a-z) letters are permitted, although lowercase letters are commonly used. The underscore symbol is also permitted in identifiers. It is usually used as a link between two words in long identifiers. The rules for writing identifiers are as follows:
a) It must begin with a letter (A-Z or a-z) or underscore (_)
b) Must not contain whitespace (Ex: sum of two is wrong, sum_of_two is ok)

c) Must consist of only letters, digits or underscore (Ex: abc$ is wrong, ab123c is ok)
d) Must not be a keyword (Ex: int is wrong, int_abc is ok)

3) Constants - Constants in C refer to fixed values that do not change during the execution of a program. Constants are mainly of two types - Numeric constants and Character constants. Numeric constants are further classified into Integer constants and Real constants. Character constants are classified into Single character constants and String constants.

Integer Constants refer to a sequence of digits, primarily decimal integers. Decimal integers are numbers without decimal point. Decimal integers consist of a set of digits, 0 through 9, preceded by an optional - or + sign. Valid examples of decimal integer constants are 123, -321, 0, +78. Embedded spaces, commas and non-digit characters are not permitted between digits. For example, $1000 is an illegal number. The largest integer value that can be stored is machine-dependent. It is 32767 on 16-bit machines and 2147483647 on 32-bit machines. It is also possible to store larger integer constants on these machines by appending qualifiers such as U, L and UL to the constants.

Integer constants are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices and so on. These quantities are represented by numbers containing fractional parts. Such numbers are called real (or floating point) constants. Further examples of real constants are 0.0083, -0.75, 435.36, +247.0. Floating point constants are normally represented as double-precision quantities. However, the suffixes f or F may be used to force single-precision and l or L to extend double precision further.

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Examples of character constants are:
'5' 'X' ';' ' '
The last constant shown above is a blank space.
Character constants have integer values known as ASCII values. Since each

character represents an integer value, it is also possible to perform arithmetic operations on character constants.

A string constant is a sequence of characters enclosed within double quotes. The characters may be letters, numbers, special characters and blank space. Examples include:

"Hello!" "1987" "?...!" "5+3" "X"

A string constant is not the same as a single character constant. The differences include:

i) A single character constant has an equivalent ASCII value. No such value is associated with a string constant.
ii) A single character constant has only one character enclosed within single quotes, whereas a string constant has one or more characters enclosed within double quotes.

4) Variables - A variable is a data name that may be used to store a data value. A variable may take different values at different times during execution. A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Variable names may consist of letters, digits and the underscore character, subject to the following conditions:

i) They must begin with a letter or an underscore.
ii) Uppercase and lowercase letters are significant. For example, Total is not the same as TOTAL.
iii) Variable name should not be a keyword.
iv) White space is not allowed. For example, group one is not a valid variable name because of the blank space in between.

5) Data types - C language is rich in its data types. C supports three classes of data types:

i) Primary data types
ii) Derived data types
iii) User-defined data types

We shall discuss derived data types as a separate chapter later.

C supports five fundamental data types: integer (int), character (char), floating point (float), double precision floating point (double) and void.

Integer types - Integers are whole numbers with a range of values supported by a particular machine. It may be 16 bits or 32 bits. If we use 16 bits, the size of the integer value is limited to the range -32768 to 32767 (-2^{15} to 2^{15} -1). C provides three classes of integer storage, namely short int, int and long int in both signed and unsigned forms. The size of short int on a 16-bit machine is 1 byte (8 bits). The size of int on a 16-bit machine is 2 bytes (16 bits). The size of long int is 4 bytes (32 bits). We declare long and unsigned to increase the range of values. The use of qualifier signed on integers is optional because the default declaration assumes a signed number.

Floating Point types - Floating point (or real) numbers are stored in 32 bits (on all 16 and 32-bit machines), with 6 digits of precision. Floating point numbers are in C by the keyword float. When the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Both double and float represent the same data type. However, the precision of double is more than that of float. To extend the precision even further, we may use long double which uses 80 bits.

Void types - The void type has no value. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

Character types - A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 and 255, signed chars have values from -128 to 127.

6) Declaration of variables - Declaration does two things:
i) Tells the compiler about the variable name.

ii) Specifies what type of data the variable will hold.
A variable can be used to store value of any data type. The name of the variable has nothing to do with the type. The syntax for declaring a variable is as follows:

data-type v1, v2, v3, ... , vN;

v1, v2, v3, ..., vN are names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example, valid declarations include:

```
int count;
int number, total;
double ratio;
```

7) User-defined type declaration - C supports a feature known as "type definition" that allows users to define an identifier that would represent an existing data type. The user-defined data type identifier can later be used to declare variables. It takes the general form:

typedef type identifier;

Where" type" refers to an existing data type and "identifier" refers to the new name given to the data type. However, this does not create a new data type. For example:

```
typedef int units;
unit batch1, batch2;
```

Another user-defined data type is enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces (known as enumeration constants). Example:

```
enum day {Monday, Tuesday, Wednesday};  //creating the enum type
enum day week_start, week_end;          //declaring variables of enum type day
```

Integer constants are automatically assigned to the enumeration constants - Monday is assigned 0, Tuesday is assigned 1, Wednesday is assigned 2. However, these constants may be overridden and explicit constants may be assigned. For example:

enum day {Monday=1, Tuesday, Wednesday};

Now Monday has the value 1. Automatically, Tuesday gets the next value 2 and Wednesday gets the value 3.

8) Assigning values to variables - We may initialize a variable with values. For example:

int a = 5; //assigning an integer constant 5 to an integer variable a

9) Declaring a variable as constant - We may like the value of certain variables to remain constant during the execution of a program. This can be done by declaring the variable with the qualifier const at the time of initialization. Example:

const int class_size = 50;