Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-2nd Year **Paper-III**

Variables, Expressions, Indentifiers, Keywords, Data Types, Constants

VARIABLES

Variables are names that are used to store values. It can take different values but one at a time. A data type is associated with each variable & it decides what values the variable can take. When you decide your program needs another variable, you simply declare (or define) a new variable and C makes sure you get it. You declare all C variables at the top of whatever blocks of code need them. Variable declaration requires that you inform C of the variable's name and data type.

Syntax – datatype variablename;

Eg:

```
int page_no;
char grade;
float salary;
long y;
```

Declaring Variables:

There are two places where you can declare a variable:

- After the opening brace of a block of code (usually at the top of a function)
- Before a function name (such as before main() in the program) Consider various examples:

Suppose you had to keep track of a person's first, middle, and last initials. Because an initial is obviously a character, it would be prudent to declare three character variables to hold the three initials. In C, you could do that with the following statement:

```
1. main()
   {
   char first, middle, last;
   // Rest of program follows
   }
```

```
2. main()
{
        char first;
        char middle;
        char last;
        // Rest of program follows
}
```

Initialization of Variables

When a variable is declared, it contains undefined value commonly known as garbage value. If we want we can assign some initial value to the variables during the declaration itself. This is called *initialization of the variable.*

Eg-

```
int pageno=10;
char grade='A';
float salary= 20000.50;
```

Expressions

An expression consists of a combination of operators, operands, variables & function calls. An expression can be arithmetic, logical or relational. Here are some expressions:

a+b – arithmetic operation

a>b- relational operation

a == b - logical operation

func (a,b) – function call

4+21

a(b + c/d)/20*

*q = 5*2 x =*

++q % 3

q > 3

As you can see, the operands can be constants, variables, or combinations of the two. Some expressions are combinations of smaller expressions, called subexpressions. For example, c/d is a subexpression of the sixth example.

An important property of C is that every C expression has a value. To find the value, you perform the operations in the order dictated by operator precedence.

Statements

Statements are the primary building blocks of a program. A program is a series of statements with some necessary punctuation. A statement is a complete instruction to the computer. In C, statements are indicated by a semicolon at the end. Therefore

legs = 4

is just an expression (which could be part of a larger expression), but

legs = 4;

is a statement.

What makes a complete instruction? First, C considers any expression to be a statement if you append a semicolon. (These are called expression statements.) Therefore, C won't object to lines such as the following:

```
8;
3 + 4;
```

However, these statements do nothing for your program and can't really be considered sensible statements. More typically, statements change values and call functions:

```
x = 25;

++x;

y = sqrt(x);
```

Although a statement (or, at least, a sensible statement) is a complete instruction, not all complete instructions are statements. Consider the following statement:

```
x = 6 + (y = 5);
```

In it, the subexpression y = 5 is a complete instruction, but it is only part of the statement. Because a complete instruction is not necessarily a statement, a semicolon is needed to identify instructions that truly are statements.

Compound Statements (Blocks)

A compound statement is two or more statements grouped together by enclosing them in braces; it is also called a block. The following while statement contains an example:

```
while (years < 100)
 {
wisdom = wisdom * 1.05;
printf("%d %d\n", years, wisdom);
 years = years + 1;
}
```

If any variable is declared inside the block then it can be declared only at the beginning of the block. The variables that are declared inside a block can be used only within the block.

Identifier

In the programming language C, an identifier is a combination of alphanumeric characters, the first being a letter of the alphabet or an underline, and the remaining being any letter of the alphabet, any numeric digit, or the underline.

Two rules must be kept in mind when naming identifiers.

1. The case of alphabetic characters is significant. Using "INDEX" for a variable is not the same as using "index" and neither of them is the same as using "InDeX" for a variable. All three refer to different variables.
2. As C is defined, up to 32 significant characters can be used and will be considered significant by most compilers. If more than 32 are used, they will be ignored by the compiler.

Keywords

Keywords are the words whose meaning has already been explained to the C compiler. The keywords cannot be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer.

There are only 32 keywords available in C. Below figure gives a list of these keywords for your ready reference.

Data Type

In the C programming language, data types refer to a domain of allowed values & the operations that can be performed on those values. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted. There are 4 fundamental data types in C, which are- *char, int, float &, double. Char* is used to store any single character; *int* is used to store any integer value, *float* is used to store any single precision floating point number & *double* is used to store any double precision floating point number. We can use 2 qualifiers with these basic types to get more types.

There are 2 types of qualifiers-

Sign qualifier- signed & unsigned
Size qualifier- short & long

The data types in C can be classified as follows:

Type	Storage size	Value range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
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	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

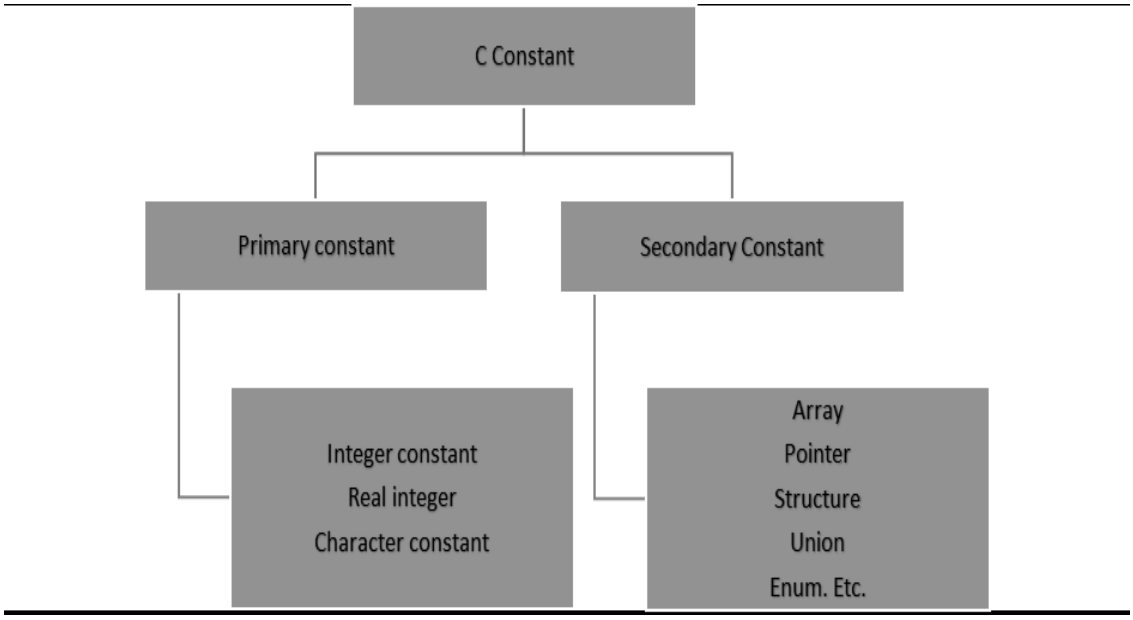
Type	Storage size	Value range	Precision
float	4 bytes	1.2E-38 to 3.4E+38	6 decimal places
double	8 bytes	2.3E-308 to 1.7E+308	15 decimal places
long double	10 bytes	3.4E-4932 to 1.1E+4932	19 decimal places

Constants

A constant is an entity that doesn't change whereas a variable is an entity that may change.

C constants can be divided into two major categories:

- Primary Constants
- Secondary Constants



Here our only focus is on primary constant. For constructing these different types of constants certain rules have been laid down.

Rules for Constructing Integer Constants:

An integer constant must have at least one digit.

a) It must not have a decimal point.
b) It can be either positive or negative.
c) If no sign precedes an integer constant it is assumed to be positive.
d) No commas or blanks are allowed within an integer constant.
e) The allowable range for integer constants is -32768to 32767.

Ex.: 426, +782,-8000, -7605

Rules for Constructing Real Constants:

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Rules for constructing real constants expressed in fractional form:

a) A real constant must have at least one digit.
b) It must have a decimal point.
c) It could be either positive or negative.
d) Default sign is positive.
e) No commas or blanks are allowed within a real constant.

Ex. +325.34, 426.0, -32.76, -48.5792

Rules for constructing real constants expressed in exponential form:

a) The mantissa part and the exponential part should be separated by a letter e.
b) The mantissa part may have a positive or negative sign.
c) Default sign of mantissa part is positive.
d) The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.

e) Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex. +3.2e-5, 4.1e8, -0.2e+3, -3.2e-5

Rules for Constructing Character Constants:

a) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

b) The maximum length of a character constant can be 1 character.

Ex.: ‘M’, ‘6’, ‘+’