C Programming Language

History

C came into being in the years 1969-1973, in parallel with the early development of the UNIX operating system; the most creative period occurred during 1972. Another spate of changes peaked between 1977 and 1979, when portability of the UNIX system was being demonstrated. Finally, in the middle 1980s, the language was officially standardized by the ANSI, which made further changes. Until the early 1980s, although compilers existed for a variety of machine architectures and operating systems, the language was almost exclusively associated with UNIX; more recently, its use has spread much more widely, and today it is among the languages most commonly used throughout the computer industry.

The C Character Set

A character denotes any alphabet, digit or special symbol used to represent information. The following table lists the valid alphabets, numbers and special symbols allowed in C

Alphabets	A, B, …….. Y,Z a, b, …….. y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special Symbols	~ ` ! @ # $ % ^ & * ( ) _ - + = | { } [ ] : ; ” ’ < > . , ? /

Constants, Variables and Keywords

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords. A *constant* is a quantity that doesn’t change during the execution of a program. The quantity can be stored at a location in the memory of the computer. A *variable* can be thought of a name given to a memory location where the data is stored. The contents of a variable can change in the program. Consider the equation

2X + 5Y = 45

Since 2, 5, 45 cannot change, they are called constants, whereas X & Y can take any value, they are called variables. Different types of constants and variables are:

Integer constant – that can represent an integral value.
Real constant – that represents a fractional value.
Character constant – stores characters (alphabets).

For constructing these different constants following rules should be followed:

Constructing Integer Constants:

1. An integer constant must have at least one digit.
2. It must not have a decimal digit.
3. It could be either positive or negative.
4. If no sign is written before the number, it means the value is positive.
5. No commas and blanks are allowed within a number.
6. Valid integer constants are: 4 -23 +98

Constructing Real Constants:

1. A real constant must have at least one digit.
2. It must have a decimal digit.
3. It could be either positive or negative.
4. If no sign is written before the number, it means the value is positive.
5. No commas and blanks are allowed within a number.

6. Valid real constants are: +7.85 11.0 -43.09

Constructing Character constants:

1. A character constant is a single alphabet, a single digit or a single special symbol enclosed within a single inverted commas (pointing towards left).
2. Maximum length of a character constant can be 1.
3. Valid character constant are: 'A' '3' '+'

The quantity that can vary during program execution is called a variable. Variables are the names given to memory locations that hold data. These data can be integer, real or character data. It is important to note that an integer variable can only hold integer data.

A real variable can only hold real data. Rules for declaring variables are:

Constructing Variables

1. A variable name is any combination of 1 to 8 alphabets, digits or an underscore _
2. The first character in the name must be an alphabet and not a number.
3. No commas and blank spaces are allowed within a variable name.
4. No special symbols other than an underscore can be used in variable name.
5. Valid variable names are: num2 sum _tempdir in_time out_time

It is always advisable to construct meaningful variable names like **num**, **sum**, **avg** rather than a, b, c etc.

C Keywords

Keywords are the words whose meaning has already been defined by the C compiler for its use. These words cannot be used as variable names. These words are also known as *Reserved Words*.

The 32 C key words are;

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*

Some C compilers offer additional key words specific to the hardware environment that they operate on. You should be aware of your own C compilers additional key words. Most notably are;

near *far* *huge*

C Instructions

The constants and variables and special symbols are combined to form instructions. There are four types of instructions:

1. Type Declaration Instruction.
   To declare the type of variable to be used in C program.

2. Input/Output Instruction.
   To perform the function of supplying input data to a program and obtaining the output results from it.

3. Arithmetic Instruction.
   To perform arithmetic operation between constants and variables.

4. Control Instruction.
   To control the sequence of execution of various statements in a C program.

Type Declaration Instruction

Any variable used in a C program must be declared. The type declaration statements are written at the beginning of the C program. Examples are

```
int number;     declares an integer with name 'number'
float average;  declares a real number with name 'average'
char choice;    declares a character variable with name 'choice'
```

Arithmetic Instruction

A C arithmetic instruction consists of a variable name on the left side = and variable names, constants on the right side. Various arithmetic operators (+, - . *, /) connect them.

```
int  count;
float sum,average;
count = 7;
sum = 312;
average = sum * 2.5  / count;
```

where = is assignment operator.
It is important to understand how the execution of arithmetic instruction takes place. First the right hand side is evaluated and the numerical value thus obtained is stored in the variable on the left hand side. In the above example, the numerical values of **sum** and **count** are used alongwith the constant **2.5**. The value thus obtained (111.428574) is stored in **average**.

Input/Output and Control instructions will be discussed later.

Comments

Comments are program lines that are ignored by the compiler. They are only used to provide some information about what the program (or part of program) does. The writing of comments should be encouraged as it makes the program more readable. Following rules must be observed for writing comments.

1. Comments are enclosed in /* and */
2. Any number of comments can be written in a program.
3. Comments cannot be nested. E.g. /* this is /* nested */ comment */
4. A comment can split on more than one line.

Role of Library functions

C programs are compiled and combined with library functions provided with the C compiler. These libraries are of generally standard functions, the functionality of which are defined in the ANSI standard of the C language, but are provided by the individual C

compiler manufacturers to be machine dependant. Thus, the standard library function "printf()" provides the same facilities on a DEC VAX as on an IBM PC, although the actual machine language code in the library is quite different for each. The C programmer however, does not need to know about the internals of the libraries, only that each library function will behave in the same way on any computer.

C program structure

The general form of a C program is as follows; (line numbers # are for reference only, and are not to be used in the C programs)

```
1.      compiler preprocessor statements
2.      global data declarations

3.      return-type main(parameter list)
4.      {
5.      statements
6.      }

7.      return-type f1(parameter list)
8.      {
9.      statements
10.     }

11.     return-type f2(parameter list)
12.     {
13.     statements
14.     }
15.     .
16.     .
17.     .
18.     return-type fn(parameter list)
19.     {
20.     statements
21.     }
```

#1 *Compiler preprocessor* statements are directives issued to compiler to specify control settings for the process of compilation.
#2 *global data declarations* are variables that are visible to all modules of the program.
#3 *return-type main(parameter list)* is the declaration of main() function. main() is the entry point of the program. All C programs must have a function named main(). The return-type if not specified is assumed to be integer and the parameter list is generally kept empty in simple programs.
f1(), f2() etc are functions with their return types.
Statements are the C statements.
{ marks the beginning of a block of C program. (this block can be of a function, or a block inside a function)
} marks the end of blocks

Data Types

There are four basic types of data in the C language; character, integer, floating point, and valueless that are referred to by the C key words; "char", "int", "float" and "void" respectively.

To the basic data types may be added the type modifiers; signed, unsigned, long and short to produce further data types. By default data types are assumed signed. The size of each data type varies from one hardware platform to another, but the least range of values that can be held is described in the ANSI standard as follows;

Type	**Size**	**Range**
char	8	-127 to 127
unsigned char	8	0 to 255
int	16	-32767 to 32767
unsigned int	16	0 to 65535
long int	32	-2147483647 to 2147483647
unsigned long int	32	0 to 4294967295
float	32	Six digit precision
double	64	Ten digit precision
long double	80	Ten digit precision

In practice, this means that the data type `char' is particularly suitable for storing flag type variables, such as status codes, which have a limited range of values. The `int' data type can be used, but if the range of values does not exceed 127 (or 255 for an unsigned char), then each declared variable would be wasting storage space.

Which real number data type to use: `float', `double' or `long double' is another tricky question. When numeric accuracy is required, for example in an accounting application, the instinct would be to use the `long double', but this requires at least 10 bytes of storage space for each variable. And real numbers are not as precise as integers anyway, so perhaps one should use integer data types instead and work around the problem. The data type `float' is worse than useless since its six digit precision is too inaccurate to be relied upon. Generally, then, you should use integer data types where ever possible, but if real numbers are required use at least a `double'.

Declaring a variable

All variables in a C program must be declared before they can be used. The general form of a variable definition is;

```
type name;
```

So, for example to declare a variable "x", of data type "int" so that it may store a value in the range -32767 to 32767, you use the statement;

```
int x;
```

Input/Output Instructions

These statements allow data movement to & from the program. Output statements are those statements that allow data to be written on some output device (screen, printer or a file). Input statements on the other hand allow data to be given into the program through input devices (like keyboard or file). The basic I/O statements are used to read input from keyboard (standard input device) and provide output on screen (standard output device).

printf() – to print output on screen.

printf() is a function which is used to print on the screen. The general form of **printf** is

printf("<format string>",<list_of_variables>);
where <format string> can be a
simple string to print some English words.
%d for printing integer values.
%f for printing real values.
%c for printing character values.

Following are some examples of using **printf** statement

```
printf("%f",interest);
printf("%f",10.22);
printf("Simple interest is Rs. %f",interest);
printf("Count = %d",count);
printf("%f saved for %d months gives Interest Rs. %f", principle, time, interest);
```

The **printf** statement doesn't write the output automatically into a new line. Incase it is desired that the output should start from a new line, extra information must be provided into **printf** <format string>.

printf("\n %f",interest); **\n** is called a **newline** and it takes the cursor to the next line. It can be placed at any place in the <format string> and not just in the beginning.

scanf() – reading input from keyboard

In order to supply values to variables from keyboard during execution of the program, a statement called **scanf**() is used. **scanf**() accepts value from keyboard and assigns it to the variables specified. The general form of **scanf**() is:

scanf("<format string>",<&var1, &var2, … >);
where <format string> can be a
%d for reading integer value.
%f for reading real value.
%c for reading character value.
The variables are written alongwith an ampersand (&), it must be ensured that & is placed before the variable name, otherwise **scanf**() will not assign the value to the variable.

```
scanf("%d", &count);
scanf("%f", &amount);
scanf("%d %f", &count,&amount);
```

while entering the numbers, the numbers must be separated by a TAB, ENTER or SPACE.

Control Instructions

As the name suggests, the control instructions enable us to specify the order in which various instructions in a program are to be executed by the computer. In simple words these statements may alter the sequence of program execution. There are four types of control instructions in C. They are:

1. Sequence control instruction.
   Ensure that instructions are executed in same order as they appear in the program.
2. Selection or decision control instruction.
   Allow the program to take a decision (based on a condition) to decide which instruction is to be executed next.
3. Repetition or loop control instruction.
   Allow the computer to execute a group of statements repeatedly.
4. Case control instruction.
   Allow the control to branch to one of a list of possible statements available.

Sample programs

```
/* Program I
   this program displays Hello on the screen */

main()
{
        printf("Hello");
}


/* End of Program I*/

/* Program II
   this program reads two integers and displays the sum. The sum  is displayed in three
different ways          */

main()
{
        int num1, num2;
        int sum;
        clrscr(); /* clear the screen at startup */
        printf("\nEnter two numbers");/*tell user what data to enter*/
        scanf("%d %d",&num1, &num2);
        sum = num1+num2;
        printf("\n %d %d %d",num1, num2, sum);
        printf("\nSum of %d and %d = %d ",num1, num2, sum);
        printf("\n %d + %d = %d",num1, num2, sum);
        getch(); /*just wait for a keypress */
}
```