Lecture Notes

C Fundamentals

In this module, you will be introduced to the basics and fundamentals of the C programming language. You will gain a proper understanding of all the basics of the C language and will be able to use them to write simple C programs.

In this module

The following topics will be covered:

- C instructions
    - Getting started with C
    - Literals and identifiers in C
    - Input/Output functions in C
    - Types of instructions in C
- Decision Control Structure
    - if statement
    - if...else statement
    - if...else if...else statement
    - Use of logical operators
    - Use of conditional operators
- Loop Control Structure
    - while loop
    - for loop
    - do-while loop
    - Break and continue statements
    - Increment and decrement operators
    - Compound assignment operators
    - Infinite loop
    - Running sums and products

Basics of C Language

Before delving deep into the basics of C, let's first understand why you are required to learn the C language. The following points describe the importance of the C programming language:

- It is simple and easy to learn.
- C programs run extremely fast, i.e., they have an extremely low execution time.
- All the operating systems are developed in the C language.
- It is extensively used in embedded systems.
- It is used in all the device drivers.
- C is a foundation to learn various other programming languages.

'Hello World' Program:

You used Visual Studio IDE to write all the C programs in this course. IDE (Integrated Development Environment) is a software that is used by software developers to build applications.

The 'Hello World' program is the first C program that you wrote. The program is as follows:

```
#include <stdio.h>

main()

{

 printf ("Hello World");

}
```

The code dissection is as follows:

- stdio.h is the header file that has the function printf() that is used later in the program.
- main() is a function.
- printf() is a function that we are using to print the message "Hello World".
- ';' after the printf() statement is called a statement terminator.

Literals and Identifiers:

The classical method of learning any human-understandable language is to first learn the alphabets used in the language and then learn how to combine these alphabets to form words, which, in turn, are combined to form sentences; these sentences are then combined to form paragraphs. Similarly, in order to learn computer-understandable languages, you should not directly jump to writing programs. You must first know the alphabets, numbers and special symbols that are used in the language and then learn how to use them to construct constants, variables and keywords and, finally, learn how these are combined to form an instruction. Several instructions would then be combined to form a program. Hence, you need to begin learning the C language starting with literals and identifiers.

Literals are also known as constants, and identifiers are also known as variables. Constants, as the name suggests, are constant values; for example 3, 5.5, and so on. The value of the variables, on the other hand, can vary.

Constants (Literals):

Constants in C can be of the following two types:

- Primary constants
- Secondary constants

Integer constants, real constants and character constants are primary constants. Whereas, pointers, arrays, strings, structures, unions, and so on, fall under the category of secondary constants.

Integer Constants:

- Integer constants can be positive, negative or zero. If the sign is not mentioned, then they are treated as positive integers.
  For example: +35   -23   0   22
- They do not contain a decimal point.
  For example, 35 is a valid integer constant. Whereas, 35.0 is not a valid integer constant.
- They should not contain any commas or spaces.
  For example, 32500 is a valid integer constant, whereas 32,500 is not.
  Similarly, 32 500 is also not a valid integer constant.
- They can range from -2147483648 to +2147483647.

Real Constants:

- Real constants must contain a decimal point.
  For example: 8.5
- They can be positive or negative.
  For example: +12.22 -99.99
- They should not contain any commas or spaces.
  For example, 12,500.88 is an invalid real constant and so is 12 500.88.
- They can range from $-3.4\text{x}10^{38}$ to $3.4\text{x}10^{38}$.

Character Constants:

- They can be any single character enclosed within ’ ’.
  For example: ’z’ ’A’ ’-’
  Note that inverted quotes should not be used. That is, both the quotes should slant to the left.

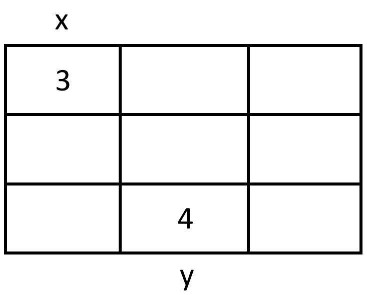
Variables (Identifiers):

A particular type of constant can only be stored in the same type of variable. Variables are divided into the following two types:

- Primary
- Secondary

Integer variables, real variables and character variables are primary constants. Whereas, variables for pointers, arrays, strings, structures, unions, and so on, fall under the category of secondary variables.

Consider the following two statements:
x = 3 and y = 4
They are stored in the memory as follows:



The value 3 is stored at some location in the memory, and this memory location is given the name x. Similarly, the value 4 is stored at some other location in the memory, and this memory location is given the name y.

So, a variable can be viewed as:

- An entity whose value can be varied, and
- A name for a location in the memory.

Declaring Variables:

- For declaring an integer type variable, the keyword ‘int’ can be used.
  For example, to store a constant integer value +3 using an integer variable ‘a’, the following steps can be used:
  int a
  a=3
- For declaring a real type variable, the keyword ‘float’ can be used.
  For example, to store a constant real value +10.22 using an integer variable ‘b’, the following steps can be used:
  float b
  b=10.22
- For declaring a character type variable, the keyword ‘char’ can be used.
  For example, to store a constant character value ‘M’ using a character variable ‘c’, the following steps can be used:
  char c
  c = ’M’

Rules for Variable Names:

- Every variable name should start with an alphabet, that is, the first character of any variable name must be an alphabet or underscore.
- Every other character, except for the first character, can be an alphabet or a number or underscore.
  For example, abc123, _123abc and A_a99 are valid variable names.
- The length of the variable name should be eight characters or less.
- There should not be any commas or spaces.
- They are case-sensitive.
  For example, ABC and Abc are treated as two different variables.

Standard Practices for Variable Names and Indentations:

- Though it is valid to give any name, adhering to the rules, to a variable, it is not a standard practice to do so.
  Example:
  Consider a programmer needs to create a variable to store the price of a book.
  1. float p;
  2. float BookPrice;

  Though both the variable names mentioned above are valid. Variable name ‘BookPrice’ is better to use, as it provides a better understanding of what the variable stores.
- The following cases are the standard naming conventions for variable names.
- Camel case: The first letter of the variable name is in lower case, the first character of all other words must be capitalized.
  Example: bookPrice, bookPages, studentAge.
- Pascal case: The first letter of every word in the variable name is capitalized.
  Example: BookPrice, BookPages, StudentAge.
- Snake case: All the characters in the variable name are in lower case and words are separated by an underscore.
  Example: book_price, book_pages, student_age.

Comments in C:

- Comments can be used in programs to make them more human-readable. These comments are not compiled or executed by the machine. ‘//’ can be used to write a single-line comment.
  Example:
  ```
  //adding a and b
  #include<stdio.h>
  int main()
  {
          int a,b,c;
          a = 5;
          b = 4;
          c = a + b;
  }
  ```
  As you can observe in the code given above, the comment ‘adding a and b’ provides the readers with more information about the program.
- ‘/*. .. */’ can be used to write multi-line comments.

Example:

```
/* This is a
Multi line comment*/
```

- We can have any number of comments in a program.
- Note that a C program does not allow nesting of comments, that is, '/*.../*. ..*/.…*/' is not valid in C.

printf() and scanf():

- printf() is an output function. As the name suggests, it is used to print out the output and display it on the screen.
  The general form of the printf() statement is 'printf("format string",list of variables);'.
  For printf(), the list of variables is optional, and the format string can contain format specifiers, escape sequences and any other characters.
- scanf() is an input function. As the name suggests, it is used to scan the values entered using a keyboard.
  The general form of the scanf() statement is 'scanf("format string",list of variables);'.
  For scanf(), the list of variables is compulsory, and the format string can only contain format specifiers.
- To print or scan variables, format specifiers need to be used. A few format specifiers are as follows:
  '%d' for 'int'
  '%f' for 'float'
  '%c' for 'char'

  Example:

```
int a;
a = 5;
printf("%d", a);
char c;
scanf("%c",&c);
printf("%d %c", a,c);
```

- '&' is the 'address of' operator, and to scan a value into a variable, the 'address of' operator needs to be used.

- Also, the sequence of format specifiers must match with the corresponding sequence of variables. That is, as seen in the second printf() statement, %d corresponds to integer data type variable ‘a’, and %c corresponds to character data type variable ‘c’.
- ‘\t’ and ‘\n’ are a few escape sequences. ‘\t’ acts as a tab, that is, the cursor jumps to the right. ‘\n’ is a new line command, that is, the cursor jumps to the next line.

To understand all these concepts, let’s take a look at the following example:

Example:

printf(“\n simple interest = Rs. %f”, si);

Here, ‘\n’ is the escape sequence and ‘%f’ is the format specifier corresponding to the variable ‘si’. ‘simple interest = Rs.’ are some other characters.

Scope of a Variable:

The scope of a variable refers to all the places where the variable is accessible. In contrast, the scope of a default variable is limited to the block in which it has been declared.

Example:

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

This program throws an error because ‘a’ was declared inside main(), and printf() is trying to access the variable from outside that block.

More about ‘scope’ will be taught in the later modules.

C Instructons

Instructions in C can be divided into the following three types:

1. Type declaration instructions
2. Arithmetic instructions
3. Control instructions

Type Declaration Instructions:

- As the name suggests, type declaration instructions are used to declare the data type of variables.

For example: int a;   char c;   float f;

- Multiple variables of the same data type can be declared in the same statement.
  For example: int a,b,c,d;
- A variable can also be declared and initiated in the same statement.
  Example 1:
  int a;
  a=5;
  The two statements given above can be combined and written in the form of a single statement as given below:
  int a = 5;
  Example 2:
  int a = 2, b = 3, c = a + b;
- Trying to initialise or use a variable without declaring it will throw an error.
  Example:
  int a=b=c=5;
  This is an invalid statement. Here, we are trying to assign the value of 'b' even before declaring 'b', so this will throw an error.

Arithmetic Instructions:

- '+', '-', '*', '/' and '%' are the arithmetic operators in C, using which addition, subtraction, multiplication, division and modulus can be performed, respectively. These operators can be used in C programs to manipulate the values of variables by performing arithmetic operations.
  For example:
  c = a + b;
  Here, '=' is known as an assignment operator, and the variables 'a' and 'b' are known as operands.
- To perform exponential operations, we can make use of the pow() function.
  Example:
  int a = pow(5,2);
  In this statement, 'a' gets equated to 5 to the power of 2, which is 25.
- The left-hand side of an assignment operator (=) should always be a variable.
- No operator is assumed to be present.
  For example, a(b*c); is an invalid expression in C; the proper expression is a*(b*c);.
- Arithmetic instructions (AI) can be of the following three types:
  - Integer mode arithmetic instructions: Here, all the operands are integers.
    Example: int a = 3+4;

    - Real mode arithmetic instructions: Here, all the operands are real values.
      Example: float b = 3.5/2.5
    - Mixed mode arithmetic instructions: Here, the operands can be both integers and real values.
      Example: int c = 2/5.5
- It is fairly obvious that an integer added to another integer results in an integer. But, when it comes to mixed mode AI, we are dealing with both integers and real values. The following table explains the data type of the result in the case of mixed mode AI.

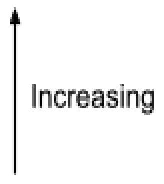
Operand 1	**Operand 2**	**Result**
int	int	int
int	float	float
float	int	float
float	float	float

To understand these concepts better, let's take a look at a few more examples given below.
Example 1:
int a;
a = 5/2;
Here, 5 is an integer, and 2 is also an integer; hence, their division results in an integer, which is 2. So, 'a' gets assigned the value 2.
Example 2:
int a;
a = 5.0/2.0;
Here, 5.0 and 2.0 are both real values; hence, their division results in a real value, which is 2.5. But, since 'a' is declared as 'int' data type, it gets assigned the value 2.

Precedence and Associativity:

- When there are multiple operators in a single expression, the order in which these operations are performed depends on the precedence and associativity of the operators.
- Precedence is also known as priority or hierarchy.
- The precedence of the operators is as follows:

()
* / %
+ -
=



Example: a = 3 + 5 * 2;
Here, since the precedence of ‘*’ is higher than that of ‘+’, ‘5*2’ is evaluated first, and the result 10 is added to 3, so the expression on the right-hand side of ‘=’ evaluates to 13. This is finally assigned to ‘a’ since the precedence of ‘=’ is the least.

- In case we want any particular operator to act first, we can enclose that expression within parentheses.
  Example: a = (8 - 5)*2
  Here, since the expression ‘8 - 5’ is enclosed within parentheses, it takes priority and is evaluated, and this results in -3; this value is then multiplied by 2 where the result is -6, which is then assigned to ‘a’.
- The operators ‘*’, ‘/’ and ‘%’ have left-to-right associativity. So, the operator that appears on the left goes to action before the operator on the right.
  Example:
  int a= 5*3%10;
  Here, ‘5*3’ is evaluated initially, and the modulus is performed on the resulting value 15, so the complete expression evaluates to 1. Finally, ‘a’ gets assigned the value 1.
- The ‘=’ operator has right-to-left associativity. So, the compiler evaluates the expression from right to left.

Control Instructions:

- Control instructions are the instructions that control the flow of the program.
  There are four types of control instructions as mentioned below.
  1. Sequence control instruction
  2. Decision control instruction
  3. Repetition/loop control instruction
  4. Case control instruction
- Sequence control instruction is the default control instruction, that is, when there are no other control instructions, the flow of all the instructions in the program is sequential (line by line).