C-Language

Learning Material

Session-I

C is a general-purpose, high-level language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.The UNIX operating system, the C compiler, and essentially all UNIX applications programs have been written in C. The C has now become a widely used professional language for various reasons.

- Easy to learn

- Structured language

- It produces efficient programs.

- It can handle low-level activities.

- It can be compiled on a variety of computer platforms.

Facts about C

- C was invented to write an operating system called UNIX.

- C is a successor of B language which was introduced around 1970

- The language was formalized in 1988 by the American National Standard Institute (ANSI).

- The UNIX OS was totally written in C by 1973.

- Today C is the most widely used and popular System Programming Language.

- Most of the state-of-the-art softwares have been implemented using C.

- Today's most popular Linux OS and RBDMS MySQL have been written in C.

Why to use C?

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems

- Language Compilers

- Assemblers

- Text Editors

- Print Spoolers

- Network Drivers

- Modern Programs

- Databases

- Language Interpreters

- Utilities

C Programs

A C program can vary from 3 lines to millions of lines and it should be written into one or more text files with extension **".c"**; for example, *hello.c*. You can use **"vi"**, **"vim"** or any other text editor to write your C program into a file.

This tutorial assumes that you know how to edit a text file and how to write source code inside a program file.

Session-II

Before we study basic building blocks of the C programming language, let us look a bare minimum C program structure so that we can take it as a reference in upcoming chapters.

C Hello World Example

A C program basically consists of the following parts:

- Preprocessor Commands

- Functions

- Variables

- Statements & Expressions

- Comments

  Let us look at a simple code that would print the words "Hello World":

```
#include <stdio.h>


int main()

{

  /* my first program in C */

  printf("Hello, World! \n");


  return 0;

}
```

Let us look various parts of the above program:

1. The first line of the program *#includes <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
2. The next line *int main()* is the main function where program execution begins.
3. The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
4. The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
5. The next line **return 0;** terminates main()function and returns the value 0.

Compile & Execute C Program:

Lets look at how to save the source code in a file, and how to compile and run it. Following are the simple steps:

1. Open a text editor and add the above-mentioned code.
2. Save the file as *hello*
3. Open a command prompt and go to the directory where you saved the file.
4. Type *gcc hello.c* and press enter to compile your code.
5. If there are no errors in your code the command prompt will take you to the next line and would generate *a.out* executable file.
6. Now, type *a.out* to execute your program.
7. You will be able to see *"Hello World"* printed on the screen

You have seen a basic structure of C program, so it will be easy to understand other basic building blocks of the C programming language.

Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminates with the characters */ as shown below:

```
/* my first program in C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, $, and % within identifiers. C is a **case sensitive** programming language. Thus, *Manpower* and *manpower* are two different identifiers in C. Here are some examples of acceptable identifiers:

```
mohd     zara   abc  move_name  a_123
myname50  _temp  j   a23b9     retVal
```

Keywords

The following list shows the reserved words in C. These reserved words may not be used as constant or variable or any other identifier names.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned

const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Whitespace in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

```
int age;
```

There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish them. On the other hand, in the following statement:

```
fruit = apples + oranges;   // get the total fruit
```

No whitespace characters are necessary between fruit and =, or between = and apples, although you are free to include some if you wish for readability purpose.

In the C programming language, data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The array types and structure types are referred to collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see basic types in the following section, whereas, other types will be covered in the upcoming chapters.

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	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator. The expressions *sizeof(type)* yields the storage size of the object or type in bytes. Following is an example to get the size of int type on any machine:

```
#include <stdio.h>

#include <limits.h>


int main()

{

    printf("Storage size for int : %d \n", sizeof(int));


    return 0;

}
```

When you compile and execute the above program it produces the following result on Linux:

Storage size for int : 4

Floating-Point Types

Following table gives you details about 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

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. Following example will print storage space taken by a float type and its range values:

```c
#include <stdio.h>

#include <float.h>


int main()

{

  printf("Storage size for float : %d \n", sizeof(float));

  printf("Minimum float positive value: %E\n", FLT_MIN );

  printf("Maximum float positive value: %E\n", FLT_MAX );

  printf("Precision value: %d\n", FLT_DIG );


  return 0;

}
```

When you compile and execute the above program, it produces the following result on Linux:

Storage size for float : 4

Minimum float positive value: 1.175494E-38

Maximum float positive value: 3.402823E+38

Precision value: 6

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive. Based on the basic types explained in previous chapter, there will be the following basic variable types:

Type	Description
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

Variable Definition in C:

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows:

```
type variable_list;
```

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool or any user-defined object, etc., and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

```
int    i, j, k;
char   c, ch;
float  f, salary;
```

```
double d;
```

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

```
type variable_name = value;
```

Some examples are:

```
extern int d = 3, f = 5;   // declaration of d and f.
int d = 3, f = 5;          // definition and initializing d and f.
byte z = 22;               // definition and initializes z.
char x = 'x';              // the variable x has the value 'x'.
```

For definition without an initialize: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

Variable Declaration in C:

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you can declare a variable multiple times in your C program but it can be defined only once in a file, a function or a block of code.

Session-III

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators:

- Arithmetic Operators

- Relational Operators

- Logical Operators

- Assignment Operators

- Misc Operators

This tutorial will explain the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

Arithmetic Operators

Following table shows all the arithmetic operators supported by C language. Assume variable **A** holds 10 and variable **B** holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increments operator increases integer value by one	A++ will give 11
--	Decrements operator decreases integer value by one	A-- will give 9

Relational Operators

Following table shows all the relational operators supported by C language. Assume variable **A** holds 1

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A!= B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operators

There are following assignment operators supported by C language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right	C -= A is

	operand from the left operand and assign the result to left operand	equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Misc Operators ↦ sizeof & ternary

There are few other important operators including **sizeof** and? **:** supported by C Language.

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of an variable.	&a; will give actual address of the variable.
*	Pointer to a variable.	*a; will pointer to a variable.
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator.

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & size of	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left

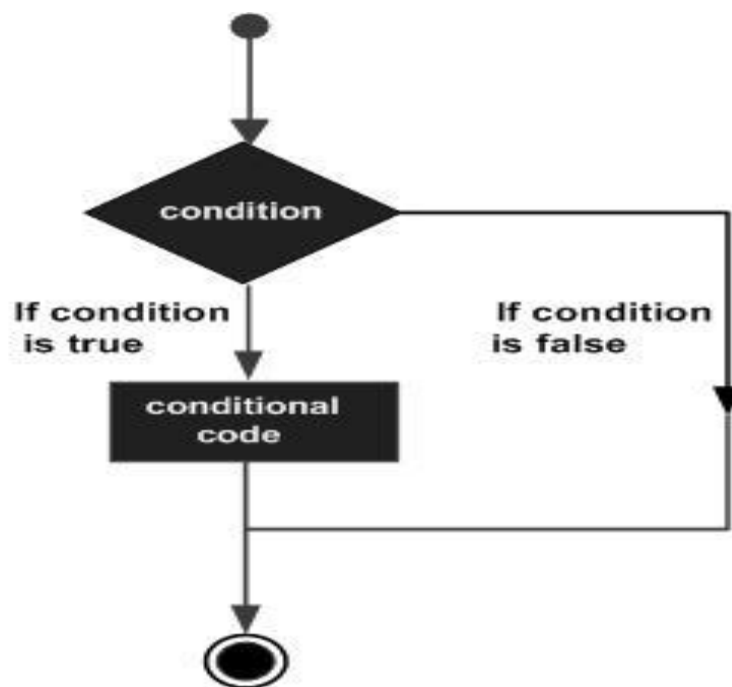| Comma	,		Left to right

Session-IV

Decision Making

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:

C programming language assumes any **non-zero** and **non-null** values as **true**, and if it is either **zero** or **null**, then it is assumed as **false** value.

C programming language provides following types of decision making statements. Click the following links to check their detail.

Statement	Description
if statement	An **if statement** consists of a boolean expression followed

	by one or more statements.
if...else statement	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expression is false.
nested if statements	You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
switch statement	A **switch** statement allows a variable to be tested for equality against a list of values.

The ? : Operator:

We have covered **conditional operator ? :** in previous chapter which can be used to replace **if...else**statements. It has the following general form:

Exp1 ? Exp2 : Exp3;

Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

The value of a ? expression is determined like this: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.
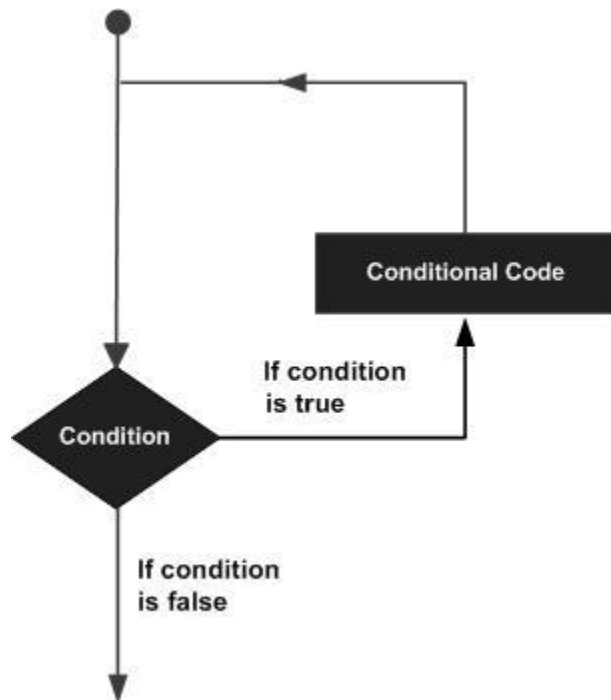
Session-V

C - Loops

There may be a situation, when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:

C programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type	Description

while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	You can use one or more loop inside any another while, for or do..while loop.

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C supports the following control statements. Click the following links to check their detail.

Control Statement	Description
break statement	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
goto statement	Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

The Infinite Loop:

A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

```c
#include <stdio.h>

int main ()
{

  for( ; ; )
  {
    printf("This loop will run forever.\n");
  }

  return 0;
}
```