E- content PIC

The C programming language 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 using any programming language.

C Environment Setup

This section describes how to set up your system environment before you start doing your programming using C language.

Before you start doing programming using C programming language, you need the following two softwares available on your computer, (a) Text Editor and (b) The C Compiler.

Text Editor

This will be used to type your program. Examples of few editors include Windows Notepad, OS Edit command, Brief, Epsilon, EMACS, and vim or vi.

Name and version of text editor can vary on different operating systems. For example, Notepad will be used on Windows, and vim or vi can be used on windows as well as Linux or UNIX.

The files you create with your editor are called source files and contain program source code. The source files for C programs are typically named with the extension ".c".

Before starting your programming, make sure you have one text editor in place and you have enough experience to write a computer program, save it in a file, compile it and finally execute it.

The C Compiler

The source code written in source file is the human readable source for your program. It needs to be "compiled", to turn into machine language so that your CPU can actually execute the program as per instructions given.

This C programming language compiler will be used to compile your source code into final executable program. I assume you have basic knowledge about a programming language compiler.

Most frequently used and free available compiler is GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have respective Operating Systems.

Following section guides you on how to install GNU C/C++ compiler on various OS. I'm mentioning C/C++ together because GNU gcc compiler works for both C and C++ programming languages.

Installation on UNIX/Linux

If you are using Linux or UNIX, then check whether GCC is installed on your system by entering the following command from the command line:

```
$ gcc -v
```

If you have GNU compiler installed on your machine, then it should print a message something as follows:

```
Using built-in specs.
Target: i386-redhat-linux
Configured with: ../configure --prefix=/usr .......
Thread model: posix
gcc version 4.1.2 20080704 (Red Hat 4.1.2-46)
```

If GCC is not installed, then you will have to install it yourself using the detailed instructions available athttp://gcc.gnu.org/install/

This tutorial has been written based on Linux and all the given examples have been compiled on Cent OS flavor of Linux system.

Installation on Mac OS

If you use Mac OS X, the easiest way to obtain GCC is to download the Xcode development environment from Apple's web site and follow the simple installation instructions. Once you have Xcode setup, you will be able to use GNU compiler for C/C++.

Xcode is currently available at developer.apple.com/technologies/tools/.

Installation on Windows

To install GCC at Windows you need to install MinGW. To install MinGW, go to the MinGW homepage, www.mingw.org, and follow the link to the MinGW download page. Download the latest version of the MinGW installation program, which should be named MinGW-<version>.exe.

While installing MinWG, at a minimum, you must install gcc-core, gcc-g++, binutils, and the MinGW runtime, but you may wish to install more.

Add the bin subdirectory of your MinGW installation to your PATH environment variable, so that you can specify these tools on the command line by their simple names.

When the installation is complete, you will be able to run gcc, g++, ar, ranlib, dlltool, and several other GNU tools from the Windows command line.

C Program Structure

Let's look into Hello World example using C Programming Language.

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 #include **<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

Let's 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.c**

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

```
$ gcc hello.c
$ ./a.out
Hello, World!
```

Make sure that **gcc** compiler is in your path and that you are running it in the directory containing source file hello.c.

C Basic Syntax

This chapter will give details about all the basic syntax about C programming language including tokens, keywords, identifiers, etc.

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.

Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens:

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

The individual tokens are:

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

Semicolons ;

In C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

For example, following are two different statements:

```
printf("Hello, World! \n");
return 0;
```

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
break
case
char
const
continue
default
do
double

C Data Types

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 types in C can be classified as follows:

S.N.	Types and Description
1	**Basic Types:** They are arithmetic types and consists of the two types: (a) integer types and (b) floating-point types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only be assigned certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

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.

Integer Types

Following table gives you details about standard integer types with its storage sizes and value ranges:

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:

```
#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
```

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

S.N.	Types and Description
1	**Function returns as void** There are various functions in C which do not return value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept as a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

The void type may not be understood to you at this point, so let us proceed and we will cover these concepts in the upcoming chapters

C -Variable :

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 initializer: 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.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function:

```
#include <stdio.h>

// Variable definition:
extern int a, b;
extern int c;
extern float f;

int main ()
{
// Variable definition:
int a, b;
int c;
float f;

// actual initialization
  a =10;
  b =20;
  c = a + b;

  printf("value of c : %d \n", c);

  f = 70.0/3.0;
  printf("value of f : %f \n", f);

  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of c : 30

value of f : 23.333334
```

Same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For example:

```
// function declaration
int func();

int main()
{
   // function call
    int i = func();
}
// function definition
int func()
{
   return 0;
}
```

Lvalues and Rvalues in C

There are two kinds of expressions in C:

1. **lvalue:** An expression that is an **lvalue** may appear as either the left-hand or right-hand side of an assignment.

2. **rvalue:** An expression that is an **rvalue** may appear on the right- but not left-hand side of an assignment.

Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and cannot appear on the left-hand side. Following is a valid statement:

```
int g = 20;
```

But following is not a valid statement and would generate compile-time error: