

Forum:[Hack Forums](#)

Thread: [Beginners Guide to C](#)

Wolfi - July 28, 2014, 11:40 am

[img]http://s12.postimg.org/jkuwc32od/C_Beginners_Tut.png[/img]

Note: This thread is been re posted here, and the [previous one](#) is closed since it can't be edited because I didn't reserve any posts. Sorry for that. ?

"Hi guys, being new to HF I was thinking on how to contribute something to HF. Saw a wide variety of sections and articles. When I came across the C coding section, I saw a lot of questions being asked on the Best Tutorials, Best Compilers and so on. That gave me the thought of posting a detailed tutorial for Beginners (only beginners) on C. In this tutorial, I've included just a brief introduction and basic terminologies of C. I must say, a lot of popular sites helped me make this tutorial whose references are provided at the end. All credit goes to the original authors." ?

What Is C ?

C is a general-purpose high-level computer programming language that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. Like most imperative languages in the ALGOL tradition, C has facilities for structured programming and allows lexical variable scope and recursion, while a static type system prevents many unintended operations. Its design provides constructs that map efficiently to typical machine instructions, and therefore it has found lasting use in applications that had formerly been coded in assembly language, most notably system software like the Unix computer operating system.

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. In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C", was completed late 1988.

C is one of the most widely used programming languages of all time, and C compilers are available for the majority of available computer architectures and operating systems. Many later languages have borrowed directly or indirectly from C, including D, Go, Rust, Java, JavaScript, Limbo, LPC, C#, Objective-C, Perl, PHP, Python, Verilog (hardware description language), and Unix's C shell. But C keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers. C has now become a widely used professional language mainly because of the following 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

Why Should I Use C ?

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems, and efficient compilers are available for machines ranging in power from the Apple Macintosh to the Cray supercomputers. The largest measure of C's success seems to be based on purely practical considerations:

The portability of the compiler

The standard library concept

A powerful and varied repertoire of operators

An elegant syntax

Ready access to the hardware when needed

The ease with which applications can be optimised by hand-coding isolated procedures

C is often called a "Middle Level" programming language. This is not a reflection on its lack of programming power but more a reflection on its capability to access the system's low level functions. Most high-level languages (e.g. Fortran) provides everything the programmer might want to do already built into the language. A low level language (e.g. assembler) provides nothing other than access to the machines basic instruction set. A middle level language, such as C, probably doesn't supply all the constructs found in high-languages - but it provides you with all the building blocks that you will need to produce the results you want!

Where Is C Used ?

C was initially used for system development work, in particular the programs that make-up the operating system. Why use C? Mainly 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

Data Bases

Language Interpreters

Utilities

In recent years C has been used as a general-purpose language because of its popularity with programmers. It is not the world's easiest language to learn and you will certainly benefit if you are not learning C as your first programming language! C is trendy (I nearly said sexy) - many well established programmers are switching to C for all sorts of reasons, but mainly because of the portability that writing standard C programs can offer.

Environment Setup

ONLINE ENVIRONMENT

You really do not need to set up your own environment to start learning C programming language. Reason is very simple, we already have a lot of C Programming environments available online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Some of the commonly used ones are:

[Compile Online](#)

[Ideone](#)

[Codepad](#)

You may optionally try the following program using any of the above online compilers.

```
Code #include <stdio.h>
int main()
{
    /* my first program in C */
    printf("Hello, World! \n");

    return 0;
}
```

The most important tip towards learning a language is, nothing but practice.

LOCAL ENVIRONMENT

If you are still willing to set up your environment for C programming language, you need the following two softwares available on your computer:

Text Editor

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:

Quote:\$ gcc -v

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

Quote: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 at <http://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.

Program Structure

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 the upcoming sections.

A C program basically consists of the following parts:

- Preprocessor Commands

- Functions

- Variables

- Statements and Expressions

- Comments

Consider the following example we tried earlier that would print the words "Hello, World! "

```
Code #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:

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.

The next line `int main()` is the main function where program execution begins.

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.

The next line `printf(...)` is another function available in C which causes the message "Hello, World!" to be displayed on the screen.

The next line `return 0;` terminates `main()` function and returns the value 0.

Wolfi - July 28, 2014, 11:41 am

Basic C Terminologies

TOKENS

C tokens are the basic building blocks in C language which are constructed together to write a C program.

Each and every smallest individual units in a C program are known as C tokens.

C tokens are of the following types: Keywords

Identifiers

Variables

Constants

Strings

Special Symbols

Operators

KEYWORDS

Keywords are the reserved words used in programming. Each keyword has fixed meaning and that cannot be changed by user.

For example:

Code `int sum;`

Here, `int` is a keyword that indicates, 'sum' is of type integer.

As, C programming is case sensitive, all keywords must be written in lowercase. Here is the list of all keywords predefined by ASCII C

[auto](#) - used to define a variable of storage class automatic.

[break](#) - used to jump out of the innermost enclosing loop.

[case](#) - allows a variable to be tested for equality against a list of values.

[char](#) - used for indicating the variable is of the type character.

[const](#) - makes the value of a pointer or a variable unmodifiable.

[continue](#) - used to skip certain statements inside the loop.

[default](#) - Used in case if it didn't match any of the case.

[do](#) - used for looping in C

[double](#) - represents double precision floating point data.

[else](#) - used for decision making in C (probably the last condition)

[enum](#) - used for defining enumerated type data type.

[extern](#) - used for indicating the variable is external and is declared globally.

[float](#) - represents single precision floating point data.

[for](#) - used for looping.

[goto](#) - used for unconditional jump to a labeled statement.

[if](#) - used for decision making.

[int](#) - indicates that the variable is of type integer.

[long](#) - type modifier that alters base datatype to new type.

[register](#) - variable of storage class register are much faster than normal variables.

[return](#) - terminates the execution of current function and returns its value to the calling function.

[short](#) - type modifier that alters base datatype to new type.

[signed](#) - type modifier that alters base datatype to new type.

[sizeof](#) - used to find the number of bytes of an object.

[static](#) - used for indicating the variable is of storage class static. It persists until the end of program.

[struct](#) - used in creating a structure which provide means to group different types of variable under one name for easier handling.

[switch](#) - switch statement tests the value of a expression with different case values.

[typedef](#) - used to explicitly associate a type with an identifier.

[union](#) - used in creating a union which provide means to group different types of variable under one name for easier handling.

[unsigned](#) - type modifier that alters base datatype to new type.

[void](#) - used to indicate that a function takes no arguments or returns no value.

[volatile](#) - used to create a volatile object. It can be modified in unspecified way by the hardware.

[while](#) - used for looping

Besides these keywords, there are some additional keywords supported by Turbo C: asm, far, interrupt, pascal, near, huge, cdecl...

IDENTIFIERS

"Identifiers" or "symbols" are the names you supply for variables, types, functions, and labels in your program. Identifier names must differ in spelling and case from any keywords. You cannot use keywords (either C or Microsoft) as identifiers; they are reserved for special use. You create an identifier by specifying it in the declaration of a variable, type, or function. In this example, result is an identifier for an integer variable, and main and printf are identifier names for functions.

Code #include <stdio.h>

```
int main()
{
    int result;

    if ( result != 0 )
        printf_s( "Bad file handle\n" );
}
```

Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements.

Following rules are to be followed while naming C variables: Name of identifier includes alphabets, digit and underscore.

First character of any identifier must be either alphabets or underscore.

Name of identifier cannot be any keyword of c program.

Name of function cannot be global identifier.

VARIABLES

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.

Code float sum;
Here, 'sum' is a variable of floating point type.

Following rules are to be followed while naming C variables: Variable name must begin with letter or underscore.

Variables are case sensitive

They can be constructed with digits, letters.

No special symbols are allowed other than underscore.

CONSTANTS

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. They can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well. They are treated just like regular variables except that their values cannot be modified after their definition.

Integer Constants

Integer constants are the numeric constants (constant associated with number) without any fractional part or exponential part. There are three types of integer constants in C language: decimal constant (base 10), octal constant (base 8) and hexadecimal constant (base 16) .

Decimal digits: 0 1 2 3 4 5 6 7 8 9

Octal digits: 0 1 2 3 4 5 6 7

Hexadecimal digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Examples:

Decimal constants: 6, 23, -3, 67, etc

Octal constants: 065, 034, 027, 054, etc

Hexadecimal constants: 0x5a, 0x252, 0x6f, etc

Floating Point Constants

A floating-point constant has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point constants either in decimal form or exponential form.

While representing using decimal form, you must include the decimal point, the exponent, or both and while representing using exponential form, you must include the integer part, the fractional part, or both. The signed exponent is introduced by e or E.

Examples: -5.6, 0.1256, -0.67E-5 (or 0.0000067)

Character Constants

A character constant is a single alphabet, a single digit or a single special symbol enclosed within single quotes.

The maximum length of a character constant is 1 character.

Examples: 'a', '\n', '\u02C0', etc.

There are some characters which have special meaning in C language.

They should be preceded by backslash symbol to make use of special function of them.

Given below is the list of some special characters and their purpose.

\b - Backspace

\f - Form feed

\n - New line

\r - Carriage return

\t - Horizontal tab

\” - Double quote

\' - Single quote

\\ - Backslash

\v - Vertical tab

\a - Alert or bell

\? - Question mark

\N - Octal constant (N is an octal constant)

\XN - Hexadecimal constant (N – hex.dcm1 cnst)

Following is the example to show few escape sequence characters:

Code #include <stdio.h>

```
int main()
{
    printf("Am\tLoving\nThis\tTutorial\n");

    return 0;
}
```

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

```
Code Am loving
This Tutorial
String Constants
```

String literals or constants are enclosed in double quotes `""`. A string contains characters that are similar to character literals: plain characters, escape sequences, and universal characters. You can break a long line into multiple lines using string literals and separating them using white spaces. Here are some examples of string literals. All the three forms are identical strings.

Code `"hello, dear"`

`"hello, \`

`dear"`

`"hello, " "d" "ear"`

Enumeration Constants

Keyword `enum` is used to declare enumeration types. Enums are numbers used to represent what you can think of as your own datatype that you create. For example:

Code `enum color {yellow, green, black, white};`

Here, the variable name is `color` and `yellow`, `green`, `black` and `white` are the enumeration constants having value 0, 1, 2 and 3 respectively by default. For more information about enumeration, visit page: [Enumeration Types](#).

OPERATORS

The type of operations that can be performed on the data objects are specified by operators. The data items on which an operator acts are called its operands. An operator can be unary, binary or ternary depending upon whether it operates on one, two or three operands. An operator along with its operands constitute a simple expression. A compound expression can be formed by using simpler expressions as operands of the different types of operators. The evaluation order of the operators in an expression will be determined by the operator precedence rules followed in the C language.

Arithmetic Operators

The binary arithmetic operators are `+`, `-`, `*`, `/`, which correspond to addition, subtraction, multiplication and division respectively. The modulus operator `%` returns the remainder after integer division. The `%` (modulus) operator cannot be applied to float or double type data. Arithmetic operators associate left to right.

The binary `+` and `-` operators have the same precedence when used in a calculation, but they have lower precedence than `*`, `/`, and `%` operators. To change the order of precedence brackets `()` are used.

Here is an example of a complex arithmetic expression, which involves multiple arithmetic operators.

Code `int a = 2, b = 4, c = 8, d = 9, e;`

`e = a * b / c - a * 2 + d * 3;`

Increment & Decrement Operators

In C, `++` and `--` are called increment and decrement operators respectively. Both of these operators are unary operators, i.e, used on single operand. `++` adds 1 to operand and `--` subtracts 1 to operand respectively. For example:

```
Code int a=5, b=10;
a++; //a becomes 6
a--; //a becomes 5
++a; //a becomes 6
--a; //a becomes 5
```

When i++ is used as prefix (like: ++var), ++var will increment the value of var and then return it but, if ++ is used as postfix (like: var++), operator will return the value of operand first and then only increment it. This can be demonstrated by an example:

```
Code #include <stdio.h>
int main(){
    int c=2,d=2;
    printf("%d\n",c++); //this statement displays 2 then, only c incremented by 1 to 3.
    printf("%d",++c); //this statement increments 1 to c then, only c is displayed.
    return 0;
}
```

Output generated for the above part of code will be:

Code 2

4

Assignment Operators

Unlike other structured programming languages such as Pascal or FORTRAN, C treats assignment of values to a variable to be result of evaluation of the assignment operator(=). The assignment of values to an identifier is done in the following way:

<identifier> = <expression>;

```
Code i = 15;
j = 17;
m = 8;
x = i * 3 + j - 5; (x gets 57)
k = (i = i + j * 4) - (m / 2); (here assignment operator is used in the right expression too. k gets 79)
The following is an example of multiple assignment.
```

```
Code i = j = 9; (is equivalent to j = 9; i = j;)
```

Note that the associativity of assignment operators is from right to left, hence assignments are carried out from right to left.

The compound assignment operators help in writing compact expressions.

For example: the statements

c = c + 5; can be written as c += 5;

f = f (g + h); can also be written as f *= (g+h); or f *= g+h;

Relational & Logical Operators

The relational operators supported in C are

> is greater than

< is less than

== is equal to

!= not equal to

>= is greater than or equal to

<= is less than or equal to

The relational operators >, <, >=, <= have the same precedence. The equality operator == and the not equal to operator != have lower precedence than the remaining relational operators. The relational operators have lesser precedence than arithmetic operators, so an expression like $i < j + 1$ will always be evaluated as $i < (j + 1)$. In C, if an expression evaluation yields zero value it is interpreted as false. Similarly it will be treated as true if the evaluation of an expression results in a non-zero value. The Logical operators supported in C are && (and) and || (or). Expressions connected by && and || are evaluated left to right, and evaluation stops as soon as the truth or falsehood of the condition is ascertained. The Unary negation operator (!) returns true if the expression evaluates false (i.e. the evaluation produces zero), false otherwise.

Wolfi - July 28, 2014, 11:43 am

DATA TYPES

C data types are defined as the data storage format that a variable can store a data to perform a specific operation. Data types are used to define a variable before to use in a program. Size of variable, constant and array are determined by data types.

Basic Data Types

1. Integer Data Type

Integer data type allows a variable to store numeric values.

“int” keyword is used to refer integer data type. The storage size of int data type is 2 or 4 or 8 byte. It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type. Like wise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.

int (2 byte) can store values from -32,768 to +32,767

int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.

If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

2. Character Data Type

Character data type allows a variable to store only one character.

Storage size of character data type is 1. We can store only one character using character data type. “char” keyword is used to refer character data type.

For example, ‘A’ can be stored using char datatype. You can’t store more than one character using char data type.

Please refer C – Strings topic to know how to store more than one characters in a variable.

3. Floating Point & Double Data Types

Floating Point data type consists of 2 types: Float and Double.

Float data type allows a variable to store decimal values.

Storage size of float data type is 4. This also varies depend upon the processor in the CPU as "int" data type.

We can use up-to 6 digits after decimal using float data type.

For example, 10.456789 can be stored in a variable using float data type.

Double data type is also same as float data type which allows up-to 10 digits after decimal.

The range for double datatype is from $1E-37$ to $1E+37$.

Enumeration Data Type

An enumeration consists of a set of named integer constants. An enumeration type declaration gives the name of the (optional) enumeration tag and defines the set of named integer identifiers (called the "enumeration set," "enumerator constants," "enumerators," or "members"). A variable with enumeration type stores one of the values of the enumeration set defined by that type.

Variables of enum type can be used in indexing expressions and as operands of all arithmetic and relational operators. Enumerations provide an alternative to the #define preprocessor directive with the advantages that the values can be generated for you and obey normal scoping rules.

In ANSI C, the expressions that define the value of an enumerator constant always have int type; thus, the storage associated with an enumeration variable is the storage required for a single int value. An enumeration constant or a value of enumerated type can be used anywhere the C language permits an integer expression.

Derived Data Types

Data types that are derived from fundamental data types are called derived data types. Derived data types don't create a new data type but, instead they add some functionality to the basic data types.

[Functions](#)

[Arrays & Pointers](#)

[Structures & Unions](#)

Void Data Type

The data type void actually refers to an object that does not have a value of any type. We have already seen examples of its use when we have defined functions that return no value, i.e. functions which only print a message and have no value to return. Such a function is used for its side effect and not for its value. In the function declaration and definition, we have indicated that the function does not return a value by using the data type void to show an empty type, i.e. no value. Similarly, when a function has no formal parameters, the keyword void is used in the function prototype and header to signify that there is no information passed to the function.

That's all about this tutorial on basics of C language...

Forgive my mistakes, and give me your valuable suggestions please... ?

References

As I told at the beginning, nothing posted above belongs to me. All credit goes to the original authors. ?

Following references are been used for making the above tutorial.

[Wikipedia](#)

[CProgramming](#)

[Tutorials Point](#)

[Programiz](#)

[MyCPlus](#)

[How2Lab](#)

[Fresh2Fresh](#)

Antigen - July 28, 2014, 11:45 am

That's a long concise and detailed tutorial, skimmed through it, seems goof enough, good to see new members contributing.

Wolfi - July 28, 2014, 11:50 am

(Jul 28, 2014 11:45 AM)Antigen Wrote: [CITE](#) That's a long concise and detailed tutorial, skimmed through it, seems goof enough, good to see new members contributing.

Thanks for your positive feed back mate.

As I told in the OP, all credit goes to the original authors... ?

Ether Copperfield - July 29, 2014, 9:10 am

Very nice tutorial. why don't You post more tutorials on languages like Python ?

P???ce - July 29, 2014, 10:11 am

If you wish to generate any number from 1-10;

```
#include<stdio.h>
void main()
{
for(int i=1;i<=10;i++)
printf("%d",i);
return 0;
}
```

if you wish to generate your name 5 times;

```
#include <stdio.h>
void main()
{
for(int i=0;i<5;i++)
printf("Prince\n");
return 0;
}
```

Good work OP ?

Wolfi - July 29, 2014, 11:59 am

(Jul 29, 2014 09:10 AM)Ether Copperfield Wrote: [\[CITE\]](#) Very nice tutorial. why don't You post more tutorials on languages like Python ?

Thanks for the feed back mate. ? Well, that's great. ?

Consider this as a beginning then. ? Kidding... ?

(Jul 29, 2014 10:11 AM)P???ce Wrote: [\[CITE\]](#) If you wish to generate any number from 1-10;

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
for(int i=1;i<=10;i++)
```

```
printf("%d",i);
```

```
return 0;
```

```
}
```

if you wish to generate your name 5 times;

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
for(int i=0;i<5;i++)
```

```
printf("Prince\n");
```

```
return 0;
```

```
}
```

Good work OP ?

Am sorry mate. I didn't understand what you meant by this post... ?

And, thanks for the positive feedback... ?

P???ce - July 29, 2014, 12:05 pm

(Jul 29, 2014 11:59 AM)Wolfi Wrote: [\[CITE\]](#) Thanks for the feed back mate. ? Well, that's great. ?

Consider this as a beginning then. ? Kidding... ?

Am sorry mate. I didn't understand what you meant by this post... ?

And, thanks for the positive feedback... ?

just two basic example programs related to for loop :)

welcome dear :)

Wolfi - July 30, 2014, 9:09 am

(Jul 29, 2014 12:05 PM)P???ce Wrote: [\[CITE\]](#) just two basic example programs related to for loop :)

welcome dear :)

Okay. ? If that's the case, adding something like:

'Consider this example' or 'You may try the following C code' will be great. ?

Thanks for that mate... ?

aswin567 - August 2, 2014, 4:07 am

That's a long concise and detailed tutorial, skimmed through it, seems goof enough, good to see new members contributing.

Ze.e.K.i.l.l - August 2, 2014, 1:09 pm

I know this is copy and paste none the less great to read.

Wolfi - August 3, 2014, 12:44 am

(Aug 2, 2014 04:07 AM)aswin567 Wrote: [\[CITE\]](#) That's a long concise and detailed tutorial, skimmed through it, seems goof enough, good to see new members contributing.

Thanks for that mate... ?

(Aug 2, 2014 01:09 PM)Ze.e.K.i.l.I Wrote: [\[CITE\]](#) I know this is copy and paste none the less great to read.

I didn't really understand what you tried to convey... ?

Maniac fx - August 4, 2014, 10:18 am

Great tutorial Wolfi. It will definitely help me improve my novice C knowledge.

Wolfi - August 5, 2014, 10:22 am

(Aug 4, 2014 10:18 AM)Maniac fx Wrote: [\[CITE\]](#) Great tutorial Wolfi. It will definitely help me improve my novice C knowledge.

It is a pleasure to know that it helped.

Thank you ma friend. ?

Maniac fx - August 12, 2014, 10:54 am

which is the best compiler for C in windows
i m using turbo C bt its nt working properly
any solutions

Wolfi - August 15, 2014, 4:02 am

(Aug 12, 2014 10:54 AM)Maniac fx Wrote: [\[CITE\]](#) which is the best compiler for C in windows
i m using turbo C bt its nt working properly
any solutions

For windows, you may use Dev C++ which is noob friendly.

You could get it from here:

http://dev-c.xtremedownload.com/?lp=adwords&tg=in&kw=Dev%20c%2B%2B&mt=b&ad=33374495913&pl=&ds=s&gclid=CjwKEAajw37afBRDO5M3h0qvj9zQSJACvwnOJ0d5-e86mB1iJStccYQLVyFkoopQN_btQYn0k2dd8shoCK67w_wcB

Qmz_ - August 15, 2014, 6:10 am

Do you code in C and wrote this with your own knowledge, or did you just copy and paste these from various websites?

Wolfi - August 15, 2014, 6:47 am

(Aug 15, 2014 06:10 AM)Qmz_ Wrote: [\[CITE\]](#) Do you code in C and wrote this with your own knowledge, or did you just copy and paste these from various websites?

I didn't write this by myself. I had referred a lot of sites that I've mentioned under Reference, for various sections in the thread.

After wards, I summed up it together and posted it here so that, at least some of the programming beginners in HF might find it worth. ?

BitCodex - August 15, 2014, 7:03 pm

This is really great, man. How long did it take you to type all this up? Either way it is greatly appreciated. Many people can learn from this including myself!

Wolfi - August 18, 2014, 9:03 am

(Aug 15, 2014 07:03 PM)BitCodex Wrote: [\[CITE\]](#) This is really great, man. How long did it take you

to type all this up? Either way it is greatly appreciated. Many people can learn from this including myself!

Thanks for the vouch mate. Its great to know it helped.

This thread is my very first one on HF.

Although, I referred a lot of sites for those, it took nearly a day to mix them up and align them in this way.

Best of luck learning one of the most fundamental programming language.

Feel free to ask any queries... ?

mehere - August 21, 2014, 6:59 am

Great read, I got into IT by doing a C course about 17 years ago, a great blast from the past. Given C is a great programming language for hacking this has just given me a quick refresher.

The Sly - August 21, 2014, 7:00 am

Holy shit that post. Thanks OP! Bookmarked!

Han. - August 21, 2014, 7:06 am

I didn't bother checking the rest of the post, but almost the entire first section is copied and pasted straight from Wikipedia. Why even bother?