1 Hello World

1.1 Hello Program 1

Using the File Manager (in KDE, Konqueror or in Gnome, Nautilus) create a new directory somewhere in your home directory called something appropriate for all the examples in this book, perhaps "Programming_In_Linux" without any spaces in the name.

Open an editor (in KDE, kate, or in Gnome, gedit) and type in (or copy from the supplied source code zip bundle) the following:

```
/*****************************************************************
C Programming in Linux (c) David Haskins 2008
chapter1_1.c
*****************************************************************/
#include <stdio.h>


int main(int argc, char *argv[])
{
        printf("Hello, you are learning C!!\n");
        return 0;
}
```

Save the text as **chapter1_1.c** in the new folder you created in your home directory.

Open a terminal window and type: **gcc -o hello chapter1_1.c**
to compile the program into a form that can be executed.

Now type "ls -l" to list the details of all the files in this directory. You should see that chapter1_2.c is there and a file called "hello" which is the compiled C program you have just written.

Now type: **./hello**
to execute, or run the program and it should return the text:

"Hello you are learning C!!".

If this worked, congratulations, you are now a programmer!

Anatomy of the program:

The part inside /*** ***/ is a comment and is not compiled but just for information and reference.

The “#include...” part tells the compiler which system libraries are needed and which header files are being referenced by this program. In our case “printf” is used and this is defined in the stdio.h header.

The “int main(int argc, char *argv[])” part is the start of the actual program. This is an entry-point and most C programs have a main function.

The “int argc” is an argument to the function “main” which is an integer count of the number of character string arguments passed in “char *argv[]” (a list of pointers to character strings) that might be passed at the command line when we run it.

A pointer to some thing is a name given to a memory address for this kind of data type. We can have a pointer to an integer: int *iptr, or a floating point number: float *fPtr. Any list of things is described by [], and if we know exactly how big this list is we might declare it as [200]. In this case we know that the second argument is a list of pointers to character strings.

Everything else in the curly brackets is the main function and in this case the entire program expressed as lines.

Each line or statement end with a semi-colon “;”.

We have function calls like “printf(...)” which is a call to the standard input / output library defined in the header file stdio.h.

At the end of the program “return 0” ends the program by returning a zero to the system.

Return values are often used to indicate the success or status should the program not run correctly.

1.2 Hello Program 2

Taking this example a stage further, examine the start of the program at the declaration of the entry point function: int main(int argc, char *argv[])

In plain English this means:

The function called "main", which returns an integer, takes two arguments, an integer called "argc" which is a count of the number of command arguments then *argv[] which is a list or array of pointers to strings which are the actual arguments typed in when you run the program from the command line.

Some Definitions:

function: a block of program code with a **return data type**, a name, some arguments of varying data types separated by commas, enclosed in **brackets**, then the body of the function enclosed in **curly brackets**, each statement ending with a **semi-colon**.
integer symbol **int** : a counting number like 0,1,2,3,4,5.
list, array symbol **[]:** a sequence of things of the same kind in a numbered order.
pointer symbol ***** : a memory address locating the start of piece of data of a certain type.
string or **char *** : a pointer to a sequence of characters like 'c' ,'a', 't' making up "cat". A character string ends with s special character NULL or '\0' ascii value 0 or hex 00

Let's rewrite the program to see what all this means before we start to panic.

ie business school
#1 EUROPEAN BUSINESS SCHOOL FINANCIAL TIMES 2013
#gobeyond
MASTER IN MANAGEMENT
Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.
Choose your area of specialization.
Customize your master through the different options offered.
Global Immersion Weeks in locations such as London, Silicon Valley or Shanghai.
Because you change, we change with you.
www.ie.edu/master-management
mim.admissions@ie.edu

```
/****************************************************************
C Programming in Linux (c) David Haskins 2008
chapter1_2.c
*****************************************************************/
#include <stdio.h>


int main(int argc, char *argv[])
{
        int i=0;
    printf("Hello, you are learning C!!\n");
    printf("Number of arguments to the main function:%d\n", argc);
        for(i=0; i<argc; i++)
        {
        printf("argument number %d is %s\n", i, argv[i]);
        }
    return 0;
}
```

Save the text as **chapter1_2.c** in the same folder.

Open a terminal window and type:
gcc -o hello2 chapter1_2.c to compile the program into a form that can be executed.

Now type **ls -l** to list the details of all the files in this directory. You should see that chapter1_2.c is there and a file called **hello2** which is the compiled C program you have just written.

Now type **./hello2** to execute, or run the program and it should return the text:

```
Hello, you are still learning C!!
Number of arguments to the main function:1
argument number 0 is ./hello2
```

We can see that the name of the program itself is counted as a command line argument and that the counting of things in the list or array of arguments starts at zero not at one.

Now type **./hello2 my name is David** to execute the program and it should return the text:

```
Hello, you are still learning C!!
Number of arguments to the main function:5
argument number 0 is ./hello2
argument number 1 is my
argument number 2 is name
argument number 3 is is
argument number 4 is David
```

So, what is happening here? It seems we are reading back each of the character strings (words) that were typed in to run the program.

Anatomy of the program:

printf("Hello, you are learning C!!\n");

the library function printf is called with one argument, a character string ending with a \n or new line character.

printf("Number of arguments to the main function:%d\n", argc);

the library function printf is called with two arguments, a character string ending with a \n that includes %d as a placeholder for the second argument argc which is an int.

for(i=0; i<argc; i++)

is a "for loop" in which we do something repeatedly using a counter integer i which is incremented (by the expression i++) at each iteration or looping which continues while i stays less than the value of argc

printf("argument number %d is %s\n", i, argv[i]);

the library function printf is called with three arguments, a character string ending with a \n that includes %d as a placeholder for the second argument argc which is an int, and %s which is a placeholder for the third argument argv[i], the i-th member of the array of pointers to character strings called argv[].

1.3 Hello Program 3

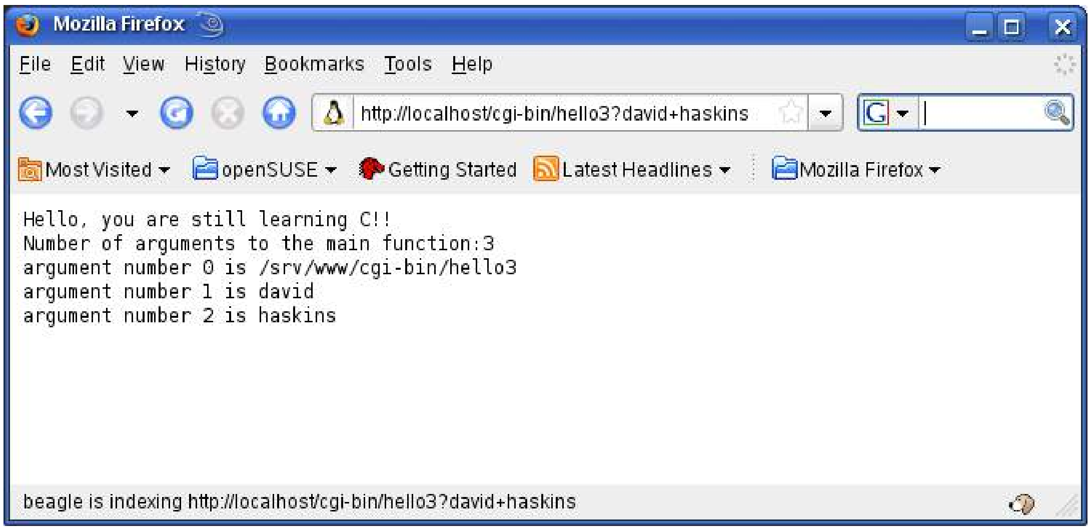
Lets get real and run this in a web page. Make the extra change adding the first output printf statement "Content-type:text/plain\n\n" which tells our server what kind of MIME type is going to be transmitted.

Compile using **gcc -o hello3 chapter1_3.c** and copy the compiled file hello3 to your public_html/cgi-bin directory (or on your own machine as superuser copy the program to /srv/www/cgi-bin (OpenSuse) or /usr/lib/cgi-bin (Ubuntu)).

```
/*****************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter1_3.c                                        *
*****************************************************************/
#include <stdio.h>


int main(int argc, char *argv[])
{
        int i=0;

    printf("Content-type:text/plain\n\n");
    printf("Hello, you are still learning C!!\n");
    printf("Number of arguments to the main function:%d\n", argc);
        for(i=0;i<argc;i++)
        {
        printf("argument number %d is %s\n", i, argv[i]);
        }
     return 0;
}
```

Open a web browser and type in the URL http://localhost/cgi-bin/hello3?david+haskins and you should see that web content can be generated by a C program.

"I studied English for 16 years but... ...I finally learned to speak it in just six lessons"
Jane, Chinese architect
ENGLISH OUT THERE
Click to hear me talking before and after my unique course download


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/cgi-bin/hello3?david+haskins
Most Visited
openSUSE
Getting Started
Latest Headlines
Mozilla Firefox

```
Hello, you are still learning C!!
Number of arguments to the main function:3
argument number 0 is /srv/www/cgi-bin/hello3
argument number 1 is david
argument number 2 is haskins
```

beagle is indexing http://localhost/cgi-bin/hello3?david+haskins

1.4 Hello Program 4

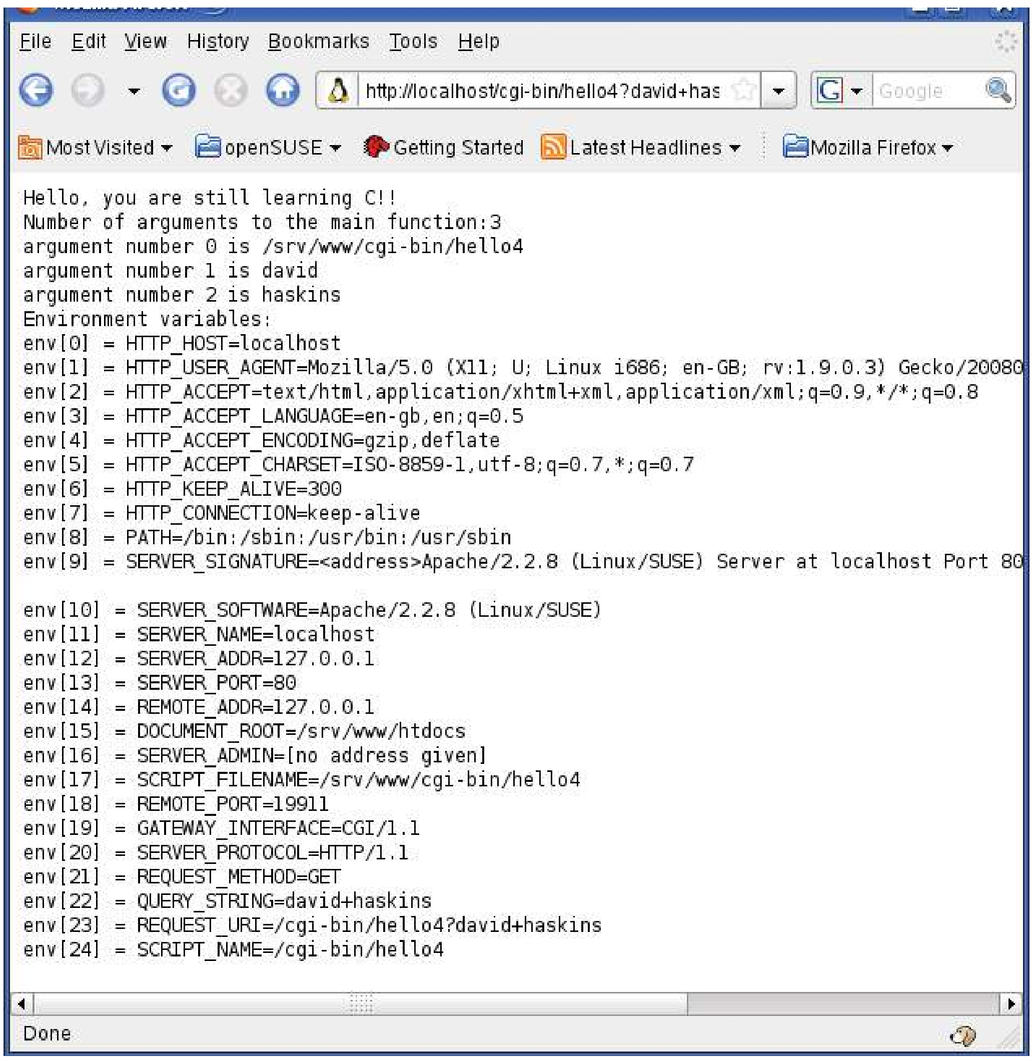
A seldom documented feature of the function signature for "main" is that it can take **three** arguments and the last one we will now look at is char *env[ ] which is also a list of pointers to strings, but in this case these are the **system environment variables** available to the program at the time it is run

```
/*********************************************************************
* C Programming in Linux (c) David Haskins 2008
* chapter1_4.c                                                       *
*********************************************************************/
#include <stdio.h>


int main(int argc, char *argv[], char *env[])
{
        int i=0;

    printf("Content-type:text/plain\n\n");
    printf("Hello, you are still learning C!!\n");
    printf("Number of arguments to the main function:%d\n", argc);
        for(i=0;i<argc;i++)
        {
        printf("argument number %d is %s\n", i, argv[i]);
        }
        i = 0;
        printf("Environment variables:\n");
        while(env[i])
        {
                printf("env[%d] = %s\n", i, env[i]);
                i++;
        }
    return 0;
}
```

Compile with **gcc -o hello4 chapter1_4.c** and as superuser copy the program to /srv/www/cgi-bin (OpenSuse) or /usr/lib/cgi-bin (Ubuntu). You can run this from the terminal where you compiled it with **./hello4** and you will see a long list of environment variables. In the browser when you enter http://localhost/cgi-bin/hello4 you will a different set altogether.

Wikipedia defines **environment variables** like this:

"In all Unix and Unix-like systems, each process has its own private set of environment variables. By default, when a process is created it inherits a duplicate environment of its parent process, except for explicit changes made by the parent when it creates the child........ All Unix operating system flavors as well as DOS and Microsoft Windows have environment variables; however, they do not all use the same variable names. Running programs can access the values of environment variables for configuration purposes. Examples of environment variables include...... PATH, HOME... "


File Edit View History Bookmarks Tools Help
http://localhost/cgi-bin/hello4?david+has
Google
Most Visited
openSUSE
Getting Started
Latest Headlines
Mozilla Firefox
Done

```
Hello, you are still learning C!!
Number of arguments to the main function:3
argument number 0 is /srv/www/cgi-bin/hello4
argument number 1 is david
argument number 2 is haskins
Environment variables:
env[0] = HTTP_HOST=localhost
env[1] = HTTP_USER_AGENT=Mozilla/5.0 (X11; U; Linux i686; en-GB; rv:1.9.0.3) Gecko/20080
env[2] = HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
env[3] = HTTP_ACCEPT_LANGUAGE=en-gb,en;q=0.5
env[4] = HTTP_ACCEPT_ENCODING=gzip,deflate
env[5] = HTTP_ACCEPT_CHARSET=ISO-8859-1,utf-8;q=0.7,*;q=0.7
env[6] = HTTP_KEEP_ALIVE=300
env[7] = HTTP_CONNECTION=keep-alive
env[8] = PATH=/bin:/sbin:/usr/bin:/usr/sbin
env[9] = SERVER_SIGNATURE=<address>Apache/2.2.8 (Linux/SUSE) Server at localhost Port 80

env[10] = SERVER_SOFTWARE=Apache/2.2.8 (Linux/SUSE)
env[11] = SERVER_NAME=localhost
env[12] = SERVER_ADDR=127.0.0.1
env[13] = SERVER_PORT=80
env[14] = REMOTE_ADDR=127.0.0.1
env[15] = DOCUMENT_ROOT=/srv/www/htdocs
env[16] = SERVER_ADMIN=[no address given]
env[17] = SCRIPT_FILENAME=/srv/www/cgi-bin/hello4
env[18] = REMOTE_PORT=19911
env[19] = GATEWAY_INTERFACE=CGI/1.1
env[20] = SERVER_PROTOCOL=HTTP/1.1
env[21] = REQUEST_METHOD=GET
env[22] = QUERY_STRING=david+haskins
env[23] = REQUEST_URI=/cgi-bin/hello4?david+haskins
env[24] = SCRIPT_NAME=/cgi-bin/hello4
```

We will soon find out that **QUERY_STRING** is an important environment variable for us in communicating with our program and in this case we see it has a value of "david+haskins" or everything after the "?" in the URL we typed. It is a valid way to send information to a **common gateway interface** (CGI) program like hello4 but we should restrict this to just one string. In our case we have used a "+" to join up two strings. If we typed: "david haskins" the browser would translate this so we would see:

```
QUERY_STRING=david%20haskins
```

We will learn later how complex sets of input values can be transmitted to our programs.

AACSB ACCREDITED
"The perfect start of a successful, international career."
CLICK HERE to discover why both socially and academically the University of Groningen is one of the best places for a student to be
www.rug.nl/feb/education

1.5 Hello World conclusion

We have seen that a simple program with a tiny bit of input and some output is in fact extremely powerful in that it reveals and exposes the inner workings of a great deal of our computer.

Even though we have just begun we have encountered many of the key concepts we will use over and over again:

- functions and arguments
- Numbers (integers) and character strings as data types
- Lists or arrays
- Loops using "for" and "while"

We have made a deliberate big leap from writing a program that runs simply in a "terminal screen" to one which will be visible over the internet in a browser.

The reason for this is that the process of writing programs that interact with users in windowing systems like Windows, Gnome or KDE is extremely complex and not something you will be asked very often to do .

The internet browser has become the de facto interface mode for almost everything we do these days so we might as well understand using it from the start.

In all the successive chapters we will follow this model: starting off with some basic technique then applying it to a web-based system.

In practice there is not much real-world C common gateway interface programming going on but there is a great deal of C and C++ based code running as Apache modules and Microsoft IIS ISAPI Dlls. Perhaps not many know that much of Ebay is written in C / C++.

Why? It is as fast as things get and their business with the bargain snipers in a global real-time market needs this lightning fast core, so there is no other way to get that performance.