

MODULE Y THE `main()` AND COMMAND-LINE ARGUMENT

My Training Period: hours

Note:

Compiled using Microsoft Visual C++ / .Net, empty win32 console mode application. Some examples also tested using Borland C++ Builder 5.02. **gcc** compilation example is given at the end of this Module.

For the `main()` command line arguments and **pointer** story, please read [Module 8](#), Section 8.9. For wide character and Unicode `wmain()` please refer to [Module G](#) (Story) and [Module M](#) (implementation).

Abilities:

- Able to understand and use a portable `main()` versions and their variation.
- Able to understand and use programs that accept command-line arguments.
- Able to build programs that accept command-line arguments.
- Able to build programs that can run with options/switches.

Y.1 The main Function and Program Execution

- The ‘concepts’ discussed in this Module may be very useful for UNIX / Linux (or command line tools for Microsoft Windows) programmers because of the extensive usage of the command-line programs or tools.
- A special function called `main()` is the starting point of execution for all C and C++ programs. It is not predefined by the compiler; so that, it must be supplied in the program.
- If your code adheres to the Unicode programming model, then you can use the wide-character version of `main()`, `wmain()`, but it is implementation dependent.
- The `main()` function serves as the starting point for program execution. It usually controls program execution by directing the calls to other functions in the program.
- A program usually stops executing at the end of `main()`, denoted by closing curly brace `}`, although it can terminate at other points in the program for a variety of reasons such as the forced program termination by using the `exit()` function.
- As you have learned before, functions within the program body perform one or more specific tasks. The `main()` function can call these functions to perform their respective tasks as shown in the following program skeleton.

```
int main()
{
    //function call...
    MyFunction();
    ...
    //another function call...
    YourFunction();
    ...
    //another function call...
    OurFunction();
    ...
    return 0;
}
```

- When `main()` calls another function, it passes execution control to the function, then, execution begins at the first statement in the function. A function returns control to `main()` when a `return` statement is executed or when the end of the function is reached denoted by the closing curly brace.
- If you aware, all the program examples presented in this tutorial do not have parameters in the `main()` function. Actually there are others version, that you can declare `main()` function, to have parameters, as any other functions but with a few exceptions.
- As explained before in Function Module, the term **parameter** or **formal parameter** refers to the identifier that receives a value passed to a function. When one function calls another, the called function receives values for its parameters from the calling function.
- These **values** are called **arguments**. The parameter(s) just acts as a placeholder. You can declare parameters to `main()` so that **it can receive arguments from the command line** using the following format:

```
int main(int argc, char *argv[])
{
```

```

...
return 0;
}

```

- Or

```

int main(int argc, char *argv[], char *envp[])
{
...
return 0;
}

```

- There is no prototype declared for `main()`, and as a conclusion, we have `main()` that can be defined with zero, two, or three parameters as shown below.

```

int main(void)
{
//...
return 0;
}
int main(int argc, char *argv[])
{
//...
return 0;
}

//implementation dependant
int main(int argc, char *argv[], char *envp[])
{
//...
return 0;
}

```

- When you want to pass information to the `main()` function, the parameters are traditionally named `argc` and `argv`, although for C compiler does not require these names.
- The types for `argc` and `argv` are defined by the C language. Traditionally, if a third parameter is passed to `main()`, that parameter is named `envp`, a Microsoft extension to the ANSI C (ISO/IEC C) standard (or `env` for Borland®).
- The following table lists some description of the parameters.

Parameter	Description
<code>argc</code>	For argument count, an integer that contains the count of arguments that follows in <code>argv</code> . Since the program name is considered an argument, the <code>argc</code> parameter is always greater than or equal to 1. Type is <code>int</code> .
<code>*argv[]/**argv</code>	For argument vector, an array of null-terminated strings representing command-line arguments entered by the user of the program. All elements of the <code>argv</code> array are pointers to strings. By convention, <code>argv[0]</code> is the command with which the program is invoked, that is the program name ; then, <code>argv[1]</code> is the first command-line argument, and so on, until <code>argv[argc]</code> , which is always NULL pointer. The first command-line argument is always <code>argv[1]</code> and the last one is <code>argv[argc-1]</code> . All elements of the <code>argv</code> array are pointers to strings. Type is <code>char</code> .
<code>*envp[]</code> <code>*env(Borland®)</code>	Is an array of pointers to environment variables. The <code>envp</code> array is terminated by a null pointer. This is a Microsoft extension to the ANSI C standard and also used in Microsoft C++. It is an array of strings representing the variables set in the user's environment. This array is terminated by a NULL entry. It can be declared as an array of pointers to <code>char</code> (<code>char *envp[]</code>) or as a pointer to pointers to <code>char</code> (<code>char **envp</code>). If your program uses <code>wmain()</code> instead of <code>main()</code> , use the <code>wchar_t()</code> data type instead of <code>char</code> . The environment block passed to <code>main()</code> and <code>wmain()</code> is a <i>frozen</i> copy of the current environment. If you subsequently change the environment via a call to <code>putenv()</code> or <code>_wputenv()</code> , the current environment (as returned by <code>getenv()/_wgetenv()</code> and the <code>_environ()/_wenviron()</code> variable) will change, but the block pointed to by <code>envp</code> will not change.

This argument is ANSI compatible in C, but not in C++. It is also a common extension in many Linux/UNIX® systems.

Table Y.1: main() parameters

- A program invoked with no command-line arguments will receive a value of one for `argc`, as the **program name** of the executable file is placed in `argv[0]`. Strings pointed to by `argv[1]` through `argv[argc-1]` represent program parameters.
- The simplest illustration is the `echo` program, which echoes its command-line arguments on a single line, separated by blanks as shown below when you run at the command prompt of Windows Operating System.

```
echo Hello world!
```

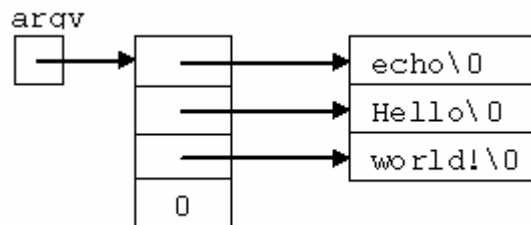
- Prints the output:

```
Hello world!
```

- As explained before, by convention, `argv[0]` is the **program name** by which the program was invoked, so `argc` is at least 1. If `argc` is 1, there are no command-line arguments after the **program name**.
- In the example above, `argc` is 3, and...

```
argv[0] is echo      - the program name
argv[1] is Hello    - string
argv[2] is world!   - string
```

- The first optional argument is `argv[1]` and the last is `argv[argc-1]`; additionally the standard requires that `argv[argc]` be NULL pointer. The explanation can be illustrated as shown below.



- The following program example is the version of `echo` program that treats `argv` as an array of character pointers:

```
//program & file names, myecho.exe
//compiled using Visual C++ .Net
#include <stdio.h>

//echo command-line argument
//using array of character pointers
int main(int argc, char *argv[])
{
    int i;
    for(i=1; i<argc; i++)
        printf("%s%s", argv[i], (i<argc-1)? " ":"");
    printf("\n");
    return 0;
}
```

- Run the `myecho.exe` at console/command prompt where the executable program is located, to see the output. For the following output, `myecho.exe` is at the C: drive.

```

C:\WINDOWS\system32\cmd.exe

C:\>myecho Testing the command-line arguments!
Testing the command-line arguments!

C:\>myecho The new echo program!
The new echo program!

C:\>myecho It's working!!!
It's working!!!

C:\>

```

- Since argv is a pointer to an array of pointers, we can manipulate the pointer rather than the index of the array.
- The next variation is based on incrementing argv, which is a pointer to pointer to char, while argc is counted down.

```

//program & file names, myecho.exe
//compiled using Visual C++ .Net
#include <stdio.h>

//another version of the myecho program,
//command-line argument
int main(int argc, char *argv[])
{
    while(--argc > 0)
        printf("%s%s",*++argv, (argc > 1) ? " ": "");
    printf("\n");
    return 0;
}

```

- The output:

```

C:\WINDOWS\system32\cmd.exe

C:\>myecho Testing another version of the echo program!
Testing another version of the echo program!

C:\>myecho You see how it works?
You see how it works?

C:\>

```

- Since argv is a pointer to the beginning of the array of argument strings, incrementing it by 1 (++argv) makes it point at the original argv[1] instead of argv[0].
- Each successive increment moves it along to the next argument; *argv is then the pointer to that argument. At the same time, argc is decremented; when it becomes zero, there are no arguments left to print. Alternatively we could write the printf() statement as:

```
printf((argc > ?) ? "%s " : "%s", *++argv);
```

- This shows that the format argument of printf() can also be an expression.
- Consider the following program example, very simple text pattern search program.

```

//searchpattern.cpp, compiled using
//Visual C++ .Net
#include <stdio.h>
//maximum input line length
#define MAXLINE 100

//Function prototypes...
int getline(char line[], int max);
int strindex(char source[], char searchfor[]);

//pattern to be searched for in the line
char pattern[] = "eat";

```

```

//find/display all line matching pattern
main()
{
    char line[MAXLINE];
    int found = 0;

    printf("Searching \'eat\' in the line of text\n");
    printf("Enter line of text:\n");
    while(getline(line, MAXLINE) > 0)
        if(strindex(line, pattern) >= 0)
            {
                printf("%s", line);
                found++;
            }

    return found;
}

//getline, get line into string str, return the length
int getline(char str[], int lim)
{
    int count, i;
    i=0;

    while(--lim > 0 && (count=getchar()) != EOF && count != '\n')
        str[i++] = count;
    if(count=='\n')
        str[i++] = count;
    str[i]='\0';
    return i;
}

//strindex, return index of t in str, -1 if none
int strindex(char str[], char t[])
{
    int i, j, k;
    for(i=0;str[i]!='\0';i++)
    {
        for(j=i, k=0; t[k] != '\0' && str[j] == t[k]; j++, k++)
            ;
        if(k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}

```

- The above program named searchpattern and when the searchpattern.exe is run, the output is shown below.

```

C:\> "g:\vcnetprojek\searchpattern\debu...
Searching 'eat' in the line of text
Enter line of text:
Testing line one
Line 2, eat the cake!
Line 2, eat the cake!
Another line with pattern 'eat'...
Another line with pattern 'eat'...
No 'eat' in this line?
No 'eat' in this line?
Nothing ... in this line
hahahahaha
heheheheat
heheheheat
Stop by Ctrl+Z, depend on your system!
^Z
Press any key to continue

```

Y.2 Command Line Argument

- Let change the program so that the pattern to be matched is specified by the first argument on the command line. As an example, let create a simple program skeleton.

```

//searchpattern.cpp, compiled using VC++ .Net
//not usable...just program skeleton
#include <stdio.h>
#include <string.h>

```

```

#define MAXLINE 100

int getline(char *line, int max)
{
printf("In getline() function\n");
//put getting line of text codes here..
return 0;}

//find and print lines that match pattern
//from the 1st argument
main(int argc, char *argv[])
{
    char line[MAXLINE];
    int found = 0;

    //If just program name (argc = 1), then...
    if(argc != 2)
        printf("Usage: searchpattern thepattern\n");

    //If the program name with switch/option (argc = 2), then...
    else

        while(getline(line, MAXLINE) > 0)
        {
            if(strstr(line, argv[1]) != NULL)
                {printf("%s", line);
                 found++;}
        }
    return found;
}

```

- The output when running the program at command prompt or where the searchpattern.exe is located is shown below.

```

C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\mike>cd\

C:\>searchpattern
Usage: searchpattern thepattern

C:\>searchpattern test
In getline() function

C:\>

```

- And the following is the working program example of the searchpattern, using command line argument.

```

//searchpattern.cpp, compiled using VC++ .Net
#include <stdio.h>
#include <string.h>
#define MAXLINE 100

int getline(char *line, int max)
{
int count, i;
i=0;

while(--max > 0 && (count=getchar()) != EOF && count != '\n')
    line[i++] = count;
    if(count=='\n')
        line[i++] = count;
    line[i]='\0';
    return i;
}

//find and print lines that match the
//pattern from the 1st argument
main(int argc, char *argv[])
{

```

```

char line[MAXLINE];
int found = 0;

//If just program name (argc = 1), then...
if(argc != 2)
    printf("Usage: searchpattern thepattern\n");

//If the program name with switch/option (argc = 2), then...
else

    while(getline(line, MAXLINE) > 0)
    {
        if(strstr(line, argv[1]) != NULL)
        {
            printf("%s", line);
            found++;
        }
    }
    return found;
}

```

- The output, when running the `searchpattern.exe` at C: (or run it where ever the `searchpattern.exe` is located):

```

C:\>searchpattern
Usage: searchpattern thepattern

C:\>searchpattern test
testing the first line
testing the first line
yes, no ... in this line
Oh noooo!!! test is here!!
Oh noooo!!! test is here!!
^Z

C:\>

```

- The standard library function `strstr(a, b)` returns a pointer to the first occurrence of the string `b` in the string `a`, or `NULL` if there is none. It is declared in `<string.h>` or C++ wrapper `<cstring>`.
- The program skeleton can now be expanded to illustrate further the pointer constructions. Suppose we want to allow 2 optional arguments (or switches). The first one is: to print all line except those that match the pattern; the second one is: precede each printed line by its line number.

Y.3 Command line and switches/options

- For C/C++ programs on UNIX/Linux systems or Windows command line, conventionally, an argument that begins with a **minus sign** introduces an optional flag or parameter, normally called switch or option.
- For Windows, the forward slash (/) also used together with the switches. For example, if we choose -v (for inversion) to signal the inversion, and -n (for number) to request line numbering, then the command line:

```
searchpattern -v -n thepattern
```

- Will print each line that doesn't match the pattern, preceded by its line number.
- Optional arguments should be permitted in any order, and the rest of the program should be independent of the number of arguments that were present.
- Furthermore, it is also convenient for users if option arguments can be combined, as in the following command line:

```
searchpattern -vn thepattern
```

- The following is the working program example.

```
//searchpattern.cpp, compiled using
```

```

//VC++ .Net
#include <stdio.h>
#include <string.h>
//maximum input line length
#define MAXLINE 100

int getline(char line[], int max)
{
int count, i;
i=0;

while(--max > 0 && (count=getchar()) != EOF && count != '\n')
    line[i++] = count;
if(count=='\n')
    line[i++] = count;
line[i]='\0';
return i;
}

//find all line of text matching pattern
//supplied by command line argument
main(int argc, char *argv[])
{
    char line[MAXLINE];
    long linenum = 0;
    int theoption, except = 0, number = 0, found =0;

    //check the minus sign...
    while(--argc > 0 && (*++argv)[0] == '-')
        //check the option...
        while(theoption = *++argv[0])
            switch (theoption)
            {
                case 'v':
                    except = 1;
                    break;
                case 'n':
                    number = 1;
                    break;
                default:
                    printf("searchpattern: illegal option %s\n", theoption);
                    argc = 0;
                    found = -1;
                    break;
            }
        //
        if(argc != 1)
        {
            //some help...
            printf("Usage: searchpattern -v -n thepattern\n");
            printf("Try: searchpattern -v -n test\n");
            printf("Try: searchpattern -v test\n");
            printf("Try: searchpattern -n test\n");
            printf("Try: searchpattern -vn test\n");
            printf("Then enter line of text, with or w/o the pattern!\n");
        }
        //
        else
            while(getline(line, MAXLINE) > 0)
            {
                linenum++;
                if((strstr(line,*argv) != NULL) != except)
                {
                    if(number)
                        printf("%ld:", linenum);
                    printf("%s", line);
                    found++;
                }
            }
        return found;
}

```

- The output when running the `searchpattern.exe` at the command prompt C: (or run it where it is located). For Borland 5.02, you have to run the full command line as shown below.

```
searchpattern -v -n test
```



```

C:\WINDOWS\system32\cmd.exe

C:\>searchpattern
Usage: searchpattern -v -n thepattern
Iry: searchpattern -v -n test
Iry: searchpattern -v test
Iry: searchpattern -n test
Iry: searchpattern -vn test
Then enter line of text, with or w/o the pattern?

C:\>searchpattern -v test
testing line one
nothing in this line
nothing in this line
the test pattern is working
yesssss...
yesssss...
^Z

C:\>searchpattern -v -n test2
testing the test2
no ... in this line
2:no ... in this line
again no ... in this line
3:again no ... in this line
^Z

C:\>searchpattern -vn test3
nothing in this line
1:nothing in this line
testing the test3
again successful!
3:again successful!
^Z

C:\>searchpattern -n test4
testing the test4 pattern
1:testing the test4 pattern
just line number for -n
check the test4 pattern
3:check the test4 pattern
^Z

C:\>

```

- The parameters `argc` and `argv` are modifiable and retain their last-stored values between program startup and program termination.
- For wide-character, Microsoft implementation, the declaration syntax for `wmain()` is as follows:

```

int wmain( )
{
    //...
    return 0;
}

```

- Or, optionally:

```

int wmain(int argc, wchar_t *argv[], wchar_t *envp[])
{
    //...
    return 0;
}

```

- For Microsoft implementation please refer to [Module G](#) (Story) and [Module M](#) for `wmain()`.
- For Borland®, `env[]` is used instead of `envp[]`. For UNIX/Linux, please check their documentation :o).
- The types for `argc` and `argv` are defined by the language. The names `argc`, `argv`, and `envp` are traditional, but are not required by the compiler. So, you may use other name isn't it? Then try yourself :o).

- Alternatively, the `main()` and `wmain()` functions can be declared as returning `void` (no return value). If you declare `main()` or `wmain()` as returning `void`, you cannot return an exit code to the parent process or operating system using a `return` statement.
- To return an exit code when `main()` or `wmain()` is declared as `void`, you must use the `exit()` function.

Y.4 Some `main()` Function Restrictions

- Several restrictions apply to the `main()` function that do not apply to any other C/C++ functions. The `main()` function compared to normal function:
 - Cannot be overloaded.
 - Cannot be declared as `inline`.
 - Cannot be declared as `static`.
 - Cannot have its address taken.
 - Cannot be called.
- The following example shows how to use the `argc`, `argv`, and `envp` arguments to `main()`:

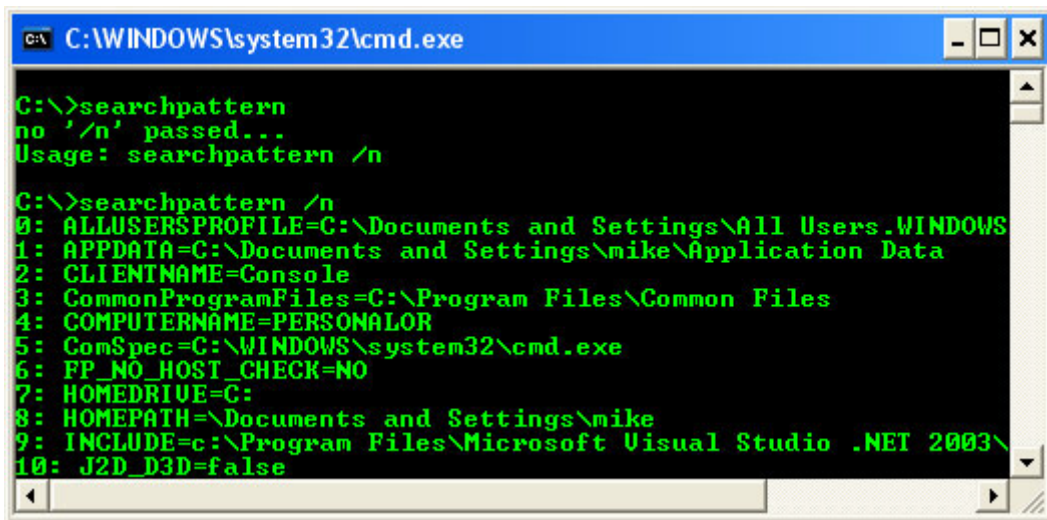
```
//main()arguments program example
//C++ codes
#include <iostream>
//For Borland 5.02 you may use <string.h> instead of <cstring>
//and comment out the 'using namespace std';
#include <cstring>
using namespace std;

int main(int argc, char *argv[], char *envp[])
{
//The default is no line numbers.
int LineNum = 0;

//If /n is passed to the .exe program, display
//numbered listing of environment variables.
//If program name and switch/option... AND strcmp...
if((argc == 2) && strcmp(argv[1], "/n" ) == 0)
LineNum = 1;
else
cout<<"no '/'\n\' passed..."<<endl;

//Walk through list of strings until a NULL is encountered.
for(int i = 0; envp[i] != NULL; ++i)
{
if(LineNum)
cout<<i<<": "<<envp[i]<<"\n";
}
cout<<"Usage: searchpattern /n\n";
return 0;
}
```

- The output, when `searchpattern.exe` run at command prompt (or where ever the `.exe` is located), depends on your environment setting, is shown below.



Y.5 Parsing C Command-Line Arguments – Microsoft Implementation

- For information, Microsoft uses Microsoft C Runtime (CRT) for C codes; that is Microsoft C version.
- Microsoft C startup code uses the following rules when interpreting arguments given on the operating system command line:
 - Arguments are delimited by white space, which is either a space or a tab.
 - A string surrounded by double quotation marks is interpreted as a single argument, regardless of white space contained within. A quoted string can be embedded in an argument. Note that the caret (^) is not recognized as an escape character or delimiter.
 - A double quotation mark preceded by a backslash, \", is interpreted as a literal double quotation mark (").
 - Backslashes are interpreted literally, unless they immediately precede a double quotation mark.
 - If an even number of backslashes is followed by a double quotation mark, then one backslash (\) is placed in the argv array for every pair of backslashes (\\), and the double quotation mark (") is interpreted as a string delimiter.
 - If an odd number of backslashes is followed by a double quotation mark, then one backslash (\) is placed in the argv array for every pair of backslashes (\\) and the double quotation mark is interpreted as an escape sequence by the remaining backslash, causing a literal double quotation mark (") to be placed in argv.
- The following list illustrates the above listed rules by showing the interpreted result passed to argv for several examples of command-line arguments. The output listed in the second, third, and fourth columns is from the following program example.

Command-Line Input	argv[1]	argv[2]	argv[3]
"a b c" d e	a b c	d	e
"ab\"c" "\\\" d	ab"c	\	d
a\\b d"e f"g h	a\\b	de fg	h
a\\"b c d	a\"b	c	d
a\\\\"b c" d e	a\\b c	d	e

Table Y.2

```

//searchpattern.cpp
//compiled using VC++ .Net and Borland C++ 5.02
//run on windows machine...
#include <stdio.h>
//For VC++ .Net, comment out the above #include <stdio.h>
//and uncomment the following...
//include <cstdio>
//using namespace std;

int main(int argc, /*Number of strings in array argv*/
char *argv[], /*Array of command-line argument strings*/
char **envp) /*Array of environment variable strings*/
{

```

```

int count;

/*Display each command-line argument.*/
printf("\nThe command-line arguments:\n");
for(count = 0; count < argc; count++)
    printf(" argv[%d]  %s\n", count, argv[count]);

/*Display each environment variable.*/
printf("\nEnvironment variables:\n");
while(*envp != NULL)
    printf("  %s\n", *(envp++));

return 0;
}

```

- The output, when running the .exe, depends on your environment setting, is shown below:

```

C:\ "g:\vcnetprojek\searchpattern\Debug\searchpattern.exe"
The command-line arguments:
argv[0]  g:\vcnetprojek\searchpattern\Debug\searchpattern.exe
Environment variables:
ALLUSERSPROFILE=C:\Documents and Settings\ALL Users\WINDOWS
APPDATA=C:\Documents and Settings\mike\Application Data
CLIENTNAME=Console
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=PERSONALOR
ComSpec=C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Documents and Settings\mike
INCLUDE=c:\Program Files\Microsoft Visual Studio .NET 2003\SDK

```

- For windows system, you can put the command line program (executable programs) in the System32 folder, and then you can run it from any relative or absolute path. For Linux it is/sbin or bin.
- Program example compiled using gcc.

```

/*****cmdline.c*****/
/*****Run on FeDorA 3 Machine*****/
#include <stdio.h>
/*For C++ compiler, comment out the above #include <stdio.h>*/
/*and uncomment the following.*/
#include <cstdlib>
/*using namespace std;*/

int main(int argc, /*Number of strings in array argv*/
char *argv[], /*Array of command-line argument strings*/
char **envp) /*Array of environment variable strings*/
{
    int count;

    /*Display each command-line argument.*/
    printf("\nThe command-line arguments:\n");
    for(count = 0; count < argc; count++)
        printf(" argv[%d]  %s\n", count, argv[count]);

    /*Display each environment variable.*/
    printf("\nEnvironment variables:\n");
    while(*envp != NULL)
        printf("  %s\n", *(envp++));

    return 0;
}

```

```

[bodo@bakawali ~]$ gcc cmdline.c -o cmdline
[bodo@bakawali ~]$ ./cmdline

```

```

The command-line arguments:
argv[0]  ./cmdline

Environment variables:
HOSTNAME=bakawali

```

```
TERM=xterm
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT>::ffff:203.106.94.71 4136 22
SSH_TTY=/dev/pts/4
USER=bodo

LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33:01:cd=40;33:01:or=01;05
;37;41:mi=01;05;37;41:ex=00;32:*.cmd=00;32:*.exe=00;32:*.com=00;32:*.btm=00;32:*.bat=00;32
:*.sh=00;32:*.csh=00;32:*.tar=00;31:*.tgz=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.zip=
00;31:*.z=00;31:*.Z=00;31:*.gz=00;31:*.bz2=00;31:*.bz=00;31:*.tz=00;31:*.rpm=00;31:*.cpio=
00;31:*.jpg=00;35:*.gif=00;35:*.bmp=00;35:*.xbm=00;35:*.xpm=00;35:*.png=00;35:*.tif=00;35:
KDEDIR=/usr
MAIL=/var/spool/mail/bodo
PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/bodo/bin
INPUTRC=/etc/inputrc
PWD=/home/bodo
LANG=en_US.UTF-8
SSH_ASKPASS=/usr/libexec/openssh/gnome-ssh-askpass
SHLVL=1
HOME=/home/bodo
LOGNAME=bodo
SSH_CONNECTION>::ffff:203.106.94.71 4136 ::ffff:203.106.94.94 22
LESSOPEN=|/usr/bin/lesspipe.sh %s
G_BROKEN_FILENAMES=1
_=./cmdline
```

-----oO-----
---www.tenouk.com---

Further reading and digging:

1. [Check the best selling C and C++ books at Amazon.com.](#)
2. [Module G \(Story\)](#) and [Module M \(implementation\)](#) for Multibytes, Unicode characters and Localization.