

Assignment 1 – Command Line Arguments

Description:

This assignment is to write a C program that accepts arguments via the command line and then displays each of those arguments to the terminal along with how many arguments there are.

Approach / What I Did:

For this assignment, I was tasked with creating a C program that demonstrates how to access and print command-line arguments. Here's a breakdown of the steps I took and my thought process while working on the assignment:

1. Understanding the Task:

I started by carefully reading the assignment requirements. I needed to create a C program that accepts command-line arguments and then prints out information about these arguments, such as their count and values.

2. Analyzing the Requirements:

I noted that I needed to use the `main` function's parameters, `argumentCount` and `argumentValues`, to process the command-line arguments. This meant that I had to understand how these parameters work and how to utilize them effectively.

3. Writing the Program Outline:

I began by setting up the basic structure of the C program. This included including the necessary header file (`<stdio.h>`) and defining the `main` function with its expected parameters.

4. Printing Total Argument Count:

To display the total number of command-line arguments, I used the `argumentCount` parameter, which represents the count of arguments passed to the program. I inserted a `printf` statement to print this information.

5. Looping Through Arguments:

To process and print individual command-line arguments, I used a loop. I initialized an integer variable `i` to serve as a counter for the loop. The loop would run from 0 to `argumentCount` to cover all the arguments.

6. Printing Argument Details:

Within the loop, I used `argumentValues[i]` to access each individual argument's value. I added `printf` statements to display both the index of the argument (formatted with leading zeros) and the argument value itself.

7. Ensuring Proper Execution:

To ensure that the program runs successfully, I added a ``return 0;`` statement at the end of the ``main`` function. This indicates that the program executed without any errors.

8. Testing and Refining:

I tested the program by compiling and running it with various command-line arguments. I made sure that the displayed information matched my expectations and the requirements of the assignment.

9. Commenting and Documentation:

After verifying that the program worked as intended, I added comments throughout the code to explain each step and parameter.

10. Final Review and Submission:

Finally, I reviewed the code, comments, and the assignment requirements to ensure I hadn't missed anything. Once satisfied, I considered the assignment complete and ready for submission.

Throughout the process, I focused on breaking down the task into manageable steps and ensuring that each step aligned with the assignment's requirements. This approach helped me create a functional and well-documented program.

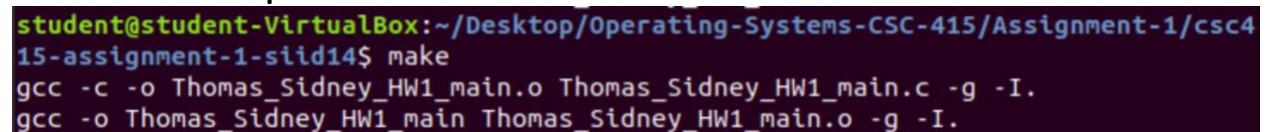
Issues and Resolutions:

Initially, I struggled to outline a method for approaching the assignment in my `.c` file. Seeking help from friends, they directed me to valuable C programming resources that clarified the language's concepts.

After coding the assignment on my MacBook, I faced the challenge of adapting it to a Linux environment within a Virtual Machine. Consulting my friends again, they guided me through the necessary command lines and steps to compile and run the program seamlessly in the VM.

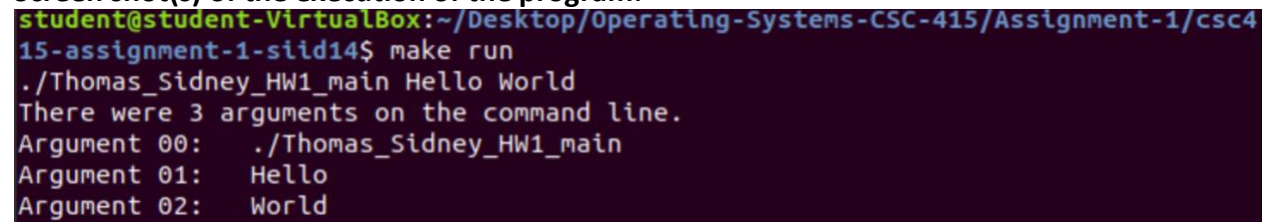
Analysis: Not required.

Screen shot of compilation:



```
student@student-VirtualBox:~/Desktop/Operating-Systems-CSC-415/Assignment-1/csc415-assignment-1-siid14$ make
gcc -c -o Thomas_Sidney_HW1_main.o Thomas_Sidney_HW1_main.c -g -I.
gcc -o Thomas_Sidney_HW1_main Thomas_Sidney_HW1_main.o -g -I.
```

Screen shot(s) of the execution of the program:

A terminal window screenshot with a dark purple background and light green text. The prompt is 'student@student-VirtualBox:~/Desktop/Operating-Systems-CSC-415/Assignment-1/csc415-assignment-1-siid14\$'. The user enters 'make run'. The program outputs 'Hello World' and then 'There were 3 arguments on the command line.' followed by three lines of argument details: 'Argument 00: ./Thomas_Sidney_HW1_main', 'Argument 01: Hello', and 'Argument 02: World'.

```
student@student-VirtualBox:~/Desktop/Operating-Systems-CSC-415/Assignment-1/csc415-assignment-1-siid14$ make run
./Thomas_Sidney_HW1_main Hello World
There were 3 arguments on the command line.
Argument 00:  ./Thomas_Sidney_HW1_main
Argument 01:  Hello
Argument 02:  World
```