CS 321 – Operating Systems
Computer Science Dep.
Assiut University

Chapter 2 - Sheet

Q1: Write a program in C that prints “Hello World”.

Here is how your program should behave:

```
$ ./Hello
Hello World
```

Q2: Create a C program that prints the count of arguments passed from the command line when the program is executed. Use the `argc` argument in the `main` function declaration to achieve this.

Here are some examples of how the program should behave:

```
~$ ./arg_count A B C
argc = 4
~$ ./arg_count A B C D
argc = 5
~$ ./arg_count A B
argc = 3
~$ ./arg_count
argc = 1
```

Q3: Develop a C program that prints the arguments passed from the command line when the program is executed. Display each argument on a new line. If no arguments are sent, print "no arguments." Utilize the `argv` parameter in the `main` function declaration.

Examples of the program's execution:

```
~$ ./arg_values A B C
A
B
C
~$ ./arg_values A B C D
A
B
C
D
~$ ./arg_values A B
A
B
~$ ./arg_values
<No arguments>
```

Q4: Write a program in C that takes any number of arguments and prints out the count of arguments and their values. If no input is provided, the program should print out its own name.

Here are some examples of how the program should behave:

```
~$ ./print x 2 1 5
Count = 4
Values: [x, 2, 1, 5]
~$ ./print ahmed mohammed 1 2 X
Count = 5
Values: [ahmed, mohammed, 1, 2, X]
~$ ./print
Program Name: ./print
```

Q5: Develop a C program that accepts only one argument sent from the command line when the program is executed. If more or less than one argument is provided (i.e., two or more, or none), the program should display "You should input one argument" and exit. However, if exactly one argument is sent, then the program should print "Great!". Utilize the `exit()` function included in `stdlib.h` library.

Examples of the program's execution:

```
~$ ./only_one A B C
You should input one argument.
~$ ./only_one
You should input one argument.
~$ ./only_one A
Great!
```

Q6: Create a C program that emulates the "echo" terminal command. The program should accept only one argument sent from the command line when the program is executed and print that argument on a new line. If more or less than one argument is provided (i.e., two or more, or none), the program should display "You should input one argument" and exit.

Examples of the program's execution:

```
~$ ./echo A B C
You should input one argument.
~$ ./echo
You should input one argument.
~$ ./echo Test
Test
~$ ./echo "Test with spaces"
Test with spaces
```

Q7: Write a program in C that takes two *integer* arguments from the command line and prints out their sum. The program should give an error message if the user inputs less or more than two numbers. Utilize the `atoi()` function included in `stdlib.h` library.

Here are some examples of how the program should behave:

```
$ ./sum 1 2
Sum = 3
$ ./sum 4 7
Sum = 11
$ ./sum 1
You should input two numbers.
$ ./sum 1 2 3
You should input two numbers.
```

Q8: Develop a C program that accepts any number of floating-point arguments from the command line and prints out their sum. If no arguments are sent, the program should print "sum = 0.0". Utilize the `atof()` function included in `stdlib.h` library.

Here are some examples of how the program should behave:

```
~$ ./sum_floats 1.2 1.5
Sum = 2.7
~$ ./sum_floats 5.2
Sum = 5.2
~$ ./sum_floats 1.2 1.5 3.4
Sum = 6.1
~$ ./sum_floats
Sum = 0.0
```

Q9: Write a program in C that accepts two integer arguments from the command line, as well as an operation character 's' for addition and 'm' for multiplication and computes the result of the operation between the two integer arguments. The program should give an error message if the user inputs less or more than three arguments or if the operation character is anything other than 's' or 'm'. (Hint: pay attention to $argv$ datatype.)

Here are some examples of how the program should behave:

```
~$ ./op s 1 2
Sum = 3
~$ ./op m 2 4
Mul = 8
~$ ./op x 1 2
Invalid operation
~$ ./op s 1
You should provide 3 arguments <op> <num1> <num2>
~$ ./op m 1 2 4
You should provide 3 arguments <op> <num1> <num2>
```

Q10: Create a C program that generates a random number between 0 and 100. The program should allow the user to input an unlimited number of guesses until they guess the correct number. While the user is guessing, the program should provide clues such as "too high" or "too low" to guide them towards the correct number. Once the user guesses correctly, the program should print a congratulatory message, along with the number of guesses it took the user to arrive at the correct answer. Utilize the `scanf()` function included in `stdio.h` library, `rand()` included in `stdlib.h`, and `stdbool.h` library to use Boolean values in C.

```
~$ ./guess
Input your guess: 40
Too Low. Try Again.
Input your guess: 90
Too Low. Try Again.
Input your guess: 95
Too High. Try Again.
Input your guess: 94
Too High. Try Again.
Input your guess: 91
That's right!
You guessed it right after 5 Tries.
```

Learning Objectives:

- **Q1: Create a C program using the text editor `nano`, then compile and execute it using the `gcc` compiler.**
- **Q2: Comprehend the purpose of the `argc` argument in the main function and its correlation with the input arguments provided through the command line.**
- **Q3: Grasp the significance of the `argv` argument in the main function, which grants access to the command-line arguments passed to the program.**
- **Q4: Understand the relationship between `argc` and `argv`, enabling access to the values of the arguments, including the program name itself.**
- **Q5: Recognize how to utilize `argc` to restrict the number of input arguments your program can accept.**
- **Q6: Gain insight into the fact that both the commands typed in the terminal (e.g., *echo*), and programs like `./echo` – which are code you wrote – are considered "programs," with one being part of the operating system and the other being your own code.**
- **Q7-Q8: Understand that the values within the `argv` array can be processed and used for other tasks, employing data types beyond just strings.**
- **Q9: Learn how to emulate "flags" similar to those used with other commands, such as the "-o" flag used with the gcc compiler to specify the output file name.**
- **Q10: Differentiate between using `argv` to capture user input and using the `scanf()` function, recognizing the significance and applicability of each approach.**

Helping references:

- C User Input (w3schools.com)
- C Booleans (w3schools.com)
- C program to generate random numbers within a range (includehelp.com)