CPE 460 Laboratory 2: String Manipulation

Department of Computer Engineering
Yarmoukd University

Spring 2017

1 Review

Write a C program `sum.c` that sum of the user-provided numbers through the command line arguments to the program. *Hint:* You can convert a string representing integer to its integer value using the `atoi` function call. Refer to `man atoi` for details.
Solution (sum.c):

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char** argv) {
    int i = 0;
    int sum = 0;
    if(argc == 1){
        printf("No numbers entered\n", name);
    } else{
        for(int i = 0; i < argc; i++){
            sum += atoi(argv[i]);
        }
        printf("Sum = %d\n", sum);
    }
    return 0;
}
```

To compile and build the source code:

```
$ gcc -o sum sum.c
```

To run the program with any arguments, type:

```
$ ./sum 1 2 3 4 5 6 7 8 9 10
$ Sum = 55
```

2 Purpose

In this lab, you will become familiar with: (1) standarad I/O function calls: `printf` and `scanf`, and (2) string manipulation function calls such as: `strcpy`, `strcat`, `strcmp`, and `strlen`.

3 Standard I/O Function Calls

3.1 `printf` Function Call

```
int printf(const char *Format, ...);
```

`printf` takes a formatting string `Format` and a variable number of extra arguments, determined by the formatting string, as indicated by the `...` notation. The keyword `const` means that `printf` cannot change the string `Format`.

The formatting strings consist of plain text, the special character `\n` that prints a newline, and directives of the form `%s` and `%d` indicates that `printf` will be looking for a corresponding variable in the argument list to insert into the output. A non-exhaustive list of directives is given here:

- `%d` Print an int. Corresponding argument should be an int.
- `%f` Print a float.
- `%c` Print a character according to the ASCII table. Argument should be char.
- `%s` Print a string. Argument should be a pointer to a char (first element of a string).
- `%X` Print an unsigned int as a hex number.

Please refer to `printf` man page for further info through the command `man printf`.

3.2 `scanf` Function Call

```
int scanf(const char *Format, ...);
```

Arguments to `scanf` consist of a formatting string and pointers to variables where the input should be stored. Typically the formatting string consists of directives like `%d`, `%f`, etc., separated by whitespace. The directives are similar to those for `printf`. For each directive, `scanf` expects to see a **pointer** to a variable of that type in the argument list. A very common mistake is the following:

```
int i;
scanf("%d",i); // WRONG! We need a pointer to the variable.
scanf("%d",&i); // RIGHT.

char message[20];
scanf("%s", &message); // WRONG! message is already a pointer to the start of array.
scanf("%s", message); // RIGHT.
```

The pointer allows `scanf` to put the input into the right place in memory. Please refer to `scanf` man page for further info.

4 String Manipulation

String manipulation is important and will use them later in the upcoming labs.

4.1 String Copy

```
char* strcpy(char *destination, const char *source);
```

Given two strings, `char destination[100]`, and `source[100]`, we cannot simply copy one to the other using the assignment `destination = source`. Instead, we use `strcpy(destination,source)`, which copies the string `source` (until reaching the string terminator character, integer value 0) to `destination`. The string `destination` must have enough memory allocated to hold the source string. There is another variable of `strncpy`:

```
char *strncpy(char *dest, const char *src, size_t n)
```

`strncpy` copies up to `n` characters from the string pointed to, by `src` to `dest`. In a case where the length of `src` is less than that of `n`, the remainder of dest will be padded with `null` bytes.

4.2 String Concatenation

```
char *strcat(char *dest, const char *src)
```

`strcat` appends the string pointed to by `src` to the end of the string pointed to by `dest`.

```
char *strncat(char *dest, const char *src, size_t n)
```

`strncar` appends the string pointed to by `src` to the end of the string pointed to by `dest` up to `n` characters long.

4.3 String Comparison

```
int strcmp(const char *str1, const char *str2)
```

`strcmp` compares the string pointed to, by `str1` to the string pointed to by `str2`. The return values that are as follows:

- If return value < 0 then it indicates `str1` is less than `str2`.
- If return value > 0 then it indicates `str2` is less than `str1`.
- If return value $= 0$ then it indicates `str1` is equal to `str2`.

```
int strncmp(const char *str1, const char *str2, size_t n)
```

`strncmp` compares at most the first `n` bytes of `str1` and `str2`. The return values that are as follows:

- If return value < 0 then it indicates `str1` is less than `str2`.
- If return value > 0 then it indicates `str2` is less than `str1`.
- If return value $= 0$ then it indicates `str1` is equal to `str2`.

4.4 String Length

```
size_t strlen(const char *str)
```

`strlen` computes the length of the string `str` up to, but not including the terminating `null` character.

5 Exercise

Write a program call it `str-op.c` to prompt the user to enter a string then prompt the user to perform one the following operations on that string:

```
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
```

Below is a sample output:

```
$ ./string
Enter  a string: I love makmoora
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 1
String Length: 15

$ ./string
Enter  a string: I love makmoora!
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 2
Enter the target string: and mansaf!
Result: Ilove makmoora and mansaf!

$ ./string
Enter  a string: I love makmoora!
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 3
Copied String: I love makmoora!

$ ./string
Enter  a string: I love makmoora
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 4
The Upper Case of String is: I LOVE MAKMOORA

$ ./string
Enter  a string: I love makmoora!
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 5
The Lower Case of String is: i love makmoora
```

```
$ ./string
Enter  a string: I love makmoora!
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 6
Enter the string you wish to compare with the previous string: I love mansaf!
String Comparison Result: -1

$ ./string
Enter  a string: I love makmoora!
Enter the operation you wish to perform on the string:
1. String Length, 2. String Concatination, 3. String Copy, 4. String Upper Case
5. String Lower Case, 6. String Comparison
Your choice> 20
You have not entered a valid option.
```