

CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Homework 4

Name (Print): _____ RHIT Username: _____

Put your answers in the spaces reserved below the questions. **Do not** write your answers on **separate sheets of paper**.

1. (10 points) Read the following code and answer the questions.

```
1 void mystery0(int a, int b, int c) {  
2     a = b + c;  
3 }  
4 void mystery1(int* a, int* b, int* c) {  
5     *a = *b + *c;  
6 }  
7 int mystery2(int* d, int e, int f) {  
8     int i = 0;  
9     int j = 0;  
10    for (i = 0; i < f; i = i + e) {  
11        j = j + d[i];  
12    }  
13    return j;  
14 }
```

- (a) What is the value of x after the following code segment executes?

```
1 int x = 0;  
2 int y = 1;  
3 int z = 20;  
4 mystery0(x, y, z);
```

x =

- (b) What is the value of x after the following code segment executes?

```
1 int x = 0;  
2 int y = 1;  
3 int z = 20;  
4 mystery1(&x, &y, &z);
```

x =

- (c) Explain why `mystery0` and `mystery1` behave differently.
- (d) If `arr` is an array containing the following values: {1, 2, 3, 4, 5, 6, 7, 8, 9}, what does `mystery2(arr, 3, 9)` return?
- `mystery2(arr, 3, 9)` returns
- (e) What would be more descriptive names for `mystery2` and its parameters?

2. (8 points) Consider the C and corresponding assembly code.

```

1  void increment(int* xp){
2      *xp = *xp + 1;
3  }
4  int my_func(int x){
5      increment(&x);
6      return x+1;
7  }
8  int main() {
9      int x = 56;
10     x = my_func(x);
11     int y = 78;
12 }
```

```

1  increment:
2      sub sp, sp, 16
3      str x0, [sp, 8]
4      ldr x0, [sp, 8]
5      ldr w0, [x0]
6      add w1, w0, 1
7      ldr x0, [sp, 8]
8      str w1, [x0]
9      add sp, sp, 16
10 my_func:
11     str x30, [sp, -16]!
12     str w0, [sp, 12]
13     add x0, sp, 12
14     bl  increment
15     ldr w0, [sp, 12]
16     add w0, w0, 1
17     ldr x30, [sp], 16
18     ret
19 main:
20     str x30, [sp, -32]!
21     mov w0, 56
22     str w0, [sp, 28]
23     ldr w0, [sp, 28]
24     bl  my_func
25     str w0, [sp, 28]
26     mov w0, 78
27     str w0, [sp, 24]
28     mov w0, 0
29     ldr x30, [sp], 32
30     ret
```

For the questions below, assume the computer is a 64-bit machine.

(a) How many bytes are **allocated** on the stack?

- In `main`'s stack frame?

- In `my_func`'s stack frame?

- In `increment`'s stack frame?

(b) How many bytes are **used** on the stack?

- In `main`'s stack frame?

- In `my_func`'s stack frame?

- In `increment`'s stack frame?

(c) How many bytes are **unused** on the stack? [Hint: (a) minus (b)]

- In `main`'s stack frame?

- In `my_func`'s stack frame?

- In `increment`'s stack frame?

3. (10 points) The following code is compiled into a program called `a.out` and then it is executed with the command line:

`./a.out PINE APPLE`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(int argc, char** argv) {
6     char* p[3]; //Note the type of p
7     int len = strlen(argv[0]);
8     for (int i = 0; i < argc; i++) {
9         if (strlen(argv[i]) < len) {
10             len = strlen(argv[i]);
11         }
12         p[i] = argv[i];
13     }
14     for (int i = 0; i < len; i++) {
15         for (int j = 0; j < argc; j++) {
16             printf("%c", *(p[j])); // Note that there is no '\n'
17             p[j]++;                // p[j] = p[j] + 1
18         }
19         printf("\n");
20     }
21
22     // printf("%c - %s\n", *(p[0]), p[0]);
23
24     return p[0] - argv[0];
25 }
```

Note: line 22 is commented out (will not execute), and `%c` tells `printf` to print a char, while `%s` is for a string. Based on the given information, answer the following questions.

- (a) How many times will line 19 (`printf("\n");`) be executed?
- (b) What will be printed when this code executes?
- (c) What would be printed by line 22 if it was not commented out?
- (d) What value does the function return?

4. Coding part (20 points): On your Linux, pull changes to your Git repository.

Create a program called `add`. This program take exactly two input arguments (in addition to the program name). The two arguments must be two decimal numbers. As the result, the program will add these two numbers together and print out the addition results.

To make it easy to implement, we can assume that the two numbers can only be in the range of 0 to 99 inclusively. Namely, they can only be at most 2-digit non-negative numbers.

NOTE: You are not allowed to call function `atoi()` in this part.

Your program must satisfy the following specifications:

- ☐ You must implement this in a new file called `add.c` in your `homework4` directory.
- ☐ Remember that the input arguments of the `main` function are defined as follows

```
int main(int argc, char** argv){
    ...
}
```

- ☐ The provided `Makefile` already has a rule that will make `add` from `add.c`. You can type `make add` to compile it.
- ☐ When executed from the command line with the proper arguments, it will print the correct addition answer on the screen. For example:

```
user@linux:~$ ./add 29 18
Do you know that 29 + 18 = 47?
```

Note: there is a '\n' after this line.

- ☐ If the wrong number of arguments are given, print the below error message:

```
user@linux:~$ ./add
Usage: ./add <num1> <num2>
user@linux:~$ ./add 12 34 56
Usage: ./add <num1> <num2>
```

- ☐ If the number of arguments are correct but the input arguments are not numbers from 0-99, print the below error message:

```
user@linux:~$ ./add 100 0
Error: 100 is not a number from 0 to 99.
user@linux:~$ ./add one 2
Error: one is not a number from 0 to 99.
```

- ☐ To test your program, you can first manually run the program with some inputs, e.g., `./add 23 43`. If the output looks good to you, you can use the provided test script to do a more thouroughly test by typing `./test_add.sh`

When finished, submit your `add.c` to **Gradescope**.