Fall ’18 CIS 314 Assignment 3 – 100/100 points – Due Wednesday, 10/17, 11:59 PM

Please submit individual source files for coding exercises (see naming conventions below). Your code and answers need to be documented to the point that the graders can understand your thought process. Full credit will not be awarded if sufficient work is not shown.

1. [80] Write a C program that defines the following struct:

```
struct IntArray {
    int length;
    int *dataPtr;
};
```

and the following functions:

- (10) struct IntArray* mallocIntArray(int length): allocates, initializes, and returns a pointer to a new *struct IntArray*. Hint: see example from class – also, you’ll need two malloc calls, one for the instance and one for the instance’s dataPtr (a pointer to an int array of size *length*).
- (10) void freeIntArray(struct IntArray *arrayPtr): frees the instance’s dataPtr and frees the instance.
- (10) void readIntArray(struct IntArray *array): prompts and reads ints from the user to fill the array (i.e., read one int for each array index). Your program should print an error message and prompt again if the user enters a value that cannot be parsed as an int. Hint: I recommend using fgets and strtol – you can Google for examples of these (cite your sources) and we’ll cover them in labs.
- (15) void swap(int *xp, int *yp): swaps the int values stored at the *xp* and *yp* pointers.
- (15) void sortIntArray(struct IntArray *array): sorts the input array in ascending order using Selection Sort (Google it, cite your sources) by repeatedly calling your *swap* function as appropriate.
- (10) void printIntArray(struct IntArray *array): prints the array (e.g., “[1, 3, 5, 7]”).
- (10) int main(): prompt the user to input a positive int length from the user (print an error message and prompt again if the input cannot be parsed as a positive int), call *mallocIntArray* to create an array, call *readIntArray* to prompt the user to fill the array, call *sortIntArray* to sort it, call *printArray* to print the resulting sorted array, then call *freeIntArray* to free the heap memory used by the array.

Name your source file 3-1.c.

Here is output from a sample run of the application (your output does not need to match exactly):

```
Enter length: cat
Invalid input
Enter length: 5
Enter int: 3
Enter int: puppy
Invalid input
Enter int: 7
Enter int: 8
Enter int: 2
[ 2, 3, 5, 7, 8 ]
```

2. [20] Consider the following C function:

long f(long a, long b, long c);

When compiled, the resulting x86-64 code for the function body is:

```
subq  %rdx, %rsi
imulq %rsi, %rdi
salq  $63, %rsi
sarq  $63, %rsi
movq  %rdi, %rax
xorq  %rsi, %rax
```

Assume that *a* is in %rdi, *b* is in %rsi, *c* in %rdx.

Your task is to reverse engineer the C code; specifically, to write a C implementation for the decode function that is functionally equivalent to the compiled x86-64 code above.

Here are some test runs:

```
decode(1, 2, 4): -2
decode(3, 5, 7): -6
decode(10, 20, 30): -100
```

Also write a main() function to test your function.

Hint: try compiling your C code to x86-64 on Arch using *gcc -Og -S 3-2.c* as you work. Note that the output assembly does not need to match the x86-64 code above exactly – it just needs to be functionally equivalent.

Name your source file 3-2.c

Zip the source files and solution document (if applicable), name the .zip file <Your Full Name>Assignment3.zip (e.g., EricWillsAssignment3.zip), and upload the .zip file to Canvas (see Assignments section for submission link).