

CSCI 2271 Computer Systems
Assignment 5: Dynamic Memory Allocation
Due Friday, March 4

1. (This is a warm-up for problem 2.) Write a program, named *intTail.c*, that takes a bunch of positive integers as input, and prints the last few of them. You indicate the end of input by entering a -1. You use the command line to specify how many integers to print; if you don't specify a number, then the program should print the last 10 numbers.

For example, here is a screenshot of my program in action:

```
adminuser@adminuser-VirtualBox ~/HW/HW5 $ intTail
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 -1
Here are the last 10 numbers: 6 7 8 9 10 11 12 13 14 15
adminuser@adminuser-VirtualBox ~/HW/HW5 $ intTail 4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 -1
Here are the last 4 numbers: 12 13 14 15
adminuser@adminuser-VirtualBox ~/HW/HW5 $ intTail 12
1 2 3 4 -1
There are only 4 numbers: 1 2 3 4
```

Note that I ran my *intTail* program three times. The first two times, I entered the numbers from 1 to 15 as input. The first time I did not specify how many to print, so it printed the last 10 numbers. The second time I asked it to print the last 4 numbers. The third time I ran the program, I asked it to print more numbers than were in the input. In this case, the program should print the entire input.

The best way to solve this problem is to use a "circular array" implementation. Allocate an array of size n , where n is the number of values to be printed. (Since the value of n could be anything, you will need to use *malloc* to allocate the array dynamically.) Each time you read each integer, you add it to the next slot of the array; when you get to the end of the array, you circle back to the beginning. When you get to the end of the input, the array will contain the last n values.

Be sure to call the *free* function to deallocate space allocated by *malloc*.

2. Linux has a command called *tail*, which returns the last few lines of the standard input. You are to write a program (called *mytail*) that behaves similarly. In particular, if the program is called with no command arguments, it returns the last 10 lines of its input. Otherwise, it must be called with one command argument, as in problem 1. In this case, the program returns the last n lines of its input. Here is a screenshot of my program in action:

```

adminuser@adminuser-VirtualBox ~/HW/HW5 $ myTail < ../HW1/calendar.c
    for (i = 1; i <= n; i++) {
        printf("%3d", i);
        if ((startDay + i - 1) % 7 == 0)
            printf("\n");
    }

    if ((startDay + n - 1) % 7 != 0)
        printf("\n");
    printf("\n");
}
adminuser@adminuser-VirtualBox ~/HW/HW5 $ myTail 4 < ../HW1/calendar.c
    if ((startDay + n - 1) % 7 != 0)
        printf("\n");
    printf("\n");
}

```

In this example, my program is reading from the *calendar.c* file that I wrote for HW 1.

The solution to this problem is nearly identical to that of problem 1. The difference is that instead of an array of ints, you need to malloc an array of (char *) pointers (which implies that the type of the array is actually (char **)).

To make things interesting, I want you write (and use) a function named *readaline* to read each line of input. The function is called like this:

```
char *line = readaline(80);
```

The argument to the *readaline* function is an integer denoting the maximum number of characters to read. The function calls *malloc* to allocate space for these characters, and passes this array to *fgets*. The function *fgets* will return NULL if it encounters end-of-file. In this case, *readaline* should free the allocated space and return NULL. Otherwise it should return a pointer to the space.

Note that *fgets* adds the newline character '\n' to the array, followed by the '\0' character. Your *readaline* function should remove the newline from the string.

You can assume that all lines are no more than 80 characters long (which means you can always call *readaline* with the value 80).

Note that your code will need to use *malloc* to allocate space for the array of pointers, as well as for each input line. Be sure to call the *free* function to deallocate these spaces appropriately.

When you are finished, submit your two files to Canvas.