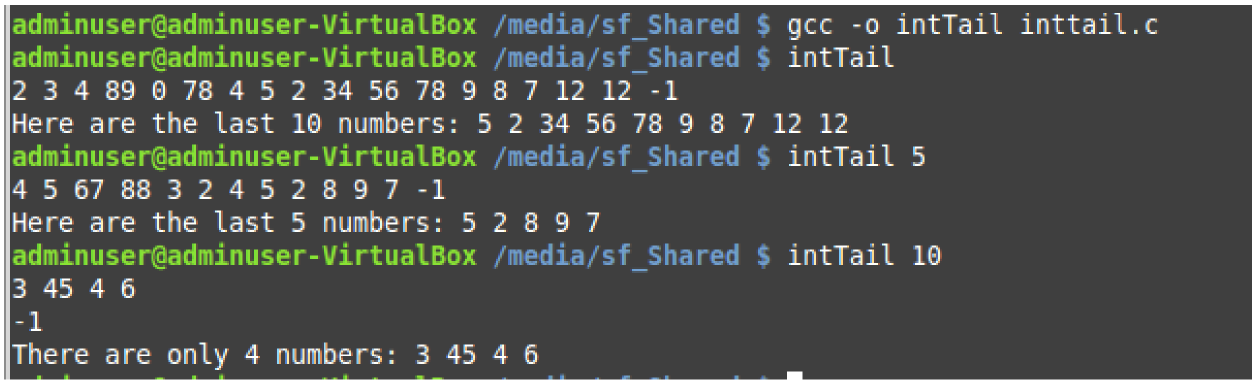
CSCI 2271 Computer Systems

Assignment 4: Dynamic Memory Allocation

Due Friday, February 22

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.

Here is an example in action:



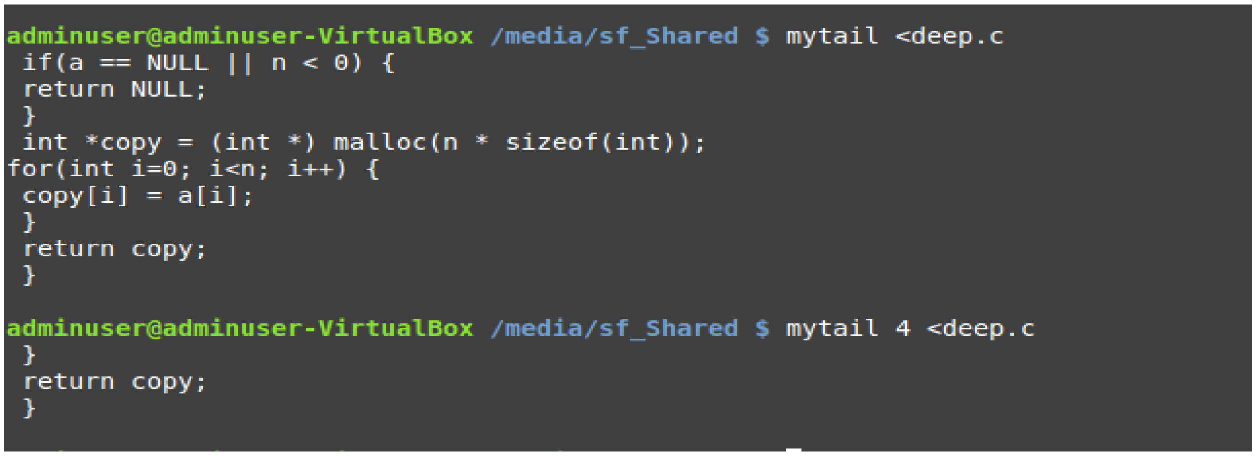
```
adminuser@adminuser-VirtualBox /media/sf_Shared $ gcc -o intTail inttail.c
adminuser@adminuser-VirtualBox /media/sf_Shared $ intTail
2 3 4 89 0 78 4 5 2 34 56 78 9 8 7 12 12 -1
Here are the last 10 numbers: 5 2 34 56 78 9 8 7 12 12
adminuser@adminuser-VirtualBox /media/sf_Shared $ intTail 5
4 5 67 88 3 2 4 5 2 8 9 7 -1
Here are the last 5 numbers: 5 2 8 9 7
adminuser@adminuser-VirtualBox /media/sf_Shared $ intTail 10
3 45 4 6
-1
There are only 4 numbers: 3 45 4 6
```

Note that I ran my intTail program three times. 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 5 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.

To solve this problem, you can 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 an example in action:



```
adminuser@adminuser-VirtualBox /media/sf_Shared $ mytail <deep.c
 if(a == NULL || n < 0) {
 return NULL;
 }
 int *copy = (int *) malloc(n * sizeof(int));
for(int i=0; i<n; i++) {
 copy[i] = a[i];
 }
 return copy;
 }

adminuser@adminuser-VirtualBox /media/sf_Shared $ mytail 4 <deep.c
 }
 return copy;
 }
```

In this example, my program is reading from the *deep.c* file that we discussed on Feb 7 (available in Demos-Feb7).
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.

You must 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 (zip them) to Canvas.