CSCI 1730 Project 2

Understanding the Heap, Dynamic Memory Management, and Pointers

Learning Objectives

- Design and implement programs that use basic programming and flow of control structures such as (but not limited to) variables, operators, expressions, decisions statements, loops, pointers, and functions (prototyping, calling, and passing arguments to functions).
- Demonstrate knowledge of simple data structures (arrays) by writing out their memory maps and tracing through the output of source code.
- Demonstrate knowledge of pointers and arrays by tracing through the output of source code containing pointers to 1D arrays.
- Design and implement a program that uses pointers and dynamic memory allocation and deallocation.
- Design and implement programs based on output from several examples.
- Use the command line in a Unix environment to run processes and to pipe or redirect input or output.
- Use valgrind to find memory leaks in programs. Implement programs that use dynamic memory allocation and deallocation and have no memory leaks.
- Understand the limitations of using arrays.

Problem / Exercise

The purpose of this assignment is to write a C program that maps out its dynamic memory usage as the program executes to solve a simple problem of computing the average grade of a list of grades and determining which grades are >= or < the average. This program's output will consist of a memory map printing out what's going on in the heap as the program executes, which will further your understanding of dynamic memory management and pointers.

The C program that you will write will process grades from standard input (inputted by a user or redirected from an input file) and store them on the heap. The number of input grades is unknown and your program should store grades on the heap until a sentinel value, a negative input, is encountered. Thus, your program should use dynamic memory to store this stream of inputs of unknown length until the sentinel value is encountered. The use of the heap to store grades is required for this program, and you will receive a grade of zero if you do not use the heap to store grades.

Grades will be inputted as doubles that are greater than or equal to zero, and the end of the input will be signaled by a double value that is less than zero, which is a sentinel value. The sentinel value should not be stored on the heap, and the sentinel value should not be used to calculate the average grade. If the first input is less than zero, then the program's output should be the same as shown in example section, and no heap memory should be used since no grades were inputted. If the first input is greater than or equal to zero, then the program should allocate space for five grades (5 doubles) to the heap and store the first grade on the heap. The program should continue to store input grades on the heap that are greater than zero until a negative value is inputted or the five places on the heap are full. If the five places on the heap are full, then the program should allocate space for five additional grades, copy over all grades from the old part of the heap to the new, larger part of the heap, free up the memory no longer being used on the heap, and the I/O of the program must be consistent with our examples in the examples section. Once the input has ended, the program should compute the average and print out which grades were >= or < the average as shown in the examples. Also, the program should correctly print out the total heap usage, the total heap usage much match our examples, it must be consist with the output from valgrind. Study the examples in the examples section to understand the problem before you start to code. This section only gives general parts of the algorithm you'll need to implement. You'll need to reverse engineer the examples shown in the examples section to get your program's I/O to be consistent with all of our examples.

Project Requirements

Your program must adhere to all requirements stated below.

1. Your program must be written in C, all of its source code should be in a file named proj2.c, and all files for this assignment should be in a directory named proj2.
2. Your program may only contain the following include statements:
   (a) #include <stdio.h>
   (b) #include <stdlib.h>
   (c) #include <stdbool.h>

   The use of any other include statements than those mentioned above will result in a grade of zero on this assignment.
3. Your program may only use the following functions that are part of the C language:
   (a) printf
   (b) scanf
   (c) malloc
   (d) free

   All other functions you must design and implement yourself. The use of any other built-in functions than those mentioned above will result in a grade of zero on this assignment. The four functions above are necessary for this assignment.
4. Your program must store all grades (all inputs >= 0) on the heap using arrays of the fixed sizes as shown in the examples (sizes of 40, 80, 120, ...).
5. Do not ask (or prompt) the user for how many grades they would like to input. The input format must match our examples, and this is designed to simulate real world data where the length of the input may not be known at the start of the program.
6. Your source code cannot contain a [or a], which is common syntax to use for arrays. This is a requirement because we want students in this course to get more experience with pointer syntax and pointer arithmetic. If a single [or a single] is found in your source code, then you will receive a zero on this assignment. Use pointer notation and pointer arithmetic instead of the square bracket notation for this assignment.
7. Your program's I/O must match the examples except the memory addresses and valgrind process ids (pids) will typically be different each time the program is run, and this is okay. However, the relative memory addresses should be consistent (consecutive grades should be in addresses that are 8 bytes apart).
8. Your program's total heap usage output must be consistent with valgrind's output for total heap usage when your program is run in valgrind. You must run your program with valgrind for many examples to ensure this is consistent and that no memory leaks are present in your program when it executes. Substantial points will be lost if your program has a memory leak or its total heap usage is not consistent with valgrind.
9. Your program must compile with all targets, dependencies, and commands given in the Makefile, which is on eLC for this project, and your program cannot contain any compiler errors or warnings when compiled using the provided Makefile.
10. All instructions written in this assignment and stated by the lecture instructor must be followed. Failure to follow any instructions given for this assignment may result in a loss of points or failing this assignment.
11. After you finish the program and submit it, consider the following questions. What are the limitations of using fixed length arrays for problems dealing with variable length input? What are the advantages of using fixed sized arrays for problems with variable length inputs? What other data structures are useful for problems dealing with variable length inputs?

C Program

Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Project 2. Submit only the following files.

1. All source files required for this assignment: proj2.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own, unmodified copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

Grading: 30 points

If your program does not compile on odin using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero for this assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	30 points
README.txt file missing or incorrect in some way	-3 points
Not submitting to the correct Assignment on eLC	-3 points
Late penalty for submitting 0 hours to 24 hours late	-6 points
One or more compiler warnings	-6 points
Not adhering to one or more coding style requirements	-6 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-6 points
Program does not use the heap to store grades	-30 points
Source code contains an unauthorized include statement	-30 points
Source code contains an unauthorized built-in function call	-30 points
Source code contains a single [or a single]	-30 points
Not submitting this assignment before its late period expires	-30 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the examples in this assignment) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.

Examples

Each example is a separate run of a correctly working program. Your program's I/O should match these examples except the memory addresses and valgrind process ids (pids) will typically be different for each run. However, the relative memory addresses should be consistent as aforementioned. input1.txt, output1.txt, input2.txt, and output2.txt are on eLC, and those files contain the full input and outputs for the large examples. Students must test, test, and retest their code with many examples and run each example in valgrind to ensure their programs are correct, contain no memory leaks, and that total heap summary outputted by their program is consistent with valgrind (please note that valgrind uses commas to separate thousands places, and we do not have to print commas to separate thousands places in our total heap usage, and this is shown in some of the examples).

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
After the last grade is entered, enter a negative value to end the list.
-2.5
The average of 0 grades is 0.000000.
total heap usage: 0 allocs, 0 frees, 0 bytes allocated
```

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
After the last grade is entered, enter a negative value to end the list.
97.5
Allocated 40 bytes to the heap at 0x8d8010.
Stored 97.500000 in the heap at 0x8d8010.
-12.5
The average of 1 grades is 97.500000.
1. The grade of 97.500000 is >= the average.
Freed 40 bytes from the heap at 0x8d8010.
total heap usage: 1 allocs, 1 frees, 40 bytes allocated
```

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
After the last grade is entered, enter a negative value to end the list.
85.4
Allocated 40 bytes to the heap at 0xb94010.
Stored 85.400000 in the heap at 0xb94010.
28.5
Stored 28.500000 in the heap at 0xb94018.
98.75
Stored 98.750000 in the heap at 0xb94020.
100.0
Stored 100.000000 in the heap at 0xb94028.
0
Stored 0.000000 in the heap at 0xb94030.
Stored 5 grades (40 bytes) to the heap at 0xb94010.
Heap at 0xb94010 is full.
Allocated 80 bytes to the heap at 0xb94040.
Copied 5 grades from 0xb94010 to 0xb94040.
Freed 40 bytes from the heap at 0xb94010.
-2
The average of 5 grades is 62.530000.
1. The grade of 85.400000 is >= the average.
```

```
2. The grade of 28.500000 is < the average.
3. The grade of 98.750000 is >= the average.
4. The grade of 100.000000 is >= the average.
5. The grade of 0.000000 is < the average.
Freed 80 bytes from the heap at 0xb94040.
total heap usage: 2 allocs, 2 frees, 120 bytes allocated
```

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
After the last grade is entered, enter a negative value to end the list.
78
Allocated 40 bytes to the heap at 0x1c82010.
Stored 78.000000 in the heap at 0x1c82010.
99
Stored 99.000000 in the heap at 0x1c82018.
100
Stored 100.000000 in the heap at 0x1c82020.
-1
The average of 3 grades is 92.333333.
1. The grade of 78.000000 is < the average.
2. The grade of 99.000000 is >= the average.
3. The grade of 100.000000 is >= the average.
Freed 40 bytes from the heap at 0x1c82010.
total heap usage: 1 allocs, 1 frees, 40 bytes allocated
```

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
After the last grade is entered, enter a negative value to end the list.
92.3
Allocated 40 bytes to the heap at 0x104c010.
Stored 92.300000 in the heap at 0x104c010.
88.7
Stored 88.700000 in the heap at 0x104c018.
56
Stored 56.000000 in the heap at 0x104c020.
78
Stored 78.000000 in the heap at 0x104c028.
45.7
Stored 45.700000 in the heap at 0x104c030.
Stored 5 grades (40 bytes) to the heap at 0x104c010.
Heap at 0x104c010 is full.
Allocated 80 bytes to the heap at 0x104c040.
Copied 5 grades from 0x104c010 to 0x104c040.
Freed 40 bytes from the heap at 0x104c010.
100.00
Stored 100.000000 in the heap at 0x104c068.
-5.6
The average of 6 grades is 76.783333.
1. The grade of 92.300000 is >= the average.
2. The grade of 88.700000 is >= the average.
3. The grade of 56.000000 is < the average.
4. The grade of 78.000000 is >= the average.
5. The grade of 45.700000 is < the average.
6. The grade of 100.000000 is >= the average.
Freed 80 bytes from the heap at 0x104c040.
total heap usage: 2 allocs, 2 frees, 120 bytes allocated
```

```
./proj2.out
Enter a list of grades below where each grade is separated by a newline character.
```

```
After the last grade is entered, enter a negative value to end the list.
99.2
Allocated 40 bytes to the heap at 0x85a010.
Stored 99.200000 in the heap at 0x85a010.
88.3
Stored 88.300000 in the heap at 0x85a018.
45.3
Stored 45.300000 in the heap at 0x85a020.
0
Stored 0.000000 in the heap at 0x85a028.
0
Stored 0.000000 in the heap at 0x85a030.
Stored 5 grades (40 bytes) to the heap at 0x85a010.
Heap at 0x85a010 is full.
Allocated 80 bytes to the heap at 0x85a040.
Copied 5 grades from 0x85a010 to 0x85a040.
Freed 40 bytes from the heap at 0x85a010.
72.5
Stored 72.500000 in the heap at 0x85a068.
98.3
Stored 98.300000 in the heap at 0x85a070.
44.5
Stored 44.500000 in the heap at 0x85a078.
0
Stored 0.000000 in the heap at 0x85a080.
0
Stored 0.000000 in the heap at 0x85a088.
Stored 10 grades (80 bytes) to the heap at 0x85a040.
Heap at 0x85a040 is full.
Allocated 120 bytes to the heap at 0x85a0a0.
Copied 10 grades from 0x85a040 to 0x85a0a0.
Freed 80 bytes from the heap at 0x85a040.
43.7
Stored 43.700000 in the heap at 0x85a0f0.
56.3
Stored 56.300000 in the heap at 0x85a0f8.
0
Stored 0.000000 in the heap at 0x85a100.
-2
The average of 13 grades is 42.161538.
1. The grade of 99.200000 is >= the average.
2. The grade of 88.300000 is >= the average.
3. The grade of 45.300000 is >= the average.
4. The grade of 0.000000 is < the average.
5. The grade of 0.000000 is < the average.
6. The grade of 72.500000 is >= the average.
7. The grade of 98.300000 is >= the average.
8. The grade of 44.500000 is >= the average.
9. The grade of 0.000000 is < the average.
10. The grade of 0.000000 is < the average.
11. The grade of 43.700000 is >= the average.
12. The grade of 56.300000 is >= the average.
13. The grade of 0.000000 is < the average.
Freed 120 bytes from the heap at 0x85a0a0.
total heap usage: 3 allocs, 3 frees, 240 bytes allocated
```

```
./proj2.out < input1.txt > output1.txt; tail output1.txt;
```

```
98. The grade of 90.758400 is >= the average.
99. The grade of 34.443700 is < the average.
100. The grade of 73.924500 is >= the average.
101. The grade of 79.194500 is >= the average.
102. The grade of 3.167200 is < the average.
103. The grade of 56.274300 is >= the average.
104. The grade of 65.125000 is >= the average.
105. The grade of 40.944500 is < the average.
Freed 880 bytes from the heap at 0x1172360.
total heap usage: 22 allocs, 22 frees, 10120 bytes allocated

valgrind ./proj2.out < input1.txt |& grep 'total'
total heap usage: 22 allocs, 22 frees, 10120 bytes allocated
==21413==   total heap usage: 22 allocs, 22 frees, 10,120 bytes allocated

./proj2.out < input2.txt > output2.txt; tail output2.txt;
28102. The grade of 2.828200 is < the average.
28103. The grade of 56.698800 is >= the average.
28104. The grade of 38.142200 is < the average.
28105. The grade of 6.610100 is < the average.
28106. The grade of 24.604700 is < the average.
28107. The grade of 89.410300 is >= the average.
28108. The grade of 90.594200 is >= the average.
28109. The grade of 83.110400 is >= the average.
Freed 224880 bytes from the heap at 0x11c3010.
total heap usage: 5622 allocs, 5622 frees, 632250120 bytes allocated

valgrind ./proj2.out < input2.txt |& grep 'total'
total heap usage: 5622 allocs, 5622 frees, 632250120 bytes allocated
==23105==   total heap usage: 5,622 allocs, 5,622 frees, 632,250,120 bytes allocated
```