CS270 Recitation 11
"Dynamic Memory Allocation"

Goals

To understand how dynamic memory allocation works.

Static Memory Allocation involves memory allocated at compile time in stack or other data segments. This is used when the amount (size) of memory is fixed and is known during compile-time.

Dynamic Memory Allocation: memory is allocated during run-time in heap. This is used when the amount of memory needed is variable and can be allocated, deallocated or changed during run-time. Dynamic allocation uses using certain functions like malloc(), calloc(), realloc(), free( ) in C

Background

The function malloc ( ) returns a pointer to space of specified size or a NULL if the request cannot be satisfied.

E.g., int* p = (int *) malloc(sizeof(int));

Deallocation can be done using free (p) which releases the space pointed to by the pointer p
Which must be must be pointer to space previously allocated. If p is NULL, nothing happens.

void free(void *p)

The function void *realloc(void *p size_t size) attempts to resize the memory block pointed to by p that was previously allocated with a call to malloc or calloc.

The Recitation:

Make a subdirectory called R11 for the recitation and copy the following files into the directory:
r11.c

The program has some missing statements. You need to debug the program using gdb and add missing statements to make it work.

Compile using:

```
gcc -g -std=c99 -Wall -c r11.c -o r11.o
gcc -g -lm r11.o -o r11
```

Use gdb to examine the code using the steps below.

```
$ gdb r11 // start gdb debugger
```

- (gdb) set logging on   // enable logging to gdb.txt
- (gdb) break 16   // set breakpoint at line 16 in r11.c {line 16: termination = 0;}
- (gdb) break 21   // set breakpoint at line 21 in r11.c {line 21: ptr_int = malloc(max * max * sizeof(int)); /* call malloc() */}

- (gdb) run // run program

The program pauses at line 16. Check the value and address of max at this moment.

- (gdb) print max // print value of max
- (gdb) print &max // print address of max

Now continue the program until next breakpoint (line 21).

- (gdb) continue // continue to next breakpoint

Observe the value and address of max.

- (gdb) print max // print value of max
- (gdb) print &max // print address of max

Observe how scanf takes an input from the keyboard and stores it at a specific address.

Write GDB commands to check the address stored at the pointer 'ptr_int' and the value stored at the address pointed by 'ptr_int'.

Now step one line of code to execute line 21.

- (gdb) step // step one line

Check the address stored at ptr_int. Has it changed? What does ptr_int point to?

What is the amount of requested memory in the given code?

Let's display the allocated memory in the heap.

- (gdb) print *ptr_int@('size of dynamic array')

Set new breakpoint at line 24 and continue the execution of program.

The program will pause at line 24 after executing the DataMultiply function. Read the code of the function and try to predict what it is doing.

Now, check the values of the dynamic array again. Has the values changed? Are the values correct?

- Continue the execution of program and observe what's printed by the TablePrint function.

Quit GDB once you are done, using the 'quit' command.

- (gdb) quit // quit debugger

Why do we need to free the pointer using **free(ptr_int)**, as done in the c code?

The given program (r11.c) prints out the multiplication table of the input value. Add few lines of code to extend the program to compute and print out the multiplication table of 2*(input value) along with the original multiplication table. Follow the instructions are given in the .c file.

Show the resulting tables to the TA.