2C-memory-api-lab

CST 334 (Operating Systems)
Dr. Glenn Bruns

Lab: C Memory API

Weird things happen when code uses memory allocation correctly and incorrectly. The following questions are simplified versions of questions 4, 5, 6, 7, 8 at the end of OSTEP chapter 14.

4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? (See starter code below.)
5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? Is the program correct?
6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run?
7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?
8. Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use realloc() to manage the vector. Use an array to store the vectors elements; when a user adds an entry to the vector, use realloc() to allocate more space for it. How well does such a vector perform? How does it compare to a linked list?

Here's some code to help you get started quickly. Read it and understand it before using it.

```c
#include <stdio.h>
#include <stdlib.h>

// starter code for Problems 4-8 of OSTEP, chapter 14
int main() {
   int *x;

   x = (int *)malloc(10 * sizeof(int));
   x[0] = 1;
   x[1] = 2;
   printf("x[1] = %d\n", x[1]);

   return 0;
}
```

If you still have time, try using gdb, the GNU debugger. It’s available on mlc104.