CSCI 262 Lecture 14 – Dynamic Allocation of Memory

Outline

- Dynamic allocation of memory
  - Use the `new` operator to allocate and obtain a pointer to memory.
  - Use the `delete` operator to deallocate memory given a pointer to that memory.
- Heap and stack
  - Memory associated with variables declared in functions is local to the function and lives in the function's stack frame on the stack; it will be reclaimed when the function returns.
  - Dynamically allocated memory lives on the heap, and its lifetime is independent of the function in which it is allocated – this allows it, e.g., to be passed out of the function and used in other contexts.
- Pointers and arrays
  - Array variables are effectively pointers to the start of the array.
  - Pointers can be used as array variables (if they point to array memory).
  - This equivalence also helps us understand pointer arithmetic, and why arrays are indexed starting at `0`.

Readings

Chapter 7 in your textbook covers the same material as today's lecture, in a somewhat different order. Note that the author refers to the "free store" - this is what the lecture notes refer to as "the heap".

Self Check

1. If we have

   ```
   int p[] = { 0, 1, 2, 3, 4 };
   int* q = p;
   ```

   and q holds the address `0x4304`, then:

   a) What is `*(q + 3)`?

   b) What is `&q[1]`?

2. Why is this code problematic?

   ```
   double* get_sum(double x, double y) {
       double ans = x + y;
       return &ans;
   }
   ```

For Further Practice

How many ways can you crash memory? Try writing programs that violate the rules listed on page 34 of today's slides; try going out of bounds on a locally declared array and on a dynamically allocated array; try the code in question #2 above. Which of these problems does your compiler warn you about? (You may be able to get additional warnings from your compiler using different compilation flags, or command line options – try searching the internet for ways to do this for your IDE or compiler.)