6.S096 | January IAP 2013 | Undergraduate

Introduction To C And C++

Lectures and Assignments

C Memory Management

Topics: Computer memory layout (heap, stack, call stack), pointers and addresses, arrays, strings, and manual memory allocation/deallocation.

Lecture Notes

Lecture 3: C Memory Management (PDF)

Lab Exercises

The primary goal of this lab period is to introduce pointers, addresses, arrays, and manual memory management.

Exercise 1

```
#include <stdio.h>

void square(int num) {
        num = num * num;
}

int main() {
        int x = 4;
        square(x);
        printf("%d\n", x);
        return 0;
}
```

Take a look at the above code. In lecture, I pointed out that function variables are passed by value, not reference. So this program will currently print out 4. Compile the code to confirm this.

Use pointers and addresses to modify the code so x is passed by reference instead and is squared. This will involve changes to the square function that *does not* involve changing void to int and giving square a return statement. Make sure your code compiles with `-Wall` flag without warnings.

Exercise 2

While coding up this exercise, listening to Hakuna Matata, I was so worry-free I forgot how to use C!

Fix the following code so that it creates a string `str` and copies “`hakuna matata!`” into it.

```
#include <stdio.h>

void main() {
        char str[];
        ???(, "hakuna matata!"); // this line should copy "hakuna matata!"
                                 // into our char array
        printf("%s\n", str);
        // Anything else?
}
```

After confirming your fix works, change the code to use heap memory instead of the stack. Remember, everything you `malloc` you must also `free`!

Assignment 3

Problem 1

sort (C)

In sort.c, I’ve implemented a basic implementation of insertion sort (not too efficient, but a very simple sorting algorithm). Look at and understand the code (read comments), and put the proper argument data type for the sort function’s first argument. Compile and run the code to make sure it works (it sorts the numbers).

Now, replace all array index access (places where I access the array with `[]`, such as in `A[i]`) in the entire program by using pointer addition instead. Also, where I create the array (line 34, `int array[n]`;), replace that with an array creation using malloc (of same size).

Hint: Since we’re using malloc, we must also make another change!

Make sure your program compiles and runs correctly (numbers are sorted).

Problem 2

resize (C)

The purpose of resize.c is to create an initial array of a user-specified size, then dynamically resize the array to a new user-specified size. I’ve given a shell of the code, including code to get user-specified sizes as `int`s.

However, the code is missing a few things. You must manage the memory for the array! Look at the comments in the code that describe what should be done and fill in blanks. Make sure the program compiles and runs as expected.

Solutions

Assignment 3 solution (PDF)