L17 - Review

December 1, 2019

1 Lecture 17

- an old final will be posted online
- last PASS workshops this weekend

1.1 Final Exam Information

- thirty-something questions, all multiple choice, on scantron
- part B is 2pts/question, part A 1pt/question
- no intentional syntax errors on any questions
  - no case sensitivity nonsense
- "no intentionally tricky questions" he says
- all the questions are covered by lectures and/or labs
- take my point: point out that there are many points on pointers

1.2 Exam review

1.2.1 Imperative Programming

- what is imperative programming?
- basics:
  - operators
  - declarations, statements:
    - loops, if, nested if, nested loops
  - arrays
  - structures
  - pointers
  - function calls
    - pass by reference vs. pass by value

- pointers
  - what? why? how? applications?
  - pointers and:
    - arrays
      - pass by reference
      - using const to avoid accidentally changing stuff
    - structures
    - struct consists of an array
    - stuct consists of another struct

```
In [ ]: // fun with pass by reference

        int addList(int [], int);
        int addElement(int);

        int main(void)
        {
            int list[3] = {100, 200, 300};

            addList(list, 0);
            printf("list[0]: %d\n", list[0]); // point one

            addElement(list, 1);
            printf("l)
        }
```

```
In [ ]: // struct in struct

        #include <stdio.h>
        #include <string.h>

        struct stud_course_t
        {
            int credit;
            char crs_title[50];
        };

        struct student_t
        {
            int id;
            char name[20];
            float percentage;

            struct struct_course_t crs_data;
        } stud_data;
        int main()
        {
            struct
        }
```

```
In [ ]: typedef struct
        {
            int id;
            char courses[30][50]; // 30 strings (courses), max 50 chars per string
        }student_t;

        student_t students[200]; // array of 200 students

        printf("%s\n", students[i].courses[1]);
```

- Big O notation for algorithm complexity
  - describes how algorithm complexity in terms of running time and space

```
In [ ]: // big O examples

        float a1[n]; // assume defined

        // completes in O(n)
        for (int i = 0; i<n; i++)
        {
            do_something();
        }

        float a2 [m][n]; // again assume defined

        // completes in O(n^2)
        for (int i=0; i<n; i++)
        {
            for (int k = 0, k<m, k++);
            {
                do_something();
            }
        }
```

1.3 Recursion

- be careful tracing multiple recursive cases
  - easy to get lost...

```
In [ ]: // traverse a tree using recursion (not a complete function)

        ... traverse_tree (tree_t start)
```

```
{
    if (start->left == NULL)
    {
        return start;
    }
    traverse_tree(start->left);
    // do something
    traverse_tree(start->right);
    ...
}

void main(){
    traverse_tree(root);
}
```