Scribe Notes for September 5, 2013
Contributors: Dennis Cui, Amna Hashmi, George Lok

Announcement: Volunteer with the Digital Learning Project to teach middle school kids about computer science

Class Mechanics

Course

CS61 is a first course in computers systems programming that is taught through immersion in the C language. You will learn how to write parallel programs, write simple network servers, and debug programs well among other things.

Assignments

6 assignments over the term, which are mostly done over the course of 2 weeks in pairs
The first assignment will be released Sept 5th night; the first C self-assessment should be completed individually and will be 'checked in', not graded, by the TFs on Friday, Sept 13
All assignments due Friday, 11:59:59 PM

Late Policy: 72 free late hours

After that, assignment drops 1 letter grade **per day (not every 4 hours)** down to F; an F is much greater than a 0, so it is far better to turn something in than nothing
If in pairs, lateness counts against both of you
Not allowed to turn in assignments after solutions are posted/discussed

Scribe Notes: student-produced lecture notes

Expected to cooperate on 1 set of scribe notes

Examinations

Midterm on 10/17 and Final during Finals Period
Situation has to be desperate in order to work around a student's schedule
Kohler: "I like giving tests that make you think. Last year, the tests made people think too much, so there was a lot of sadness and stress. Continue to take the course despite the fact the tests make you think and will be hard."
Will adjust grading breakdown in syllabus at end of course if necessary

Collaboration:

Do your own work.

Cite Help

Tell CS61 staff in ReadMe.txt who was the other set of eyes that viewed your code

Pairs can talk to other pairs in pseudocode.

Use Piazza to talk to each and talk to CS61 staff

Don’t post code to Piazza publicly; can post code in private message

Don't use other people's solutions

Don't use solutions from last year or online

Staff may compare your solution against those of last year, and there is enough scope of variation in the psets to detect most forms of copying

Don't try to pay people to do your solutions

Textbook:

Computer Systems: A Programmer's Perspective, Second Edition (also known as CS:APP2e) by Randal E. Bryant and David R. O'Hallaron. Prentice Hall, 2011, ISBN 0-13-610804-0.

CS61 does not follow text in chronological order

Readings will be posted shortly (different from last year's)

Not required but highly recommended

Will post sample exercise numbers from textbook around midterm time

Lecture: Systems Performance

Problem: Construction of a numeric set

A numeric set has 3 operations

1. new()
   return an empty numeric set
2. add(s,v)
   add number v to set s
3. remove_index(s, i)
   removes and returns ith element in sorted order

```
s = new()
add(s,5)
add(s,9)
add(s,1)
add(s,0)
remove_index (s,2)
        (returns 5)
        /* Remember, in computer programming, an index starts with 0! */
remove_index(s,2)
```

```
        (returns 9)
remove_index(s,0)
        (returns 0)
remove-index(s,0)
        (returns 1)
remove_index(s,300)
         */ POORLY DEFINED: Did not explicitly say what would happen if function tried
        to call an index outside of the array's size, so remove_index has a poorly defined
        function for this corner case. However, in this example, it will return 0. */
```

Pointers: use pointers to prevent adding/removing to a copy

From Wikipedia: "A **pointer** is a programming language data type whose value refers directly to (or "**points** to") another value stored elsewhere in the computer memory using its address."

C is a call-by-value language.

From Wikipedia: "In call-by-value, the argument expression is evaluated, and the resulting value is bound to the corresponding variable in the function (frequently by copying the value into a new memory region). If the function or procedure is able to assign values to its parameters, only its local copy is assigned — that is, anything passed into a function call is unchanged in the caller's scope when the function returns."

For example:

```
f() {
        int x = 0;
        g(x);
        printf("%d'n",x);
}
void g(int p) {
        p += 1;
}
```

WOULD PRINT 0! The variable x's value is used to initialize p in function g. They are two different objects with two different addresses. C is a call by value language because the value of the parameter is copied into the local scope of the function.

What if we print the addresses of x and p?

```
f() {
        int x = 0;
        g(x);
        printf("%p\n",&x);
    }
```

```
void g(int p) {
        p += 1;
        printf("%p\n",&p)
    }
```
PRINTS DIFFERENT ADDRESSES!

Kohler: “If the prototype for add and remove functions takes objects, they would be taking copies of a set. Any modification that would happen to the set might be thrown away. This is why we should expect in an API like this for operations on objects to involve pointers. If I pass a pointer, I am passing the address of that object and can modify the bytes associated with that object.”

```
void g(int *p) {
    *p += 1;
    printf("%p/n",&p)
}
f() {
    int x = 0;
    g(&x);
    printf("%d/n", x)
}
```

WILL NOW PRINT 1! By passing the address into function “g”, the function can access the original object “x” in function “f”, and thus, “x” can be incremented.

How should we implement a numeric set?

- Array
  - How big should I make the array?
  - Array needs to be as big as required
  - O(n) (worst-case time) to remove element from list; have to shift every element to left if remove s[0]
- Linked LIst
  - Dynamic memory allocation
  - Can add in sorted order so remove_index would be easier
- Binary tree
  - Balanced?

Actual Implementation:

- New -> new file
- Add -> Makes backup and reads from it. Write all elements in new file, and when you get to right
- place, print new element, and print rest of old elements

Available at file.c in lecture 2 code.

time: runs another program and tells you how much time it took

real 0m1.594s
wall clock time
user 0m0.004s
how much time application is taking
system 0m1.576s
how much time operator system is taking

Why does the operating system seem to be taking so much time??
From the CPU's (central processing unit's) point of view, most of the work that is required is to move data from the processor to the flash drive. This transfer is the limiting factor in terms of time!

What if we tried to implement the num set as a **singly-linked list**?
A singly-linked list is a data structure that contains pointers; points to other objects like itself

Available at list.c in lecture 2 code.

```
#include "numset.h"
#include <stdlib.h>
#include <stdio.h>

typedef struct node {
    unsigned value;
    struct node *next;
} node;

struct numset {
    node *head;
};

numset *numset_new(void) {
    numset *s = (numset *) /*CASTING*/ malloc(sizeof(numset));
    s->head = 0;
    return s;
}
```

time ./numset-list 100 10000

When adding and removing numbers from the linked list num set 10,000 times, it worked approximately 1000 times faster than the file implementation.

Dynamic memory allocation puts less stress on processor, so less work from the processor means

less time required.

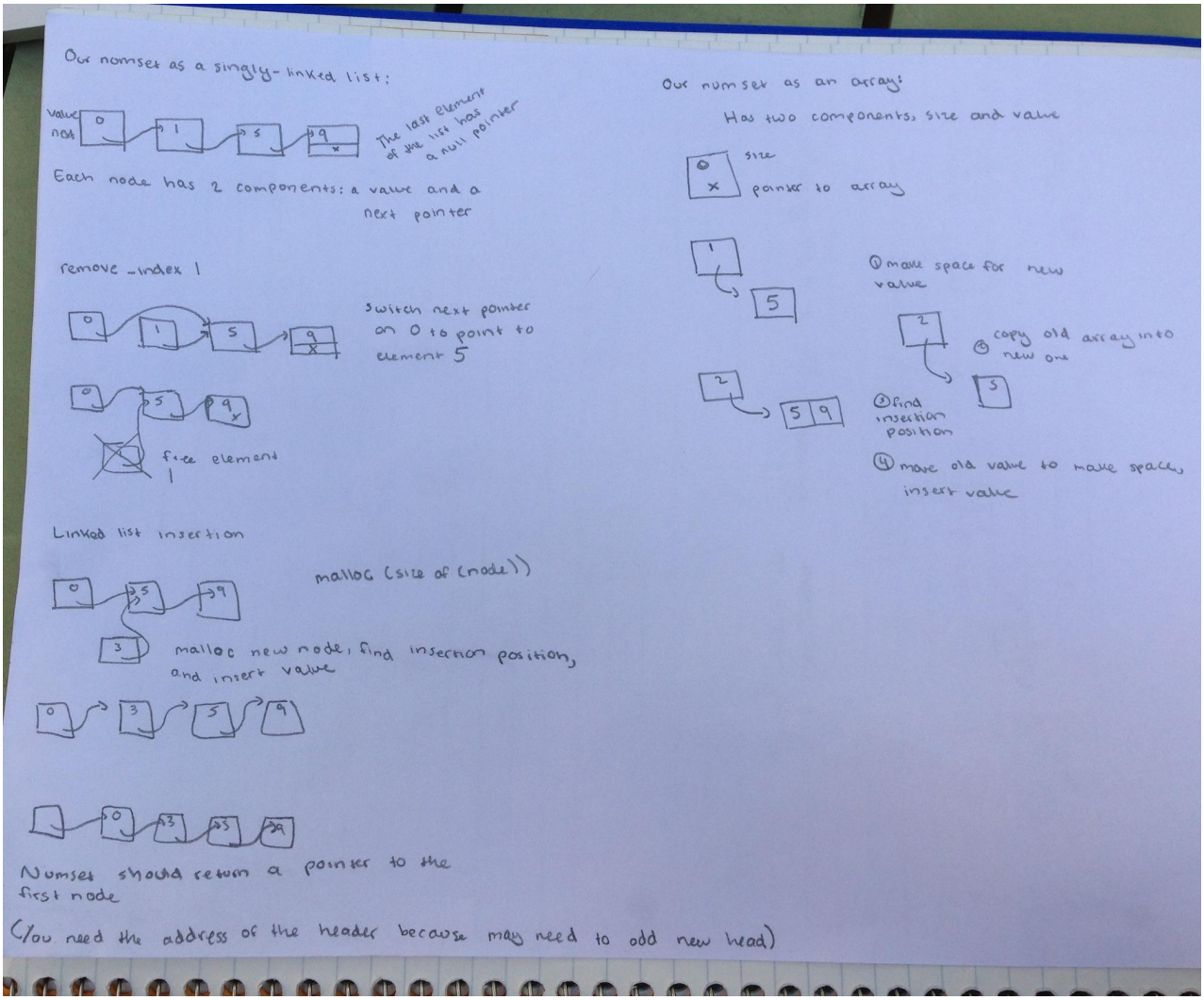
What if we tried to implement the num set as an **array (a sequence of objects of the same type)?**

Available at array.c in lecture 2 code.

```
struct numset {
   unsigned *v;
   unsigned size;
};

numset *numset_new(void) {
   numset *s = (numset *) malloc(sizeof(numset));
   s->v = 0;
   s->size = 0;
   return s;
}
```

Pictorial overview:



Time vs. n graphs - performance

As set size gets larger, the time it takes to complete the num set function is longer. Initially, the linked list implementation seems to be following one trajectory, but at ~2000 num set elements, its trajectory changes. Each list node is 16 bytes, which for 2000 elements would be 32 KB. That amount of memory would most likely be getting bigger than the level 1 cache (recently-used memory) of the machine, so the amount of time required at 2000 elements increases at a higher exponential constant that at fewer num set elements.

Both the linked list and array implementation have the same computational complexity of n, but the array ends up taking less time in most cases (except very small sample sizes) because the elements are stored next to each other and more likely to be stored in the cache. However, the file implementation works best when there is no more random access memory left. For example, Facebook uses 2 pedabytes of disk and not much memory.