CS 222: Data Structures: Vector and List

Chris Kauffman

Week 7-1

Logistics

Reading

- Ch 10 (Vector/List Data Types)
- Start finishing up exercises

Final Exam: Next Week Thursday

Homework 6 Posted Later Tonight

- 5 problems
- Counts as 2 HWs
- Will be due next Tue night

Exam 2 Results

Histogram

Count	Count
0 - 100	7
80 - 89	16
70 - 79	6
60 - 69	7
50 - 59	3
40 - 49	0

Summer 2015

Stat	Val
Count	39
Min	50.00
Max	96.00
Average	78.51
Median	82.00
Stddev	12.53

Summer 2014

Stat	Val
Count	31
Min	41.17
Max	96.47
Average	76.65
Median	82.35
Stddev	14.47

Vector/ArrayList Motivation

Array Limitations

- `malloc`'d arrays can't grow
- Very inconvenient in many situations
  - Reading from files
- A *data structure* is an arrangement of memory for convenience and efficiency
- Can create illusion of expandable arrays with the right data structure

Vector or ArrayList

- Like an array: get elements, set elements
- Can grow and resize (how?)

Vector Operations

- Create
- Destroy (free)
- Get current size
- Change current size
- Get an element at given index
- Set an element at a given index
- Append an element (to the end)

See `int_vector.h`

Reminder: Arrows for `struct` pointer field access

Actual `struct` access fields with `s.field`

```
channel_params cp = {...};
double f = cp.frequency;
```

Pointer to `struct` access fields with with `p->field`

```
channel_params * ptr = &cp;
double x = ptr->phase;
```

Will be used more with data structures as usually have pointer to vector/list

Demonstration: `sort_numbers.c`

- Read numbers from user
- Adds numbers to end of array during reading
- Code reads much simpler than previous attempts
- Can do I/O in single pass
- Hidden cost: `realloc()`

Tour of Vector Functions

Examine `vector/int_vector.c`

Practice: `int_vector_remove(vec, i)`

```
void int_vector_remove(int_vector *v, int rm_idx)
```

- Remove element at index `rm_idx`
- Must be in bounds (less than `size`
- Elements shift left to fill in gap
- Size decrements

Examples

```
int_vector *v = int_vector_create(); int_vector_add(v, 5);
int_vector_add(v, 8); int_vector_add(v, 4); int_vector_add(v, 1)
// [5, 8, 4, 1]
//  0  1  2  3
int_vector_remove(v, 1);
// [5, 4, 1]
//  0  1  2
int_vector_remove(v, 2);
// [5, 4]
//  0  1
```

realloc()

```
void *realloc(void *ptr, size_t size);
```

- Relative of `malloc()`
- Attempts to reallocate in place
- If no room, allocate and copy memory

Discussion: Efficiency of Expanding by 1

- Consider the efficiency of always growing vector by 1
- Very bad in practice: $O(N)$ append cost
- Alternative: Allocate extra space, leads to $O(1)$ amortized append cost
- Study in detail in CS 310

Sorting with `qsort()`

- If time, discuss the `qsort()` routine
- Library call to do sorting on arbitrary data
- Screwy because it requires a function argument

`Makefile` Defining your own Libraries, Compilation

- Examine the `vector` directory
- `Makefile` to build library
- Examine compile line for `sort_numbers.c`

Notion of a *List*

- Abstract notion of ordered elements
- Can index by number (0th element, 5th element)
- set(i,x) and get(i) operations
- Can grow list somewhere, end or beginning

Vector is a kind of List

- Advantages
- Disadvantages

Linked List

Fundamental in computer science

- Most basic use of pointers to create a useful data structure
- Compared to Arrays

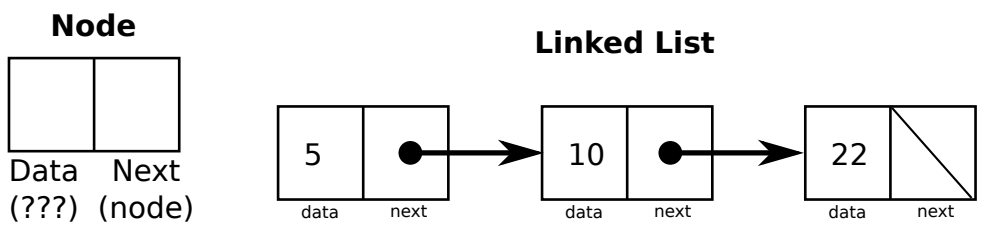
  Give up fast indexed access

  Gain unlimited append, flexible insert

- An element contains

  Data number, struct, pointer, whatever

  Next A pointer to another element



Interactive Demo: `read_all_numbers.c`

- People are stack frame variables
- People are nodes
- Chris is `malloc()`