CS 222: Linked Lists, Queues, Stacks

Chris Kauffman

Week 7-2

Logistics

Reading

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

Final Exam: Next Week Thursday

Homework 6 Up

- HWs reweighted, worth 7% each
- Due next Tue night

HW 6

Problem 1: I/O on `channel_params`

- `save_channel_params()`: write array to a file (with what function?)
- `load_channel_params()`: load array from a file

Problem 2: List Insertion `int_list_insert()`

- Discuss linked lists today
- Insert integer at an arbitrary index in list
- Requires traversing the list
- Deal with special cases

Problem 3: Digital Clock Display

- Given seconds since beginning of day
- Determine AM/PM + hour + minute
- Manipulate bits so that right LCD bars are shown (shifts, bitwise-or/and, masks)
- This one is AWESOME

Review: Vector

- Describe similarities of `int_vector` to standard arrays
- Describe differences of `int_vector` from standard arrays
- How does a vector grow?
- Strengths and weaknesses of `int_vector`?
- What tool was used to easily build all the code for `int_vector`?
- What is a linked list/linked nodes?
- How does a linked list differ from an array?

Linked List

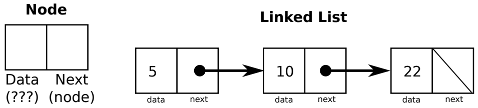
Fundamental in computer science

- 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



int_list

- Linked list of integers
- Created for int_nodes
- Housed in an int_list
- Small library built with a Makefile

See int_list.h and int_list.c

Demos

```
int int_list_size(int_list *list)
```

- Return how many nodes are in the list
- Iterate through the list counting nodes
- `NULL` ends lists

```
int_node* int_list_set(int_list *list, int index,
                       int new_val)
```

- Set a specific element
- Illegal to set non-existent element
- First node is index 0
- Error to remove nonexistent node
- Problems with removing first element, return new head

Exercise: `int_list_get()`

```
int_list_get(int_list *list, int index)
```

- Return the data stored at node `index`
- First node is index 0
- On out of bounds access, signal error with `report_error(msg)`
- *Very* similar to `int_list_set(list,index,new_data)`

Exercise: `int_list_remove()`

```
int int_list_remove(int_list *l, int index)
```

- Remove a node from the list at position `index`
- Iterate through list to node before removal index
- Point preceding next "over" removal node
- Free removal node
- Several special cases
  - Removal from head of list
- Return 1 if node is removed
- Return 0 if `index` is out of bounds
  - Length is N
  - In bounds: index $0, 1, 2, ..., N - 1$
  - Out of bounds: index $N, N+1, N+2,...$
  - Careful for index N case

make me a program as quick as you can

- Test `int_list_remove()`
- Use `test_list_remove.c` with `main()`
- Requires building the `int_list` library
- Requires linking to it
- `Makefile` is really handy

```
> make
gcc -c -g test_list_remove.c -I int_list
cd int_list && make
make[1]: Entering directory 'int_list'
gcc -c int_list.c
ar rcs libint_list.a int_list.o
make[1]: Leaving directory 'int_list'
gcc -o test_list_remove -g test_list_remove.o -lint_list -L int_list
> test_list_remove 10 20 30 40 50 60
```

Comparison of Linked List and Array List (vector)

- Easy to add to the front of a list of nodes
- Easy to add to the back of a vector of ints
- Harder to add to other sides of these

	Add to Front `prepend()`	Add to back `append()`
List	Cheap: *O(1)*	Expensive: *O(N)*
Vector	Expensive: *O(N)*	Cheap: *O(1)*

- Vector/ArrayList favored in most applications due to cache
- Linked list has advantage of best worst-case adding

Two Data structures: Stack and Queue

CS folks love their containers, and more generally data structures

Stacks and Queues

- Abstract containers, can hold anything
- Both support a few operations to preserve order
- Stack: Like a pile
  - Put things on the top
  - Take things from the top
- Queue: Like a line
  - Put things at the rear
  - Take things from the front

Zyante Ch 12 has brief demos of these

- Also hash table, read it but won't discuss in class

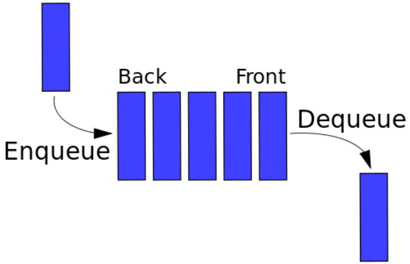
Stack

LIFO: Last in first out. Basic operations are

Push	Put something on the top of the stack (void)
Pop	Remove the top stack element (void)
Top	Look at the top element (data)

Demo this.
Note: The stack part of program memory acts like the data structure: calling a function pushes stuff on the stack, finishing a function pops stuff off the stack.

Queue



FIFO : First in first out (a line). Basic operations are

Enqueue	Put something at the end of the line (void)
Dequeue	Remove whatever is first in line (void)
Front	Look at what's first in line (data)

Demo this.

How would I build them?

New data types, not built into C, require structs

- `typedef struct {...} stack;`
- `typedef struct {...} queue;`

Brainstorm ideas for implementing

- Stack?
- Queue?

Any other data structures that would make this easier?

str_node

Basis for linked node based Queues and Stacks

```
typedef str_node_struct{
  char data[128];
  struct str_node_struct *next;
} str_node;
```

Basis for array-based Queues and Stacks is

```
char **elements;
int size;
```

Interactive Development

Develop one or several of these for =strings

Easiest	Stack of strings using arrays
Easy	Stack strings using linked list (use `str_node`)
Medium	Queue of strings using linked list (`str_node`)
Hard	Queue of strings using arrays

Stack

LIFO: Last in first out. Basic operations are

Push	Put something on the top of the stack (void)
Pop	Remove the stack element (void)
Top	Look at the top element (data)

Queue

FIFO : First in first out (a line). Basic operations are

Enqueue	Put something at the end of the line (void)
Dequeue	Remove whatever is first in line (void)
Front	Look at what's first in line (data)