Lecture 14: Finish doubly linked lists, start stacks

Admin

Skim Doubly Linked Lists, section 10.2

This week: Stacks and Queues (not priority queues, though.)

Reading for upcoming lecture on binary tree nodes

- B.5.2, Rooted trees
- B.5.3, Binary trees

Exams

Average: 74%

Big number of B-C grades

Big number of F grades

If you’re scoring badly, you probably need to brush up. Come to my office hours, or the CSEL LA help hours, or the BOLD LA help hours.

Doubly linked lists go forward and backward

```
struct node {
        int data;
        node* next;
        node* prev;
};
```

We still kept a pointer to the first node in the list (head_ptr).

For speed, we also kept track of the last node in the list (tail_ptr).

Much of our linked list code did not change much.

But we added a routine to print backwards, so we test these new links…

Doubly linked lists

We began with my test code, Makefile, and headers from the singly linked list code. We modified as we went. I missed 2 things in remove…

```
if (removed_node_ptr->data == target)  {
        if (head_ptr->next == nullptr)  tail_ptr = nullptr;
        head_ptr = head_ptr->next;
        if (head_ptr != nullptr)
                head_ptr->prev = nullptr;
        delete removed_node_ptr;
        removed_node_ptr = nullptr;
        return true;
}
```

Stacks

Store data in order given to them. Adding to stacks is a push:

push(1)

```
            1
```

push(2)

```
            2
            1
```

push (-3)

```
            -3
            2
            1
```

Stacks

Stacks can also tell you their top item:

push(1)		
	1	top == 1
push(2)		
	2	top == 2
	1	
push (-3)		
	-3	top == -3
	2	
	1	

Stacks

Removing the top element of a stack is a pop():

	-3	size == 3
	2	
	1	
pop()	2	size == 2
	1	
pop()	1	size == 1
pop()		size == 0

Why stacks?

Memory for previous input

Example: check if {} match in a program

{{{{}}}}

{{} {{{}{}}}}

When we see {, push ‘{‘ onto stack

When we see }, pop stack

If stack’s empty at the end and never underflowed,

your {} are balanced

Why stacks?

Memory for previous input: reverse Polish notation

Example: 9 45 * 2 / 8 + 9 - = ???

```
When see number, push number onto stack
When see operator (+-*/),
        pop 2 numbers off stack
        combine them with the operator,
        push the resulting number on the stack.
If stack contains one number at the end and never
        underflowed,  your answer is this number
```

How stacks?

Unsorted dynamic array

- push: add
- pop: remove last element
- top: return last element
- empty: count

Linked list:

- push: add to head
- pop: remove from head
- top: return head element
- empty: head_ptr

Queues

Opposite rule from stacks. First come, first served.

Unsorted dynamic array

- push: add to end
- pop: remove ‘first’ element
- front: return ‘first’ element
- empty: count

Linked list:

- push: add to tail
- pop: remove from head
- top: return head element
- empty: head_ptr

Queues

Unsorted dynamic array

- push: add to end
- pop: remove ‘first’ element
- front: return ‘first’ element
- empty: count

push(1)	1
push(2)	1 2
pop()	_ 2
push(3)	_ 2 3
push(4)	_ 2 3 4
pop()	_ _ 3 4

Queues

Circularize the array. Suppose its size is 4.

push(1)	1	
push(2)	1 2	
pop()	_ 2	
push(3)	_ 2 3	
push(4)	_ 2 3 4	
pop()	_ _ 3 4	
push(5)	5 _ 3 4	wrap around…
push(6)	5 6 3 4	
	B F	

What condition could we use to test for this?