Day's Goals

Thursday, February 8, 2018 4:43 PM

Finish SLL class
- show multiple file compilation
add destructor

Doubly Linked List

Multi-File Compilation

single file approach single.cpp

```
# include <      >
using namespace ..
struct
 {  //
  }
Class
{ private:
     //
  public:
     //
}

int main( )
{


}
```

Multiple-file Approach

Multiple-file Approach

SLL.h

```
#ifndef SLL_H
#define SLL_H

struct
{

}

Class SLL
{
private:
    Node *head, *tail;
public:
    List;  //constructor
  ~List;
    void growList(string newItem);
      :

}
#endif
```

SLL.cpp

```
#include <     >
using namespace std;

#include "SLL.h"

SLL::SLL()
{
   // constructor definition
}

SLL::~SLL()
{
   // destructor def
}

void SLL::growlist(string newItem)
```

```
void SLL::growList(string ...)
{
   // definition
}
```

SLL_test.cpp

```
#include <iostream>
using ...
#include "SLL.h"

int main()
{
    SLL l;
    l.displayList();
    return 0;
}
```

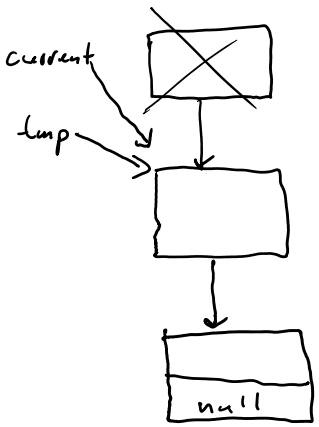
)

Destructor

- Method for "clean-up" gets called automatically when function goes out of scope.
- Default generated if none explicitly defined.
- Need explicitly defined when working w/ dynamically allocated mem.

e.g. Destructor for SLL class

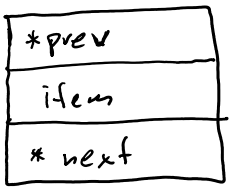
```
current = head

while (current != null)
{
  tmp = current->next
  delete current;
  current = tmp
}
```



Doubly Linked List

Similar to singly linked, except in each node 2 pointers instead of 1.

To keep track of whole list, keep head + tail.



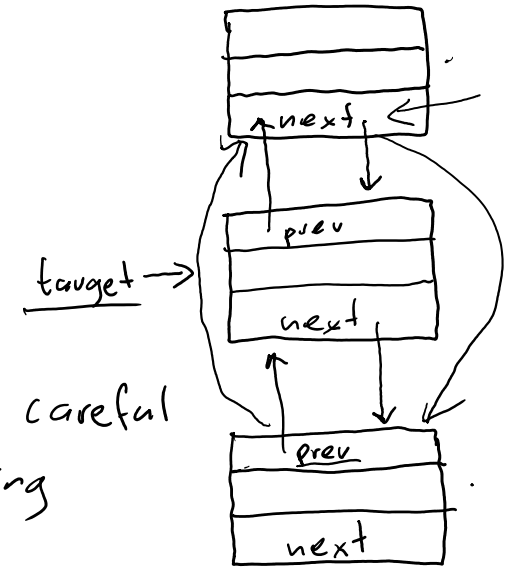
Advantage over SLL

- Can traverse in both directions

- Delete operation is faster
  - ↳ given target node, no need to traverse list

Delete Node

e.g. given address of node to be deleted



need to be careful when reassigning pointers

target -> prev -> next = target -> next

target -> next -> prev = target -> prev