MORE ON GDB AND RULE OF THREE
RECURSION
INTRO TO PA01

Problem Solving with Computers-II

```
#include <iostream>
using namespace std;

int main(){
   cout<<"Hola Facebook\n";
   return 0;
}
```

Announcements

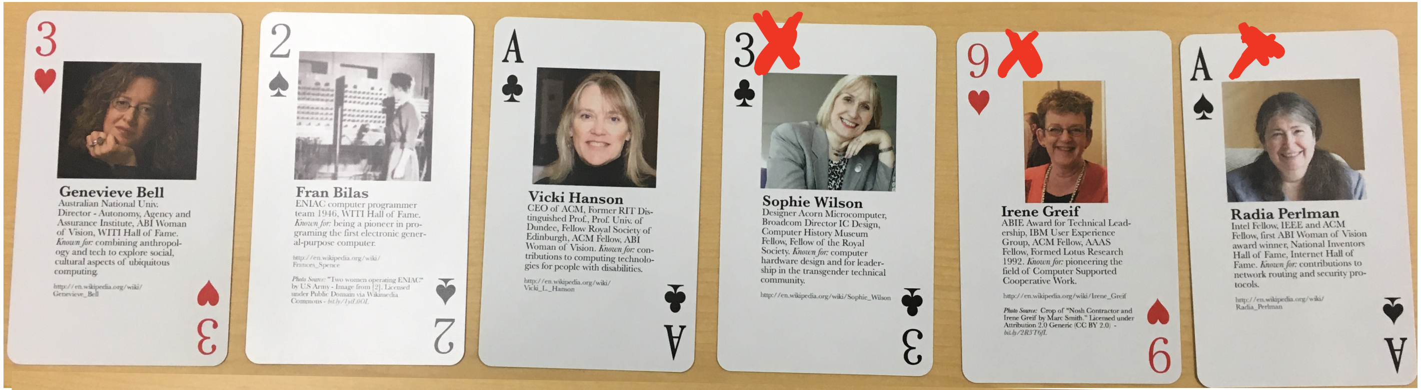
- PA01 will be released tomorrow (04/18), due (05/07)
- Lab02 due tomorrow Thursday (4/18)
- Midterm next week (Wed)(04/24) - All topics covered so far.

  For more details visit https://ucsb-cs24.github.io/s19/exam/e01/
- TAs and Tutors will hold review sessions on Monday and Tuesdays (1p-2p).

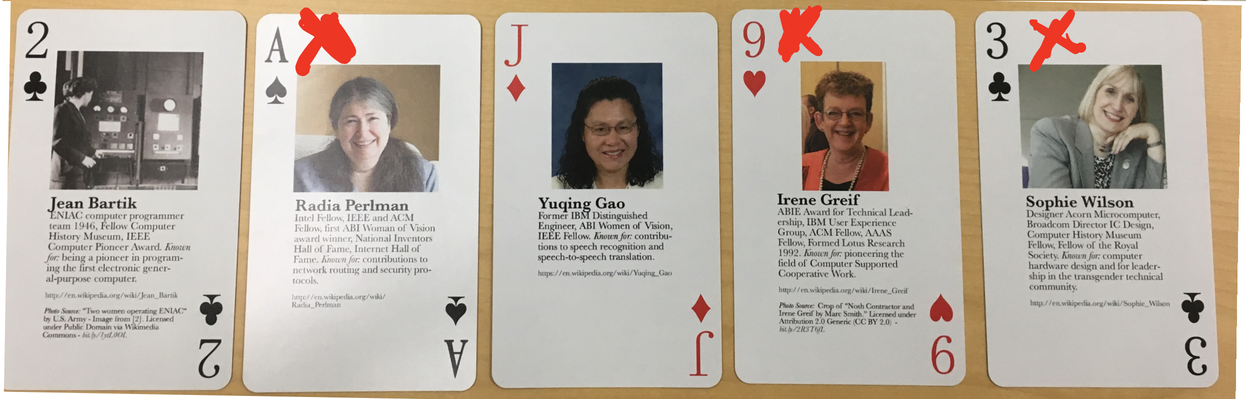
  Look out for announcements on Piazza

PA01: Card matching game with linked lists

Alice:


3 ♥
Genevieve Bell
Australian National Univ. Director - Autonomy, Agency and Assurance Institute, ABI Woman of Vision, WITI Hall of Fame. Known for: combining anthropology and tech to explore social, cultural aspects of ubiquitous computing
http://en.wikipedia.org/wiki/Genevieve_Bell
2 ♠
Fran Bilas
ENIAC computer programmer team 1946, WITI Hall of Fame. Known for: being a pioneer in programing the first electronic general-purpose computer.
http://en.wikipedia.org/wiki/Frances_Spence
Photo Source: "Two women operating ENIAC" by US Army - Image from [2]. Licensed under Public Domain via Wikimedia Commons - bit.ly/1ytL0OL
A ♣
Vicki Hanson
CEO of ACM, Former RIT Distinguished Prof., Prof. Univ. of Dundee, Fellow Royal Society of Edinburgh, ACM Fellow, ABI Woman of Vision. Known for: contributions to computing technologies for people with disabilities.
http://en.wikipedia.org/wiki/Vicki_L._Hanson
3 ♣
Sophie Wilson
Designer Acorn Microcomputer, Broadcom Director IC Design, Computer History Museum Fellow, Fellow of the Royal Society. Known for: computer hardware design and for leadership in the transgender technical community.
http://en.wikipedia.org/wiki/Sophie_Wilson
9 ♥
Irene Greif
ABIE Award for Technical Leadership, IBM User Experience Group, ACM Fellow, AAAS Fellow, Formed Lotus Research 1992. Known for: pioneering the field of Computer Supported Cooperative Work.
http://en.wikipedia.org/wiki/Irene_Greif
Photo Source: Crop of "Nosh Contractor and Irene Greif by Marc Smith." Licensed under Attribution 2.0 Generic (CC BY 2.0) - bit.ly/2R3T6fL
A ♠
Radia Perlman
Intel Fellow, IEEE and ACM Fellow, first ABI Woman of Vision award winner, National Inventors Hall of Fame, Internet Hall of Fame. Known for: contributions to network routing and security protocols.
http://en.wikipedia.org/wiki/Radia_Perlman

Bob:


2 ♣
Jean Bartik
ENIAC computer programmer team 1946, Fellow Computer History Museum, IEEE Computer Pioneer Award. Known for: being a pioneer in programming the first electronic general-purpose computer.
http://en.wikipedia.org/wiki/Jean_Bartik
Photo Source: "Two women operating ENIAC" by U.S. Army - Image from [2]. Licensed under Public Domain via Wikimedia Commons - bit.ly/1ytL0OL
A ♠
Radia Perlman
Intel Fellow, IEEE and ACM Fellow, first ABI Woman of Vision award winner, National Inventors Hall of Fame, Internet Hall of Fame. Known for: contributions to network routing and security protocols.
http://en.wikipedia.org/wiki/Radia_Perlman
J ♦
Yuqing Gao
Former IBM Distinguished Engineer, ABI Women of Vision, IEEE Fellow. Known for: contributions to speech recognition and speech-to-speech translation.
https://en.wikipedia.org/wiki/Yuqing_Gao
9 ♥
Irene Greif
ABIE Award for Technical Leadership, IBM User Experience Group, ACM Fellow, AAAS Fellow, Formed Lotus Research 1992. Known for: pioneering the field of Computer Supported Cooperative Work.
http://en.wikipedia.org/wiki/Irene_Greif
Photo Source: Crop of "Nosh Contractor and Irene Greif by Marc Smith." Licensed under Attribution 2.0 Generic (CC BY 2.0) - bit.ly/2R3T6fL
3 ♣
Sophie Wilson
Designer Acorn Microcomputer, Broadcom Director IC Design, Computer History Museum Fellow, Fellow of the Royal Society. Known for: computer hardware design and for leadership in the transgender technical community.
http://en.wikipedia.org/wiki/Sophie_Wilson

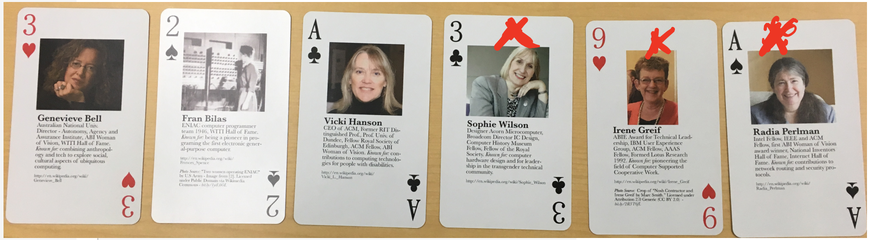
Review PA01: Card matching game with linked lists

Correct output after running `make && ./game alice_cards.txt bob_cards.txt`:

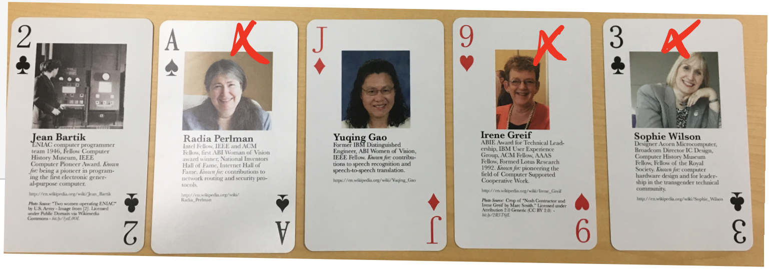
```
Alice picked matching card c 3
Bob picked matching card s a
Alice picked matching card h 9

Alice's cards:
h 3
s 2
c a

Bob's cards:
c 2
d j
```

Note: 0=10, a=ace, k=king, q=queen, j=jack

Contents of `alice_cards.txt`:



Contents of `bob_cards.txt`:



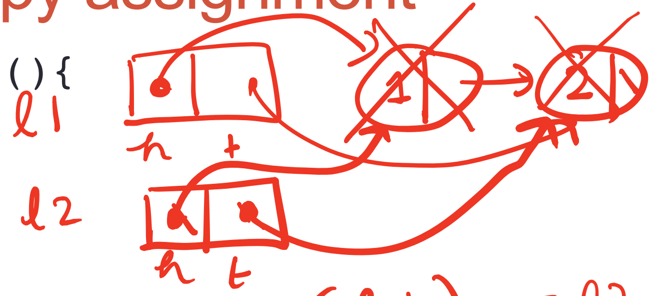
GDB: GNU Debugger

- To use gdb, compile with the -g flag
- Setting breakpoints (b)
- Running programs that take arguments within gdb (r arguments)
- Continue execution until breakpoint is reached (c)
- Stepping into functions with step (s)
- Stepping over functions with next (n)
- Re-running a program (r)
- Examining local variables (info locals)
- Printing the value of variables with print (p)
- Quitting gdb (q)
- Debugging segfaults with backtrace (bt)

* Refer to the gdb cheat sheet: http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf

Behavior of default copy assignment

```
void test_copy_assignment(){
   LinkedList l1;
   l1.append(1);
   l1.append(2);
   LinkedList l2;
   l2 = l1;
   TESTEQ(l1, l2, "test copy assignment");
}
```



$l2 . \text{equal} (l1); \equiv l2 . \underbrace{\text{operator=}}(l1)$

name of a function

In this case l1 & l2 share the same nodes. After the test function returns l1's destructor is called which deletes l1's nodes. Then l2's destructor is called which tries to delete the same nodes → double free (segfault)

Assume:

destructor: overloaded

copy constructor: overloaded

copy assignment: default

What is the output?

A. Compiler error

B. Memory leak

[✓ C. Segmentation fault]

D. Test fails

E. None of the above

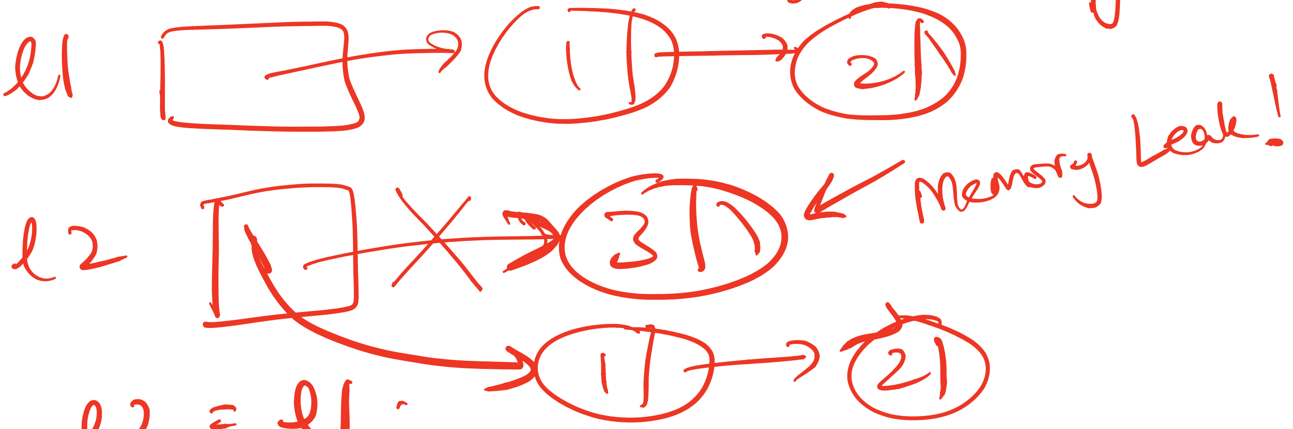
Write another test case for the copy assignment

```
void test_copy_assignment_2(){
        // Similar to previous case except l2 has existing
        // nodes before the assignment operator is applied

          Linked List l1;
          l1.append(1);
          l1.append(2);
          LinkedList l2;
          l2.append(3);
          l2 = l1;
}         TESTEQ(l1, l2, "case two");
```

Write another test case for the copy assignment

```
void test_copy_assignment_2(){
```

Suppose that the assignment operator
has the exact same implementation as the
overloaded copy constructor



Memory Leak!

```
l2 = l1;
```

```
}
```

Overloading Binary Comparison Operators

We would like to be able to compare two objects of the class using the following operators

==

!=

and possibly others

all these operators can be used with Linkd List objects If you implement them as operator functions.

Last class: overloaded == for LinkedList

To overload the = operator for Linkd List, declare it as a public member function as follows:

```
Void operator= (Const LinkdList & source);
```

A void return type only works if the intended usage is always of the form l1=l2;

In the lab02 code, the return type for the assignment operator was a reference to a LinkedList :

$$\underline{\text{LinkedList \&}}\ \text{operator} = (\text{const LinkedList \& source});$$

↑

① The return type is a LinkedList so that the overloaded operator can be used in more complex assignment expressions. For example expressions of the form

$$l1 = \underbrace{l2 = l3}_{} ;$$

↑ This subexpression calls l2's `=' operator passing l3 as a parameter

If the operator returns a " void " then the expression $l1 = l2 = l3 ;$ will boil down to

$l1 = \text{void} ;$

In this case the `=' operator is being used between a LinkedList object and a void which is problematic : no matchg function definition

So, if you want to use your implementation of the assignment operator in expressions of the form $l1 = l2 = l3;$, it should return a Linked List.

If the return type is not a reference, the copy constructor will be called just to return a value: This is unnecessary which is why we return a reference.

Overloading input/output stream

Wouldn't it be convenient if we could do this:

```
LinkedList list;
cout<<list; //prints all the elements of list
```

+ this expects a function of the form

operator<< (ostream& out LinkedList list);

return type may be void but as before if you would like to write expressions like:

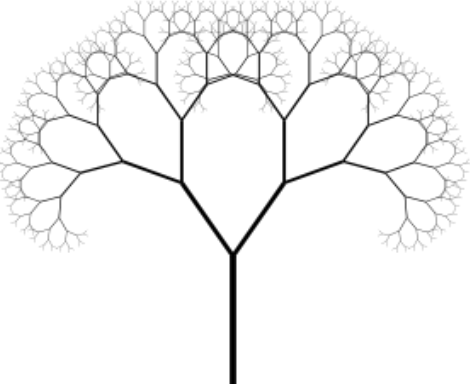
cout<<e1<<e2; return type should be ostream&

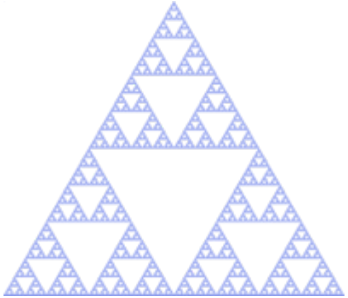
Overloading Binary Arithmetic Operators

We would like to be able to add two points as follows

```
LinkedList l1, l2;

//append nodes to l1 and l2;

LinkedList l3 = l1 + l2 ;
```

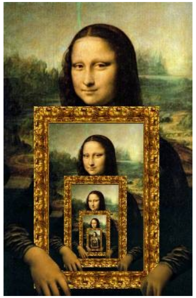
Recursion





Sierpinski triangle

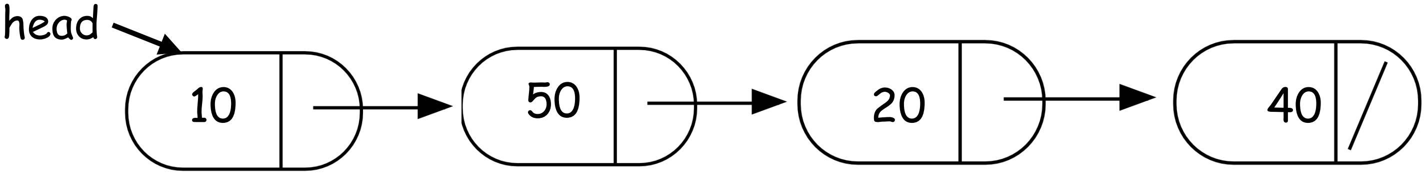
Zooming into a Koch's snowflake



Describe a linked-list recursively

Which of the following methods of LinkedList CANNOT be implemented using recursion?

A. Find the sum of all the values
B. Print all the values
C. Search for a value
D. Delete all the nodes in a linked list
E. All the above can be implemented using recursion



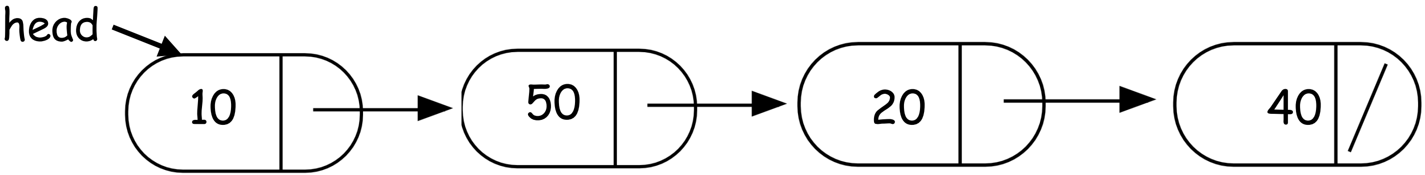
```
int IntList::sum(){

      //Return the sum of all elements in a linked list
  }
```

Helper functions

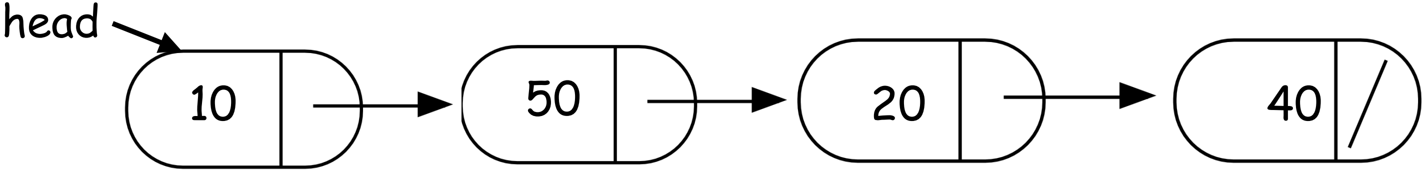
- Sometimes your functions takes an input that is not easy to recurse on
- In that case define a new function with appropriate parameters: This is your helper function
- Call the helper function to perform the recursion
- Usually the helper function is private

For example

```
Int IntList::sum(){

    return sum(head);
      //helper function that performs the recursion.

}
```



```
int IntList::sum(Node* p){






 }
```



```
bool IntList::clear(Node* p){






 }
```

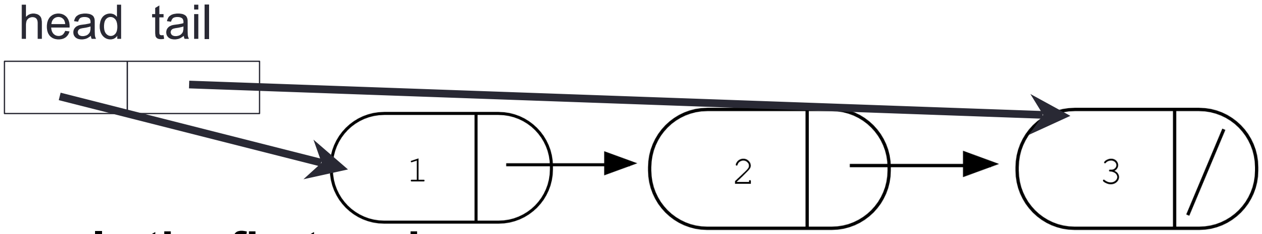
Concept Question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
class Node {
    public:
       int info;
       Node *next;
};
```

Which of the following objects are deleted when the destructor of Linked-list is called?

(A)



(B): only the first node

(C): A and B

(D): All the nodes of the linked list

(E): A and D

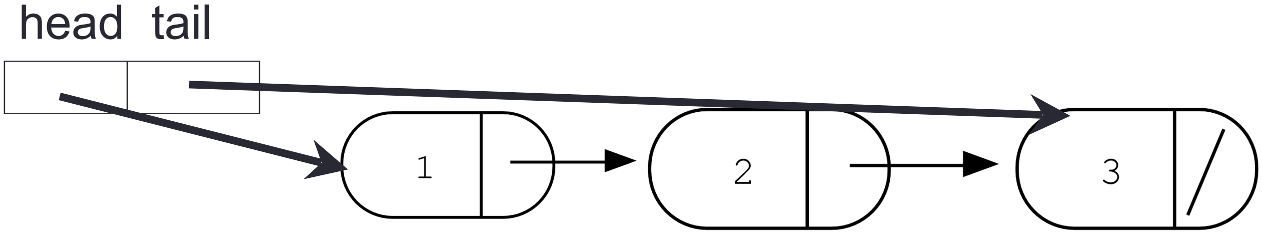
Concept question

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```

Which of the following objects are deleted when the destructor of Linked-list is called?

(A)



(B): All the nodes in the linked-list

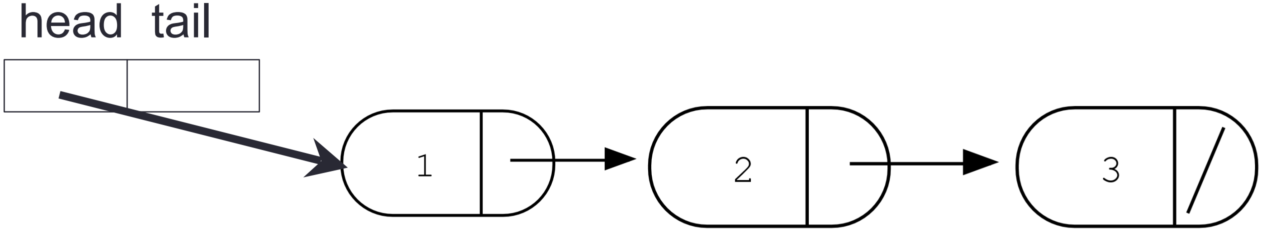
(C): A and B

(D): Program crashes with a segmentation fault

(E): None of the above

```
LinkedList::~LinkedList(){
   delete head;
}
```

```
Node::~Node(){
   delete next;
}
```



CAUTION
THIS SIGN HAS
SHARP EDGES
DO NOT TOUCH THE EDGES OF THIS SIGN
ALSO, THE BRIDGE IS OUT AHEAD

Next time

- Binary Search Trees