vector class

- homogeneous aggregate with random access
  - templated class:  `Vector<int> iv; Vector<string> sv;`
  - accessible using `#include "vector.h"`
  - construct by specifying # elements (initial value)

- access with indexes starting at 0
  ```
  for(k=0; k < eltCount; k++)
    cout << list[k] << endl;
  ```

- growable, using `resize` or, this semester, `append`
  - choice: vector tracks # elts or client code does
What is a class?

- **encapsulates state and behavior**
  - state is the innards, of which a class is comprised
  - behavior is what a class does, how it is known
- **state is (typically) private, behavior public/private**
  - private functions callable from public, not client
- **class is constructed, and destructed (sometimes)**
  - automatic when an object/variable is defined
  - (destruction) when object gone (deleted/scope)
- **examples: ifstream, string, vector, dice, date,...**
  - what about int, char, double, bool?
What is a pointer, why use pointers?

- **indirect reference**
  - index in book
  - forwarding address
  - universal email: `ola@acm.org`

- **pointer benefits**
  - multiple pointers to same object
  - self-referential structures
  - attach/detach object
  - facilitates inheritance

**linked list, benefits?**

```c
struct Node {
    string info;
    Node * next;
};
Node * list = __________;
```

```
"dog" ----> "cat"
```

Node * temp = new Node;
temp->info = “ant”;
temp->next = list->next;
list->next = temp;

how to remove cat?
pointer operations, properties

- a pointer is a variable/object
  - points to something
  - how to find what’s pointed to?
  - * has lots of meanings

- provide a reference
  (something to point to)
  - list = new Node;
  - list = 0; list = NULL;
  - temp->next = list;
  - temp = lastNode(list);

```
Node * temp = new Node;
temp->info = "ant";
temp->next = list->next;
list->next = temp;
```
using linked lists

- facilitate splicing: adding or deleting nodes
  - time to splice is independent of # nodes in list (compare to vector)
  - 33 elements means 33 nodes (compare vector)

- problems managing memory (using new)
  - possible to access “bad” memory
  - need to clean up, throw unused space back
List traversals

```c
int nodeCount(Node * list)
{
    int total = 0;
    while (list != 0)
    {
        total++;
        list = list->next;
    }
    return total;
}
```

```
//....
int animalCount = nodeCount(animalList);
```

- **what is value of list after loop? is this a problem?**
- **alternatives to loop test?**
- **what about a recursive version?**
Storing words from a file in a list

Node * makeListFromFile(istream & input)
// precondition: input is readable
// postcondition: returns pointer to first node of list
//                with strings in input, in same order as in input
{
    Node * first = 0;
    Node * last = 0;
    string word;

    while (input >> word)
    {
        last->next = new Node;
        last = last->next;
        last->info = word;
        last->next = 0;
    }
    return first;
}

● Are there any problems here? anything missing?
Using a header node

- advantages of header node
  - ▶
  - ▶
- disadvantages?
  - ▶
- what about counting nodes?
- what about printing nodes?

in `makeListFromFile` code, how are first and last initialized?
Deleting one node, some nodes, all nodes

- **delete operator**

```cpp
Node * temp = new Node;
delete temp;
```

- pointer is argument, but through pointer deletes storage
- access after deletion?
- delete first? first->next?

- **Do NOT delete until kinks worked out of code**
specified node, all nodes, assume header node

```c
void remove(Node * list,
            const string & key)
{
    Node * temp = 0;

    while (list != 0)
    {
        if (list->info == key)
        {
            list = list->next;
        }
    }
}
```

```c
void remove(Node * list)
{
    Node * temp = 0;
    while (list != 0)
    {
        list = list->next;
    }
}
```