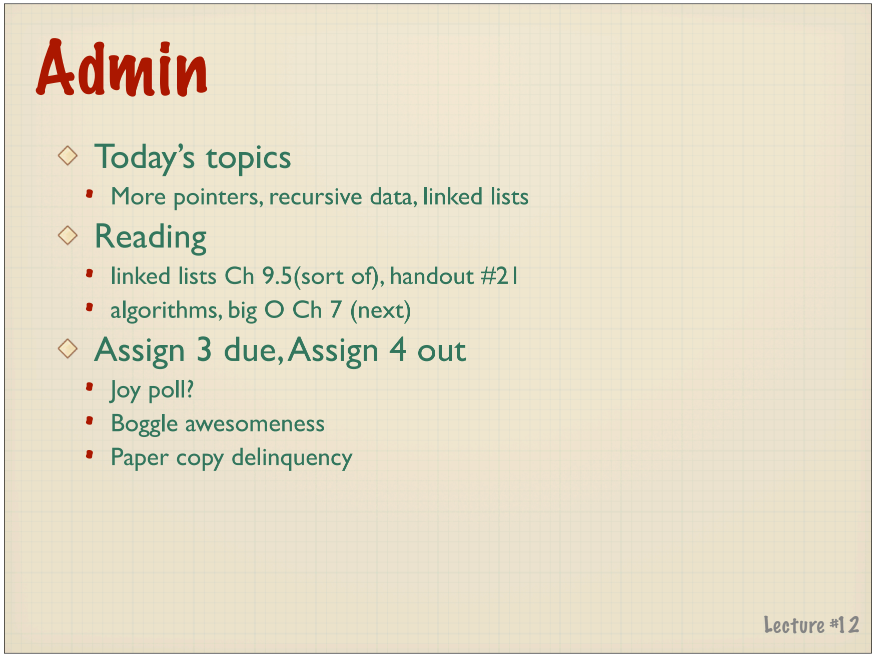
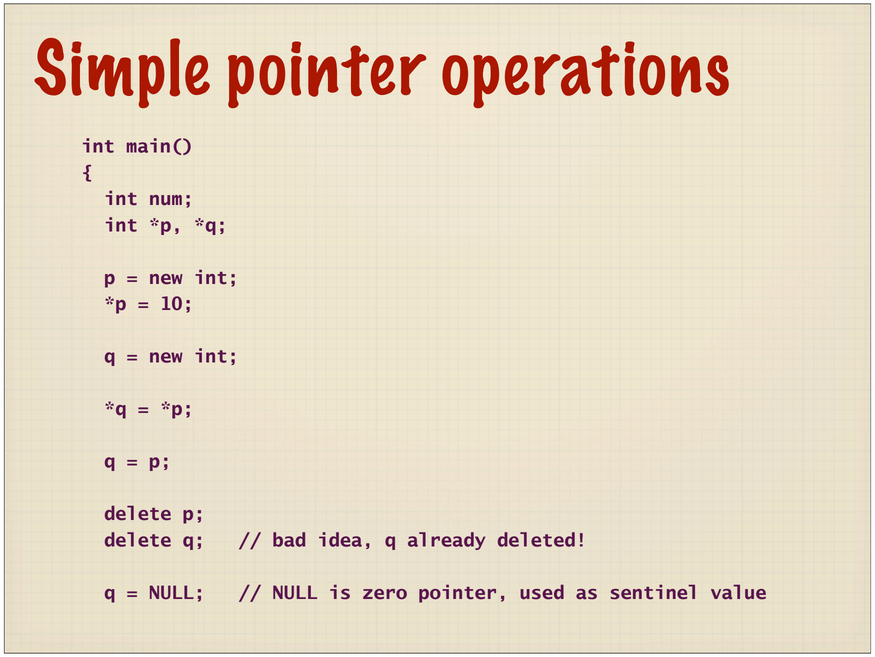


Admin

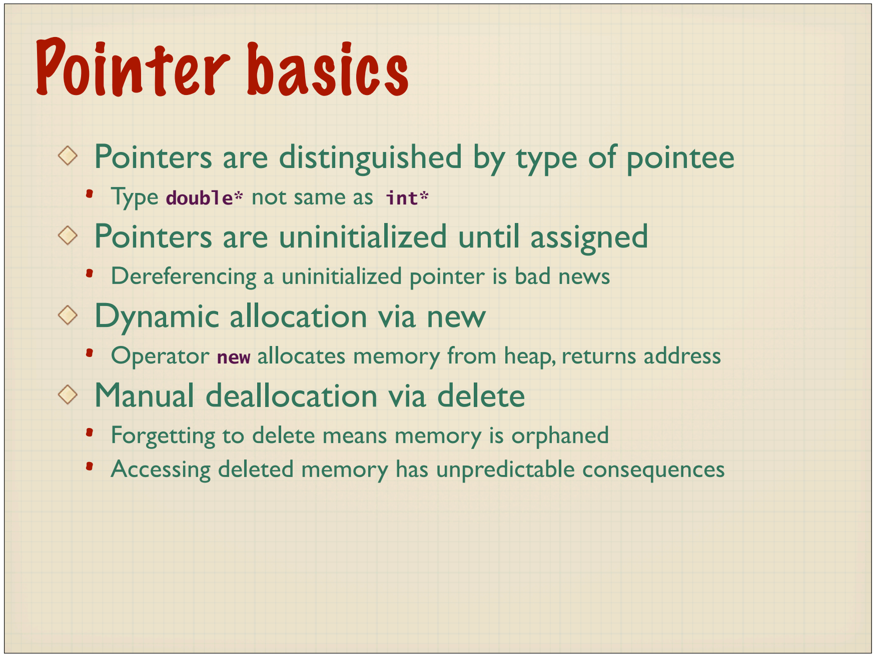
- Today's topics
  - More pointers, recursive data, linked lists
- Reading
  - linked lists Ch 9.5(sort of), handout #21
  - algorithms, big O Ch 7 (next)
- Assign 3 due, Assign 4 out
  - Joy poll?
  - Boggle awesomeness
  - Paper copy delinquency

Lecture #12



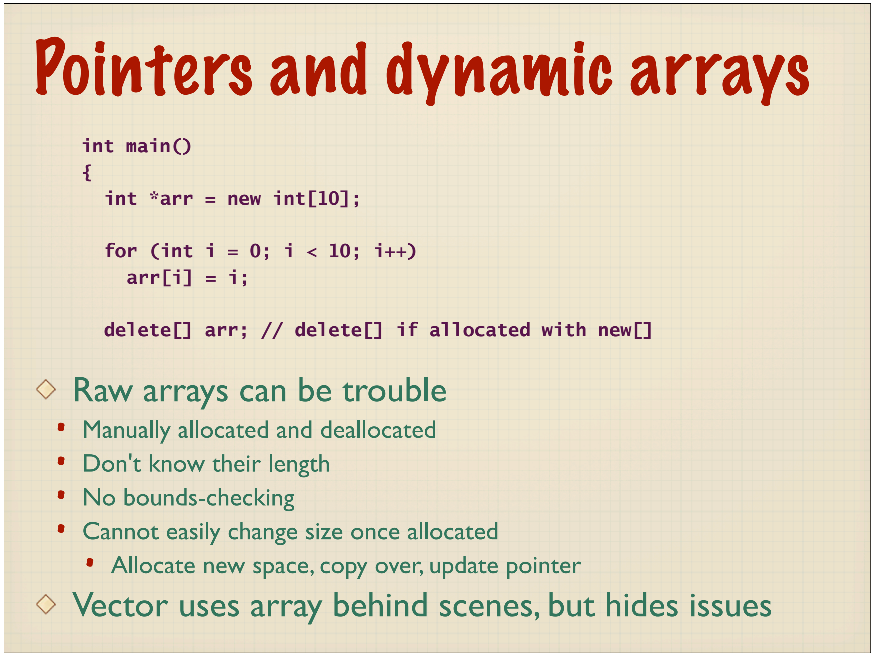
Simple pointer operations

```
int main()
{
  int num;
  int *p, *q;

  p = new int;
  *p = 10;

  q = new int;

  *q = *p;

  q = p;

  delete p;
  delete q;   // bad idea, q already deleted!

  q = NULL;   // NULL is zero pointer, used as sentinel value
```



Pointer basics

- Pointers are distinguished by type of pointee
  - Type `double*` not same as `int*`
- Pointers are uninitialized until assigned
  - Dereferencing a uninitialized pointer is bad news
- Dynamic allocation via new
  - Operator `new` allocates memory from heap, returns address
- Manual deallocation via delete
  - Forgetting to delete means memory is orphaned
  - Accessing deleted memory has unpredictable consequences



Pointers and dynamic arrays

```
int main()
{
  int *arr = new int[10];

  for (int i = 0; i < 10; i++)
    arr[i] = i;

  delete[] arr; // delete[] if allocated with new[]
```

- Raw arrays can be trouble
  - Manually allocated and deallocated
  - Don't know their length
  - No bounds-checking
  - Cannot easily change size once allocated
    - Allocate new space, copy over, update pointer
- Vector uses array behind scenes, but hides issues

Use of pointers

- Axess database

```
struct studentT {
   string first, last;
   string address, phone;
};

struct courseT {
   string dept, name;
  Vector<studentT *> students;
};
```

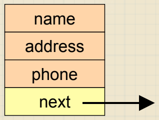
- A course has *pointers* to enrolled students
  - Allocate studentT record in heap for new student
  - Each course student enrolls in stores pointer to record
  - Saves space by not repeating student information
  - If student gets new phone number, change in one place only!

Recursive data

- Recursion applied to data
  - Self-referential, self-similar
  - Within itself, data has smaller version repeated
- Examples
  - Matroshka dolls
  - Nesting boxes
  - Onions
  - Structure containing pointer to same structure

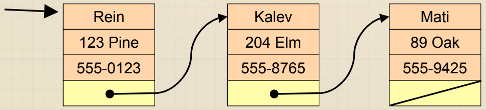
A recursive struct

```
struct Entry {
  string name, address, phone;
  Entry *next;
};
```



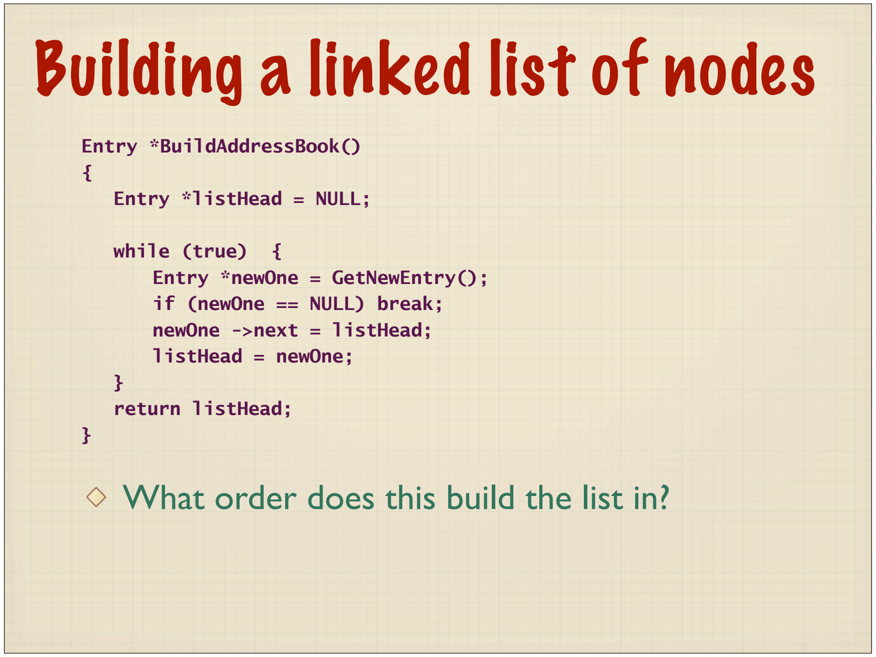
Each entry points to another Entry!

Wired together, you get a linked list!



Creating a node

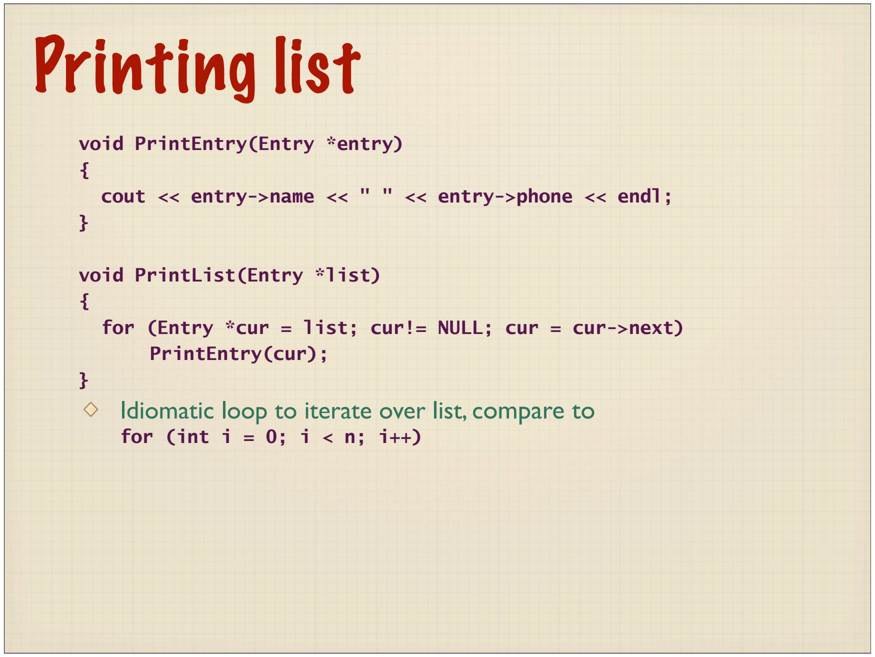
```
Entry *GetNewEntry()
{
   cout << "Enter name (ENTER to quit):";
   string name = GetLine();
   if (name == "") return NULL;

   Entry *newOne = new Entry;
   newOne->name = name;
   cout << "Enter address: ";
   newOne->address = GetLine();
   cout << "Enter phone: ";
   newOne->phone = GetLine();
   newOne->next = NULL; // no one follows
   return newOne;
}
```



Building a linked list of nodes

```
Entry *BuildAddressBook()
{
   Entry *listHead = NULL;

   while (true)  {
       Entry *newOne = GetNewEntry();
       if (newOne == NULL) break;
       newOne ->next = listHead;
       listHead = newOne;
   }
   return listHead;
}
```

- What order does this build the list in?



Printing list

```
void PrintEntry(Entry *entry)
{
  cout << entry->name << " " << entry->phone << endl;
}

void PrintList(Entry *list)
{
  for (Entry *cur = list; cur!= NULL; cur = cur->next)
       PrintEntry(cur);
}
```

- Idiomatic loop to iterate over list, compare to `for (int i = 0; i < n; i++)`