CS368 Lecture 8

Wednesday 22 October 2014

Reminders:

- P3 assigned, due Tuesday 4 Nov

Last class:

- Constructors
    - Initializer Lists
    - Default Values
- Multi-file compilation
- More on shallow copy and delete

Today:

- The “Big 3”
    - Copy Constructor
    - Destructor
    - Assignment operator overloading
- Makefiles

The Big 3 – Why?

Destructor

Copy Constructor

'=' operator

Copy Constructor

Use:

```
AddressBook ab3(ab1);
// constructor call using another AddressBook as an argument
```

Implementation:

In the header file:

```
AddressBook(const AddressBook&);
// original is passed by constant reference
```

In the source file:

```
AddressBook::AddressBook(const AddressBook& original):
    bookName(original.name), numContacts(0),
    head(NULL), tail(NULL) {

        // get the original head pointer
        ContactNode* tmp = original.head;

        // go through original contacts and add them
        while (tmp != NULL) {
            addToEnd(tmp->c);
            tmp = tmp->next;
        }

    }
```

Destructor

Use:

delete ab1;

Implementation:

In the header file:

```
~AddressBook(); // in the public section

void deleteAllNodes(); // in the private section
```

In the source file:

```
// private helper function
void AddressBook::deleteAllNodes() {
    ContactNode* current = head;
    ContactNode* tmp;

    while (current != NULL){
        temp = current;
       current = current->next;
       cout << "deleting contact "
            << temp->c.name << endl;
       delete temp;
     }

     head = NULL;
     tail = NULL;
}


// Destructor
AddressBook::~AddressBook() {
   cout << "Destructor for book " << bookName << endl;
   deleteAllNodes(); // call helper method;
}
```

'=' Operator

Overriding operators in C++ through operatorX functions

Clarity about what needs to happen.

What should get “returned”?

Use:

```
AddressBook ab3 = ab1;
// make a deep copy of ab1 and call it ab3

ab3 = ab2;
// now make a deep copy of ab2 and store it in ab3
```

Implementation:

In the header file:

```
AddressBook& operator= (const AddressBook& original);
```

In the source file:

```
          AddressBook& AddressBook::operator= (const
AddressBook& original) {

              cout << "Operator= for " << bookName << endl;

              // do not allow a self-assignment i.e. ab1 = ab1;
              if (this != &original) {

                   deleteAllNodes(); // why?

                   // do what the constructor does
                   bookName = original.bookName;
                   numContacts = 0;

                   ContactNode* current = original.head;
                   while (current != NULL) {
                        addToEnd(current->c);
                        current = current->next;
                   }
               }
               return *this;  // returns a reference
          }
```

Why don't we need to (re-)assign `head` and `tail`?

Other operators you can override: `+, -, ++, --, []`, others.

Makefiles