Lecture 05a
Header Classes
Sept. 19th, 2017

Announcements
• Quiz #4 due today (before class)
• PA1/2 Grading:
  – If something is wrong with your code, you get sympathy
• PA3 is due today (midnight)
  – Questions?
• Next week is an exam (midterm) week:
  – No Quiz next Tuesday (just a midterm 😊)
  – No programming assignment for next week
  – Recitations are OPTIONAL help sessions this week
    • Not graded
    • No new material or tools
    • Just GTAs available to give help

Midterm Format
• After class Thursday, code will appear on the class web site
  – On the assignments page
  – .h files and .cpp files
  – But no main.cpp file
• Study this code
  – Write your own main.cpp’s
  – Test the code; what does it do? How does it do it?
  – Treat this like a take-home portion of the exam
    • But with no questions, and not graded
• On Tuesday:
  – You will get the same code, plus a main.cpp file
    • Expect a big switch statement
  – And you will get questions to answer

Midterm Questions
• Expect to answer questions about
  – Constructors, copy constructors, destructors…
  – Functions, methods & parameters
  – Arrays and pointers
  – Data on stack and data on heap
  – Memory issues
  – Anything else we have covered
• int main(int argc, char* argv[])
• …

More Midterm Stuff
• My midterms are hard
  – Don’t panic, there will be a curve
  – Last year’s median: 63
• My midterms reflect the lectures more than the book
• Normal administrative stuff:
  – Bring your student ID
  – No books, notes, cell phones, laptops, neighbors
  – No hats with forward brims
  – All work must be your own
  – Come early (you will need the full time)

Memory Management Strategy #0 (Review)
• Use local variables, avoid pointers
  – Constructed when they enter scope
  – Destructed when they leave scope
• No memory leaks are possible
  – Memory reclaimed when function returns
• Invalid references almost impossible
  – Unless you return an invalid reference
Arrays

- An array is
  - A contiguous block of memory
  - Containing instances of 1 type of data
- The data type of an array is a pointer
  - Quagga* and Quagga[] are the same data type
- The Brackets [] are used to offset into arrays
  - Arr[0]: 1st element
  - Arr[1]: 2nd element
  - Arr[n-1]: Nth element

What About Vectors?

- Vector is a header class
- Header classes are Memory Management Strategy #1
- A simple vector class might look like...

```cpp
class intvector {
public:
  intvector(int sz); // allocate
  ~intvector(); // deallocate
  // Access methods (no visible ptr)
  int at(int index) const;
protected:
  int* data; //protected heap ptr
  int size; // other info
};
```

Example: An Int Vector

```cpp
intvector::intvector(int sz) : size(sz) {
  data = new int[size];
}
intvector::~intvector()
{
  delete data;
  data = NULL;
  size = 0;
}
int intvector::at(int index) const
{
  if ((index >= 0) && (index < size)) return data[index];
  else throw std::exception();
}
```

Using intvector

```cpp
#include<intvector.h>
#include<iostream>
using std::cin;

int main(int argc, char* argv[])
{
  int data_size;
  cin >> data_size; //todo: error check
  intvector ivec(data_size);
  return 0;
}
```

Header Classes

- Header classes manage heap memory
- They allocate & de-allocate memory
- Properly implemented, they allow you to treat dynamic memory like a stack variable
  - Pointers stay hidden
  - Examples include vector and string
- To do this, they rely on
  - Constructors
  - Destructors
  - The Assignment operator

"Your book calls these "The Big 3"

Example (continued)

```cpp
intvector::intvector(int sz) :
  size(sz) { 
    data = new int[size];
  }
intvector::~intvector()
{
  delete data;
  data = NULL;
  size = 0;
}
int intvector::at(int index) const
{
  if ((index >= 0) && (index < size)) return data[index];
  else throw std::exception();
}
```
Header Classes

- The goal of a header class is to make a dynamic data structure act like a local variable
  - Intvectors are local variables
    - They just have a pointer to data on the heap
  - Like all local variables, their constructor is called when they come into scope
    - Under the hood, their constructor dynamically allocates heap memory
  - When they leave scope, their destructor is called
    - Under the hood, this deletes the heap memory
  - For those of you using vectors and strings in PA1/2/3, this is what you were doing!

Improving intvector

class intvector {
public:
    intvector(int sz);
    ~intvector();
    int at(int index) const;
    int& at(int index);
protected:
    int* data;
    int size;
};

Using Both Methods

void Foo(intvector& a, const intvector& b) {
    int test;
    if (b.size() > 0) {
        test = b.at(0);
        if (a.size() > 0) {
            a.at(0) = test;
        }
    }
}

Make intvector Dynamic

class intvector {
public:
    intvector(int sz);
    ~intvector();
    int at(int index) const;
    int& at(int index);
    // creates dynamic memory
    void push_back(int value);
protected:
    int* data;
    int size;
};

Implementing push_back

void intvector::push_back(int value) {
    int* temp = data;
    data = new int[size+1];
    for(int i=0; i<size; i++) {
        data[i] = temp[i];
    }
    data[size] = value;
    size++;
    delete [] temp; // discuss
Problem: why does this crash?

```cpp
void Foo(intvector a)
{
    ...
}
void Quagga::Bar()
{
    intvector iv(0);
    iv.push_back(37);
    Foo(iv);
}
```

Copy Constructors

- When an object is passed by value, its copy constructor is called.
- Copy constructors take a constant reference to same-type object as their arguments

```cpp
Quagga::Quagga(const Quagga& src)
```

- The default copy constructor copies the values of the parameter to the new instance
  - Including pointers
- Fortunately, you can redefine the copy constructor

Fixing intvector

```cpp
class intvector {
public:
    intvector(int sz);
    intvector(const intvector& src);
    ~intvector();
    int at(int index) const;
    int& at(int index);
    void push_back(int value);
protected:
    int* data;
    int size;
};
```

Implementing the Copy Constructor

```cpp
intvector::intvector(const intvector& src)
{
    size = src.size;
    data = new int[size];
    for(int i=0; i < size; i++) {
        data[i] = src.data[i];
    }
}
```

- Are these legal? Yes. Size and data are protected, which means only intvectors can access them.
  - It would be legal if they were private, too.

Elements of Copy Constructors

- Pretty much every copy constructor will:
  - Check the size of the object being copied
  - Allocate new memory of that size
  - Copy data from source to destination
- Note: Copy constructors do not delete data

Assignment Operators

```cpp
class intvector {
public:
    intvector(int sz);
    intvector(const intvector& src);
    ~intvector();
    intvector& operator = (intvector& src);
    int at(int index) const;
    int& at(int index);
    void push_back(int value);
protected:
    int* data;
    int size;
};
```
Properties of Assignment Operator

• Called by =
  – e.g. quagga1 = quagga2;

• Pretty much every assignment operator will:
  – Delete existing memory of destination
  – Note: the destination is the hidden argument
  – Allocate new memory (size of source)
  – Copy data from source
  – Bookkeeping for destination
  – Return reference to destination (for chaining)

• Similar to copy constructor, but with a deletion step first

Assignment Operator

```cpp
intvector& intvector::operator = (const intvector& src)
{
    size = src.size;
    delete [] data;
    data = new int[size];
    for(int i = 0; i < size; i++)
        data[i] = src.data[i];
    return *this;
}
```

Memory Management Strategy #1

• Used for dynamic data
  – Does not solve problem of persistence

• Define a header class
  – Header constructors allocate memory with new
    • Must include a copy constructor
    • At least one other constructor
  – Header destructor delete memory
  – Header assignment operator does both
  – Optionally, other methods may play with memory
    • Must balance a delete and a new
    • e.g. push_back

MM Strategy #1 (cont.)

• Header classes encapsulate data & code
  – Data structure & its code isolated from the application
  – Pointer(s) are protected or private

• News and deletes are balanced
  – Every constructor calls new
  – Every destructor calls delete
  – Assignment calls both
  – Other methods call both or none

• Use header classes as local variables
  – Constructors & destructors called automatically