Lecture 05
Header Classes

February 17th, 2016
Announcements

• PA3 is due on today
  – Any questions?

• Reading assignment & quiz for today
  – Unusual, so don’t miss it!
  – Numbered Quiz 4a, covers Sections 4.1 – 4.5
  – Quiz 4b will be due Monday…

• Midterm is a week from Wednesday
  – Code handed out Monday

• ACM meeting (6:00 in CSB 130)
  – 3 million lines of code every 6 weeks
Revisiting const

- Imagine the following method declaration:
  ```
  Class Quagga {
    Quagga& Breed(Quagga& q2);
  };
  ```

- How many places in the declaration of Quagga::Breed could the keyword const appear?
const Quagga& Breed(const Quagga& q2) const;

• Const before a parameter says that the parameter cannot be changed inside the method
  – Very common for pass by reference parameters
• Const after the parameter list says that the object it is called on cannot be changed inside the method
  – This makes the method an accessor
  – The hidden argument ‘*this’ cannot be changed
• Const before the return value says that the return value cannot be changed after the method returns
  – Only common for returned references
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
• To do this, they rely on*
  – Constructors
    • Including copy constructors
  – Destructors
  – The Assignment operator

* Your book calls these “The Big 3”
Example: An Int Vector

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

```cpp
intvector::intvector(int sz) : size(sz)
{
    data = new int[size];
}

intvector::~intvector()
{
    delete data;  // this is a bug, but will discuss later
    data = NULL;
    size = 0;
}

int intvector::at(int index) const
{
    if (index < size) return data[index];
    throw std::exception();
}
```

2/18/16 CS253 Fall 2016 © Ross Beveridge & Bruce Draper
Using intvector

#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;
}

Constructor allocates memory on heap

Destructor deletes memory on heap
Improving intvector

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

• Two methods with the same name and arguments?
  – The hidden argument isn’t the same
    • It is of type const intvector in one
    • It is of type intvector in the other

• Why declare both methods?
  – The 1\textsuperscript{st} lets you get value from a constant vector
  – The 2\textsuperscript{nd} lets you modify a non-constant vector
Using Both Methods

```cpp
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);
    void push_back(int value);
protected:
    int* data;
    int size;
};
Implementing push_back

```cpp
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

    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

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.
One more thing…

• What happens in the following case?

```c
intvector a(5);
intvector b(7);

a = b;
```

• We need to redefine assignment to
  – Delete old memory
  – Allocate new memory
  – Copy data
Defining Assignment

• For the moment, just trust me on the syntax

```cpp
intvector& operator = (const intvector& src)
{
    if (size != src.size) {
        size = src.size;
        delete [] data;
        data = new int[size];
    }
    for(int i=0; i < size; i++) {
        data[i] = src.data[i];
    }
}
```
Header Class Summary

- Header classes manage heap data
- Goal: treat heap data like stack data
- Every header class needs three things:
  - “Deep” constructors that allocates memory via new and copies data
    - Including a copy constructor
  - “Deep” destructor that deletes memory
  - An assignment operator that does both
    - Deletes old memory
    - Allocates new memory and copies data