Lecture 07
Objects & Memory
February 29th, 2016

Announcements
• Recitations: optional
  – Mandatory
  – Topic: Valgrind memcheck
• PA4 is due on Wednesday
  – Any questions?
• Expect reading and a quiz for Friday.
  – In general, pace of quizzes will slow down
  – More programming instead!
• Midterm #1 was last week
  – Comments, questions or complaints?

Emailed Question
(paraphrased)
• “Should I re-use my distance computation from PA3, or write a new one”
• Take away all the details, there are only three skills in programming:
  – Breaking a task down into parts
  – Testing
  – Organizing parts via abstraction
• Computer science is the science of complexity

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

Implementing push_back (review)

```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
}
```

What about efficiency?
• Invector acts lack a dynamic variable
• But it copies the whole vector with every push_back.
  – Pushing back 10 items does 45 copies
    • 45 copy constructors
    • 10 user-specified constructors
    • 45 destructors
  – Pushing back N items does \( \sum_{i=1}^{n} (i-1) \) copies
    – Calls \( \sum_{i=1}^{n} (i-1) \) copy constructors
    – Calls \( \sum_{i=1}^{n} (i-1) \) destructors
• Is there a better way?
Efficient intvector

- Allocate arrays that are bigger than needed
  - Separate size (number of elements)
  - From capacity (amount of memory allocated)
- When there is extra capacity, push_back doesn't have to copy or allocate
- When there isn't, push_back doubles the capacity of the array
  - Why double it?

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

New implementations
intvector::intvector() : size(0), capacity(10) {
  // harder if data is not primitive
  data = new int[capacity];
}
intvector::intvector(int sz) : size(sz), capacity(sz) {
  data = new int[size];
}
Intvector::~intvector() {
  delete [] data;
  data = NULL;
  size = 0;
  capacity = 0;
}

void intvector::reserve(int sz) {
  if (sz > capacity) {
    int* temp = data;
    data = new int[sz];
    for(int i=0; i < size; i++) {
      data[i] = temp[i];
    }
    delete [] temp;
    capacity = sz;
  }
}

intvector::push_back(int value) {
  if (size == capacity) {
    int* temp = data;
    capacity *= 2;
    data = new int[capacity];
    for(int i=0; i < size; i++) {
      data[i] = temp[i];
    }
    delete [] temp;
  }
  data[size] = value;
  size++;
}

Vectors of Objects

- Vectors of objects are a little different
  - Remember that ‘new’ does two things:
    - Allocate memory (malloc, or something similar)
    - Invoke the constructor
  - Delete also does two things
    - Call the destructor
    - Deallocate memory (free, or something similar)
  - Vectors separate memory (de)allocation from object creation/destruction
  - The assignment in push_back invokes an assignment operator.
  - The vector destructor has to explicitly destruct the components of the vector.
Quaggavector destructor

quaggavector::~quaggavector()
{
    for(int i=0; i < size; i++) {
        data[i].~Quagga();
    }
    delete[] data;
    data = NULL;
    size = 0;
    capacity = 0;
}