CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 10 Feb 6, 2013

Vectors in C++
Heap Sort

Upcoming Homework Assignment

HW #3 **Due: Friday, Feb 8**

Sorting Algorithms

Bubble Sort, Merge Sort, Quick Sort, Mystery Sort!

RG will be deployed *very late in the day* on Wednesday. Should have lots of improvements when the HW graders are released. I'll tweet when I put them out. Until then, use those drivers on your side.

Lecture Goals

1. About C++ Vectors
2. Pass By Reference
3. Heap Sort

None of this is “required”

All the topics today are for your information only. Not going to test on these things, but it will definitely help you in doing this assignment and many subsequent ones.

Vectors in C++

A Vector is a math term for a multidimensional collection of data.

In C++, a Vector is sort of like that. It is a collection of data that are all of the same sort. You can't necessarily use it like mathematical vectors (there's no dot or cross product).

But you can use them like magic arrays that simplify working with lists of data.

#include <vector>

To use a Vector you need to #include<vector>.

A C++ vector is a templated class. In Java you'd say it was a *generic*. This means you have to give the type of the elements it will contain.

This is how you create a new Vector of integers:

```
std::vector<int> some_numbers;
```

Creating Vectors

```
8:  vector<int> my_vec();
9:  my_vec.push_back(10);
```

```
vec.cpp:9: error: request for member
'push_back' in 'my_vec', which is of non-class
type 'std::vector<int, std::allocator<int> > ()()'
```

The bug is actually line 8. Don't use parens:

```
vector<int> my_vec;
```

Stack vs. Heap Allocation

```
// this program outputs 10, then 20.
#include <vector>
#include <iostream>

using namespace std;

int main() {
  // allocate a spot on the stack for my_vec
  vector<int> my_vec;
  my_vec.push_back(10);
  cout << my_vec[0] << endl;

  // allocate memory from the heap for other_vec
  vector<int>* other_vec = new vector<int>;
  other_vec->push_back(20);
  cout << (*other_vec)[0] << endl;
}
```

Notice the different way we use push_back: dot vs. arrow.

Useful Vector methods

Current list length	`vec.size()`
Put value at position	`vec.insert(pos, val)`
Assign into vec.	`vec[3] = x`
Read vector position	`y = vec[7]`
Add to end of list	`vec.push_back(num)`
Remove end of list	`vec.pop_back()`
Test if vector is empty	`vec.empty()`

Vector Iterators

```
for (vector<int>::iterator it = my_vec.begin();
       it != my_vec.end(); it++)
{
    cout << "value is: " << *it << endl;
}
```

An iterator refers to a spot in the vector. You can increment it to move to the next spot, or dereference it to get the value at that spot.

```
value is: 3
value is: 28
value is: 4
value is: 74
```

Iterator Names

There are a couple iterators that might be handy:

`begin`	Returns an iterator that initially points to the beginning of the vector.
`end`	Actually refers to the element one past the end.

Also notice that some functions like **`insert`** take an iterator, not an int, to specify position. Don't blame me, I didn't write the spec.

Pass By Reference

```
void vegas(vector<int> data) {
  data.push_back(999);
}

void change_it(vector<int> &data) {
  data.push_back(444);
}

vector<int> my_vec;
cout << my_vec.size() << endl;  → 0
vegas(my_vec);
cout << my_vec.size() << endl;  → 0
change_it(my_vec);
cout << my_vec.size() << endl;  → 1
```

PBR is good!

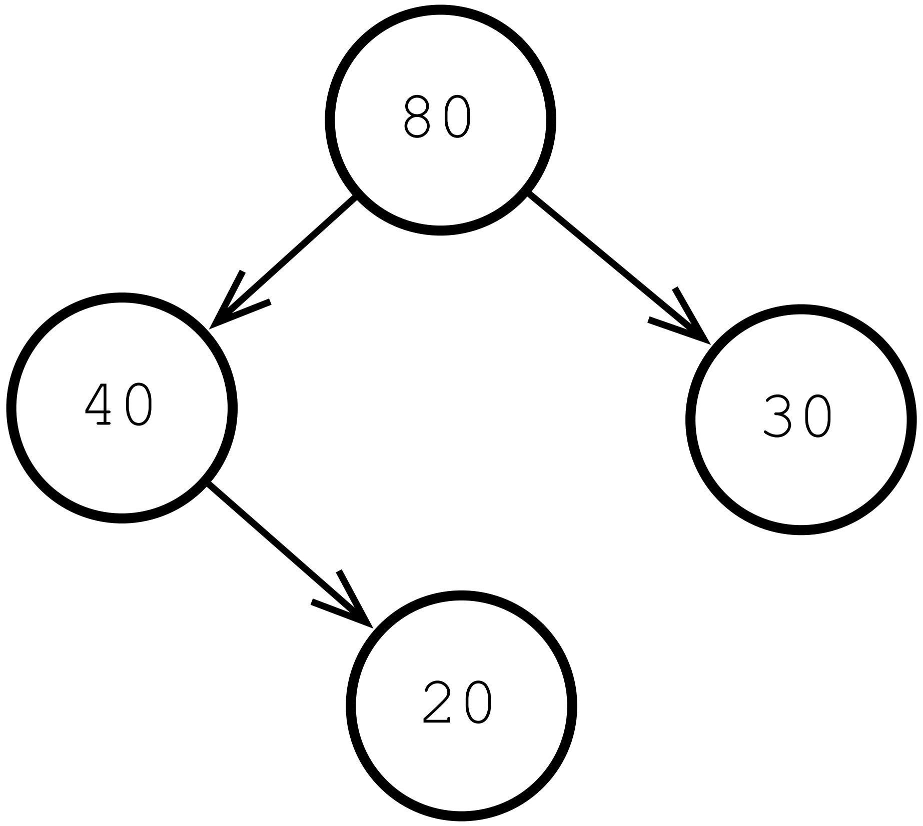
Some recursive functions benefit from using pass-by-reference. Say we want to traverse a binary tree and report the value *and inorder index* of a node:

```
void dive(bt_node* &node, int &idx) {
  if (node == NULL) return;
  dive(node->left, idx);
  cout << "idx " << idx << " = "
       << node->value << endl;
  idx++;
  dive(node->right, idx);
}
```

Going Diving with PBR

```
void dive(bt_node* &node, int &idx) {
  if (node == NULL) return;
  dive(node->left, idx);
  cout << "idx " << idx << " = "
       << node->value << endl;
  idx++;
  dive(node->right, idx);
}
```

```
idx 0 = 40
idx 1 = 20
idx 2 = 80
idx 3 = 30
```



Note: this is a heap

Heaps and Heapsort

Please see HeapSort.pdf in the hw3 directory...