CSCI 1300

Intro to Computing

Gabe Johnson

Lecture 40 April 24, 2013

C++ Vectors and Pass-By-Reference

Lecture Goals

1. C++ Vectors
2. Pass by reference

Upcoming Homework Assignment

HW #9 **Due: Friday, Apr 26**

Linked Lists

Two versions due friday:

— A C-style implementation for 30 points (out of 10, so you can get 20pt extra credit)
— A C++-class implementation for 10 points of pure extra credit.

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()`

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);
}
```

PBR code

Please see:

cs1300/code/cpp/pbr.cpp

for the code we did in class.