Lecture 28: More Vector Operations

Announcements and reminders

- Project 2 posted, due **Saturday March 23 by 6 PM**
  (no early bonus aside from being a badass)

Last time on *Intro Computing…*

We met **vectors!**

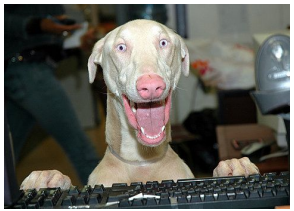
- Like a 1d array whose size can change
- Get current size of vector `vec` using: `vec.size()`
- Add elements to the ***back*** using: `vec.push_back()`
- Remove elements from the ***back*** using: `vec.pop_back()`
- Passing vectors into/out of functions
- … and using as return values

Last in first out (LIFO) →

Arrays… some drawbacks

S'pose you have two arrays: `int your_money[5] = { 0, 18, 7, 43, 4 };`
`int my_money[5];`

And further, s'pose we want what is stored in your_money to become my_money



… let's just say.

With arrays, we can **not** simply do this: `my_money = your_money;`

Instead, we must loop:

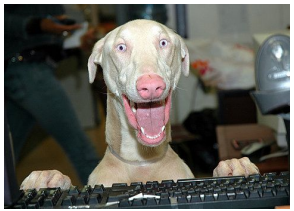
```
for (int i=0; i < 5; i++) {
    my_money[i] = your_money[i];
}
```

Arrays… some drawbacks

S'pose you have two arrays:

```
vector<int> your_money;
vector<int> my_money;
```

And further, s'pose we want what is stored in `your_money` to become `my_money`



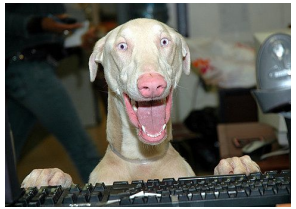
… let's just say.

With vectors, we **can** simply do this: `my_money = your_money;`

Question: But how would we get the values { 0, 18, 7, 43, 4 } stored in `your_money` ?

Vectors -- initialization

We can also initialize vectors like we have initialized arrays:



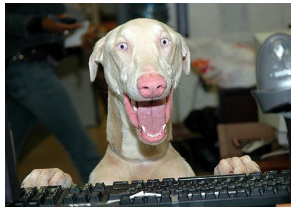
```
vector<int> your_money{ 0, 18, 7, 43, 4 };
```

… is equivalent to…

```
vector<int> your_money;
your_money.push_back( __ );
your_money.push_back( __ );
your_money.push_back( __ );
your_money.push_back( __ );
your_money.push_back( __ );
```

Vectors -- initialization

We can also initialize vectors like we have initialized arrays:



```
vector<int> your_money{ 0, 18, 7, 43, 4 };
```

… is equivalent to…

```
vector<int> your_money;
your_money.push_back(0);
your_money.push_back(18);
your_money.push_back(7);
your_money.push_back(43);
your_money.push_back(4);
```

Common algorithms -- finding matches

S’pose we want to keep all values from an array that are greater than a certain value, say, 100.

How could we do this with arrays?

- Create a second array
- … same size as the original
- Loop over it, and copy all elements that meet the condition

Drawback: new array is same size as old one (maybe only partially filled)

Common algorithms -- finding matches

S’pose we want to keep all values from an array that are greater than a certain value, say, 100.

How could we do this with arrays?

- Create a second array
- … same size as the original
- Loop over it, and copy all elements that meet the condition

Drawback: new array is same size as old one (maybe only partially filled)

Better idea: this is MUCH easier with vectors!

→ **Reflect:** *why?*

Common algorithms -- finding matches

S’pose we want to keep all values from an array that are greater than a certain value, say, 100.

```
// input:    double scores[SIZE];   an array of scores of size SIZE
vector<double> overachievers;
for (int i=0; i < SIZE; i++) {
    if (scores[i] > 100) {
        overachievers.push_back(scores[i]);
    }
}
```

Common algorithms -- removing an element, unordered

S'pose we want to remove an element from a vector `values` ***and*** the order of the vector `values` elements is **not important**. Then we could…

- Find the position of the element we want to remove (call it index `i_rem`)
- Overwrite that element with the last one from the vector
- Remove the last element from the vector
  (makes the vector smaller by 1)

Handy member function: **`[vec].back()`** -- returns the last element of a vector (doesn't pop it)

Common algorithms -- removing an element, unordered

S'pose we want to remove an element from a vector `values` ***and*** the order of the vector `values` elements is **not important**. Then we could…

- Find the position of the element we want to remove (call it index `i_rem`)
- Overwrite that element with the last one from the vector
- Remove the last element from the vector
  (makes the vector smaller by 1)

Handy member function: **`[vec].back()`** -- returns the last element of a vector (doesn't pop it)

```
// first, need to loop over to find i_rem
values[i_rem] = values.back();
values.pop_back();
```

Common algorithms -- removing an element, ordered

S'pose we want to remove an element from a vector `values` ***and*** the order of the vector `values` elements ***is*** **important**. Then we could…

- Find the position of the element we want to remove (call it index `i_rem`)
- Overwrite that element with the next one from the vector (`values[i_rem+1]`)
- Overwrite the next element with the one after that (`values[i_rem+2]`)... and so on.
- Remove the last element from the vector
  (makes the vector smaller by 1)

Common algorithms -- removing an element, ordered

S'pose we want to remove an element from a vector `values` ***and*** the order of the vector `values` elements ***is*** **important**. Then we could…

- Find the position of the element we want to remove (call it index `i_rem`)
- Overwrite that element with the next one from the vector (`values[i_rem+1]`)
- Overwrite the next element with the one after that (`values[i_rem+2]`)... and so on.
- Remove the last element from the vector
  (makes the vector smaller by 1)

```
// first, need to loop over to find i_rem
for (int i=i_rem; i<(values.size()-1); i++) {
    values[i] = values[i+1];
}
values.pop_back();
```

Common algorithms -- inserting an element, unordered

S'pose we want to insert an element into a vector `values` ***and*** the order of the vector `values` elements ***is not*** **important**. Then we could…

- Slap the new element (`noob`) onto the end of our vector!

```
values.push_back(noob);
```

Common algorithms -- inserting an element, ordered

S’pose we want to insert an element into a vector `values` ***and*** the order of the vector `values` elements ***is*** **important**. Then we could…

… basically do our algorithm for removing an element, but in reverse.
S’pose we have `i_ins` as the index we want the inserted element to be at

Common algorithms -- inserting an element, ordered

S'pose we want to insert an element into a vector `values` ***and*** the order of the vector `values` elements ***is*** **important**. Then we could…

… basically do our algorithm for removing an element, but in reverse.
S'pose we have `i_ins` as the index we want the inserted element to be at

- Add the last element to the ***new last element slot***

```
values.push_back(values.back());     // now vector is one size larger!
```

- Move the third-to-last element into the second-to-last slot
- Move the fourth-to-last element into the third-to-last slot … and so on.
- Place the new element at `i_ins` after all those after `i_ins` are shifted backward to make room

Common algorithms -- inserting an element, ordered

S'pose we want to insert an element into a vector `values` ***and*** the order of the vector `values` elements ***is*** **important**. Then we could…

… basically do our algorithm for removing an element, but in reverse.
S'pose we have `i_ins` as the index we want the inserted element to be at

- Add the last element to the ***new last element slot***

```
values.push_back(values.back());     // now vector is one size larger!
```

- Move the third-to-last element into the second-to-last slot
- Move the fourth-to-last element into the third-to-last slot … and so on.

```
for (int i=values.size()-2; i>i_ins; i--) {
   values[i] = values[i-1];
}
values[i_ins] = noob;    // place the new element after those after it are shifted back
```

Vectors -- other member function

Vectors are a **class**. They have **lots of member functions,** so we don't have time to go through each and every one.

You've got the highlights. Feel free to explore others!

http://www.cplusplus.com/reference/vector/vector/

Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Arrays or vectors?

Short answer: Vectors are usually easier, and more flexible.

- Can grow/shrink as needed
- Don’t have to keep track of their size in a separate variable (vec.size())

But arrays are often more efficient. So beefier programs typically use arrays

Vectors. Ahhh! That's the stuff.

We just saw… **vectors!**

- Like a 1d array whose size can change
- Other ways to initialize vectors
- Some vector algorithms:
  - Finding matches within an array (or other vector)
  - Inserting an element into a vector
  - Removing an element from a vector

Now, a bit of a chat about Project 3!