PIC 10A

Lecture 19: More about Vectors

Passing vectors to functions

We have already seen how to pass vectors to functions. However, the method we have used has some disadvatages.

We have passed vectors by value in all previous examples.

What if the vector is very big?

What if we want any changes to our vector to carry over to the calling function?

Passing vectors by reference

Passing a vector by reference is just like passing any other variable by reference:

```
void foo (vector<int> &numbers);
```

Passing vectors by reference is much more efficient.

Only problem is that all changes to the vector will be permanent.

If your function does not change the vector we can add a little bit on insurance:

```
void foo(const vector<int> &numbers);
```

Finding elements in a vector

Search through a vector and return the first position where value occurs.

```
int find(const vector<int> &list, int value)
{
    int i = 0;

    while (i < list.size() && list[i] != value)
    {
        i++;
    }
    return i;
}
```

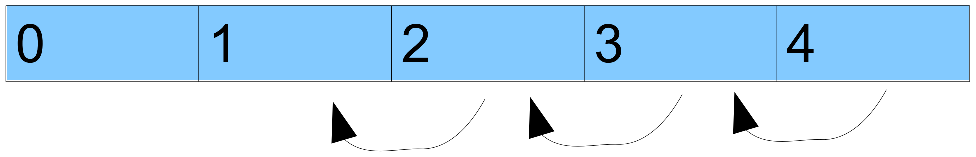
Erasing elements

Given a vector v and an index i we want to erase an element at the index i.

Strategy: Say we are told to erase slot 1.

0	1	2	3	4

First we will copy over to the 1 slot the contents of the two slot and then to 2 slot the contents of the 3 slot and so on.



0	2	3	4	4

Problem is that the last slot is now repeated. So we simply pop the last slot!

Erase function implementation

```
void erase(vector<double> &v, int pos)
{
   for (int i = pos; i < v.size() - 1; i++)
      v[i] = v[i+1];
   v.pop_back();
}
```

Inserting an element

Given a vector v and an index i we want to insert an element at the index i.

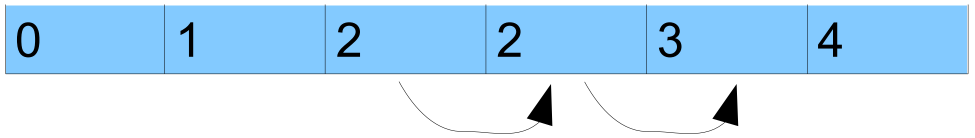
Strategy: Say we are told to insert to slot 3 value 5.

0	1	2	3	4

First we will push to the end of the array the last element so it occurs twice at the end.

0	1	2	3	4	4

Then starting at the end of the vector we work backwards and we copy values over to the next slot.



Final step is to over write the insert slot with the insert element.

0	1	5	2	3	4

Implementation

```
void insert(vector<double>& v, int pos, double x)
{
   int last = v.size() - 1;
   v.push_back(v[last]);
   for (int i = last; i > pos; i--)
      v[i] = v[i - 1];
   v[pos] = x;
}
```

Putting it together

Assume we have a vector of strings called words. We will ask a user to enter a word and then erase all occurances of word in words vector.

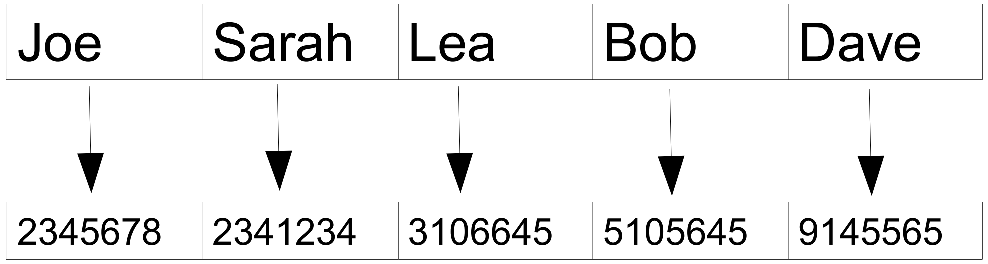
```
int find_pos = find(words,word);
while (find_pos >= 0)
{
   erase(words, find_pos);
   find_pos = find(words,word);
}
```

Parallel vectors

Assume we want to keep track of a list of names and corresponding phone numbers.

Names are strings and phone numbers are ints. We can not put all this data directly into a same vector since all elements in a vector need to have the same data type.

One solution is to create two vectors, one for names and the other for numbers. We will also organize the data so that it matches up by the slots in the vector. So that if Joe is in slot 0 for names, Joe's number is in slot 0 for numbers.



Parallel vectors example

```
vector<string>names;
vector<int>numbers;

string name;
int number;

for (int i=0; i < 10; i++)
{
    cin >> name;
    cin >> number;
    names.push_back(name);
    numbers.push_back(number);
}

for (int i=0; i < 10; i++)
{
    cout << "Name:  " <<  names[i] <<"Number:  " << numbers[i]
            << "\n";
}
```

Parallel vectors example

Parallel vectors can be annoying to deal with. You need to remember to update two arrays instead of one.

A better solution is to make class

```
class person
{
public:

// constructors and member functions

private:
string name;
int number;

};
```

Parallel vectors example

Then we make a vector:

```
vector <Person> phone_book;
```

We would then fill this vector `phone_book` with instances of the `Person` class.

Now we only need to work with one vector.

2D vectors

By 2D vector we mean a vector that is storing the data for a matrix or a table.

1.2	5.6	-3.2	0.7
6.8	0	4.2	3.3
-7	2.1	3.9	2.2

How can we get a vector to store this data?

One way is to just make a vector with 12 elements where first 4 elements make up the first row next 4 the second row and last 4 elements store the last row.

This is pretty clumsy!

2D vectors cont.

Second method is to make a vector of vectors.

We will make a vector where each element of the vector is itself a vector.

```
vector <vector<int>> table(3);
```

Each element of the table vector will be a row vector.

```
vector <int> row(4);

for (int i=0; i < 3; i++)
   table[i] = row;
```

2D vectors cont.

Now we have created an unfilled 3x4 table. Lets fill it.

```
int value;

for (int i=0; i < 3; i++)
{
   for (int j=0; j < 4; j++)
   {
      cout << “Enter a value: ”;
      cin >> value;
      table[i][j] = value;
   }
}
```

Shuffling a vector

Lets say we are trying to shuffle a vector with 10 elements.

Our strategy for shuffling a vector is to randomly pick two slots in a vector and swap their contents.

Of course this only shuffles two elements. We will keep switching two random slots 100 times and this should give us a pretty well shuffled vector.

Our swap function is important so lets write it:

```
void swap(int &a, int &b)
{
   int temp=a;
   a=b;
   b=temp;
}
```

Shuffling a vector

```
vector<int>v(10);

//We fill v;

for (int i=0; i < 100; i++)
{
   slot1 = rand() % 10;
   slot2 = rand() % 10;

   swap(v[slot1],v[slot2]);
}
```

Sorting a vector

The `<algorithm>` library contains a sort function

Given that `v` is a vector. We use it like this:

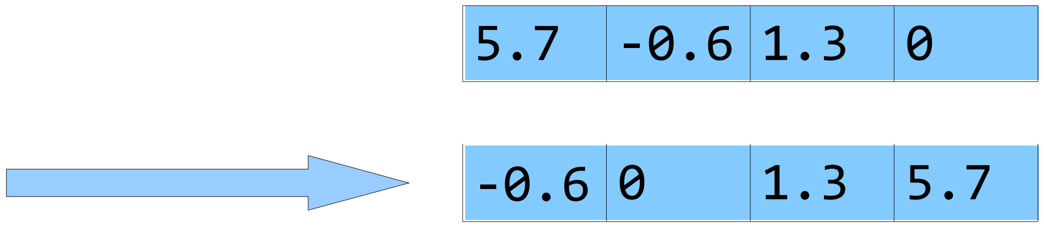
```
sort(v.begin(), v.end());
```

```
#include <algorithm>
...
   vector<int> v(4);
   v[0]=42; v[1]=12; v[2]=37; v[3]=2;

   sort( v.begin(), v.end() );

   for (int i = 0; i < v.size(); i++)
      cout << v[i] << " ";
```

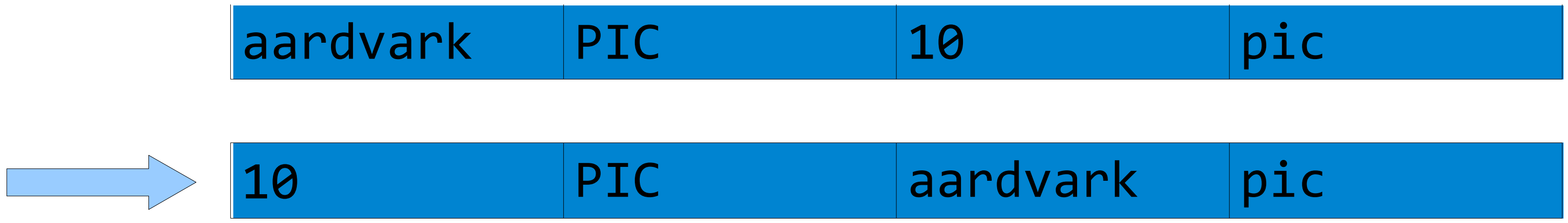
`OUTPUT: 2 12 37 42`

Sort a vector cont.

We can also sort a vector of doubles:



Or even strings:



How about sorting a vector of Persons?

We cannot sort it unless we define < for Person class.