# COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300

Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 26

## Announcements

- Rec 11 due on 4/7
- Hmwk 8 (Project 3)
  - Classes & Code Skeleton – due 4/8
  - Final deliverables – due 4/22
- Practicum III – 4/11
  - 6pm option in Duane 1B20??
  - scheduler open for 100 spots

## Agenda

- Today:
  - Vectors

## Arrays – One Drawback

The size of an array *cannot* be changed after it is created.

You have to get the size right – *before* you define an array.

The compiler has to know the size to build it.
and a function must be told about the number
elements and possibly the capacity.

It cannot hold more than it's initial capacity.

Wouldn't it be good if there were something that never filled up?

## Vectors

A *vector*

- is not fixed in size when it is created

  and

- it does not have the limitation of needing an auxiliary variable

  AND

- you can keep putting things into it

… forever!

Well, conceptually forever. (There’s only so much RAM.)

## Defining Vectors

When you define a vector, you
must specify the type of the elements.

$\downarrow$

```
vector<double> data;
```

Note that the element type is enclosed in *angle* brackets.

`data` can contain *only* `double`s

## Defining Vectors

By default, a vector is empty when created.

```
vector<double> data; // data is empty
```

# Defining Vectors

You can specify the initial size.

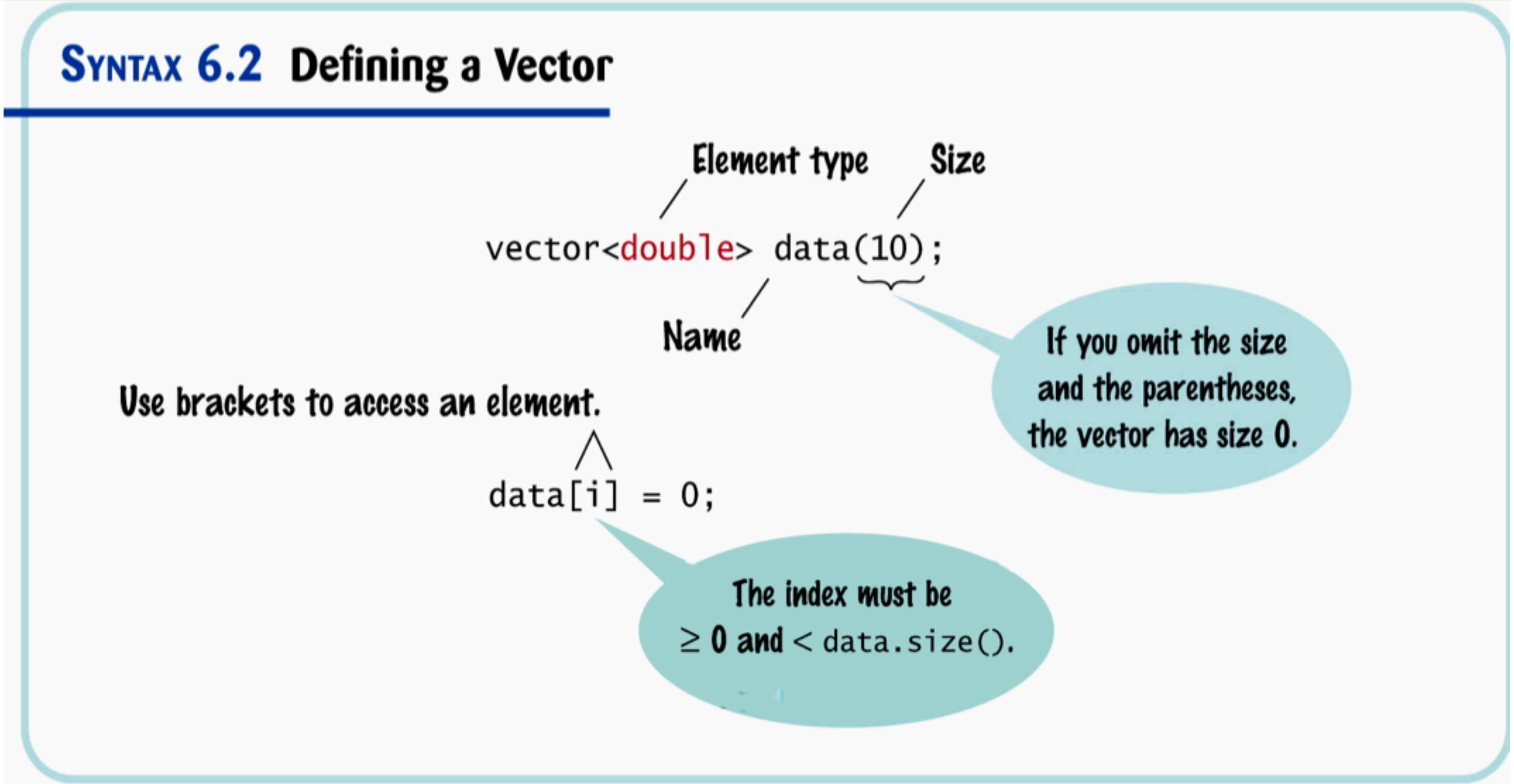
You still must specify the type of the elements.

For example, here is a definition of a vector of `double`s whose initial size is `10`.

```
vector<double> data(10);
```

This is very close to a `data` *array* definition.

# Defining Vectors



**Syntax 6.2 Defining a Vector**

```
vector<double> data(10);
```

```
data[i] = 0;
```

Element type
Size
Name
If you omit the size and the parentheses, the vector has size 0.
Use brackets to access an element.
The index must be ≥ 0 and < data.size().

# Defining Vectors

| | | |
|---|---|---|
| | `vector<int> numbers(10);` | A vector of ten integers. |
| | `vector<string> names(3);` | A vector of three strings. |
| | `vector<double> values;` | A vector of size `0`. |
| 🚫 | `vector<double> values();` | **Error:** Does not define a vector. |
| | `vector<int> numbers; for (int i = 1; i <= 10; i++) { numbers.push_back(i); }` | A vector of ten integers, filled with 1, 2, 3, ..., 10. |
| | `vector<int> numbers(10); for (int i = 0; i < numbers.size(); i++) { numbers[i] = i + 1; }` | Another way of defining a vector of ten integers and filling it with 1, 2, 3, ..., 10. |

## Accessing Elements in Vectors

You access the elements in a vector
the same way as in an array, using an index.

```
vector<double> values(10);
//display the forth element
cout << values[3] << end;
```

HOWEVER…

# Accessing Elements in Vectors

It is an error to access a element that is not there in a vector.

EMPTY!

```
vector<double> values;
//display the forth element
cout << values[3] << end;
```

ERROR!

## `push_back`

So how do you put values into a vector?

You push ’em—

—in the back!

The method *`push_back`* is used to put a value into a vector:

```
values.push_back( 32 );
```

## `push_back` and `pop_back`

```
values.push_back( 32 );
```

adds the value `32.0` to the vector named `values`.

The vector increases its size by 1.

## `pop_back`

And how do you take them out?

You pop ’em!

—from the back!

The method *`pop_back`* removes the last value placed into the vector with `push_back`.

```
values.pop_back();
```

## `push_back` and `pop_back`

```
values.pop_back();
```

removes the last value from the vector named `values`

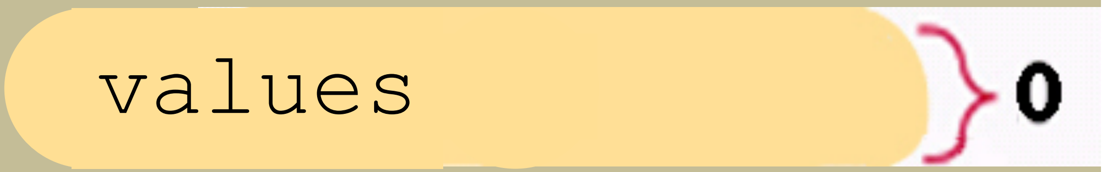
and the vector decreases its size by 1.

## `push_back` Adds an Element

```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

## `push_back` Adds an Element

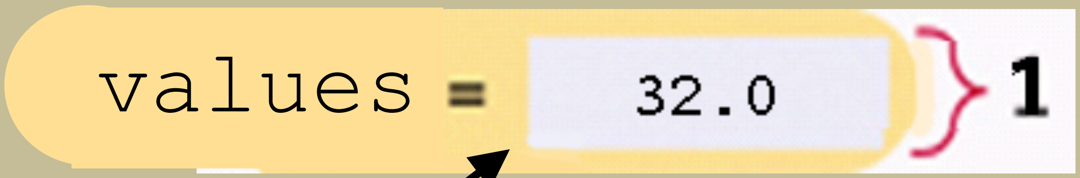


```
vector<double> values;
```

`values` is an empty vector.
Its size is 0.

```
values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

## `push_back` Adds an Element

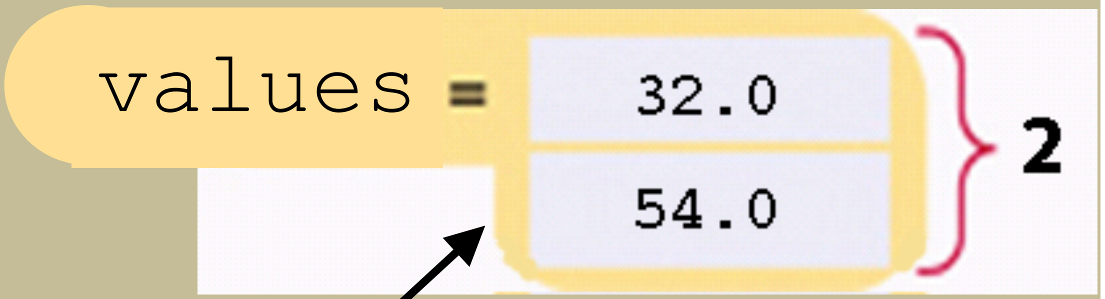


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

32 is placed into the vector.
Its size is now 1.

# `push_back` Adds an Element

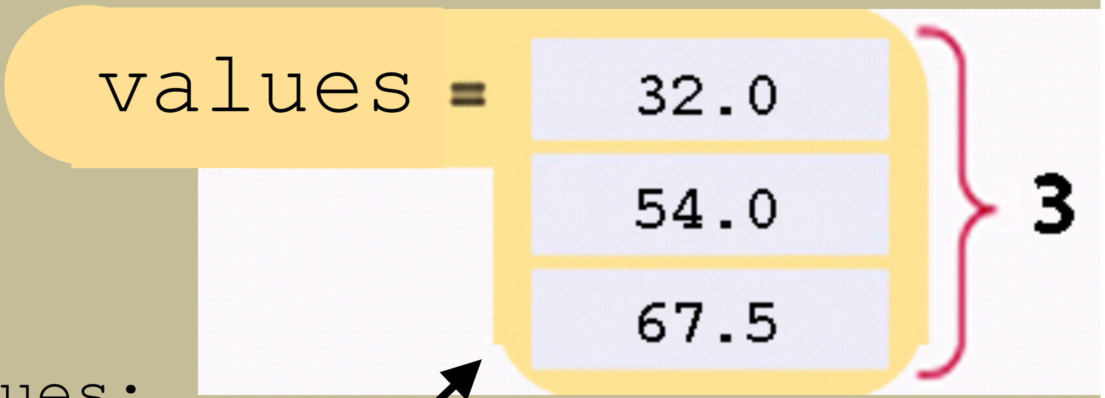


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`54` is placed into the vector.
It now contains the elements
`32.0` and `54.0`,
and its size is `2`.

## `push_back` Adds an Element

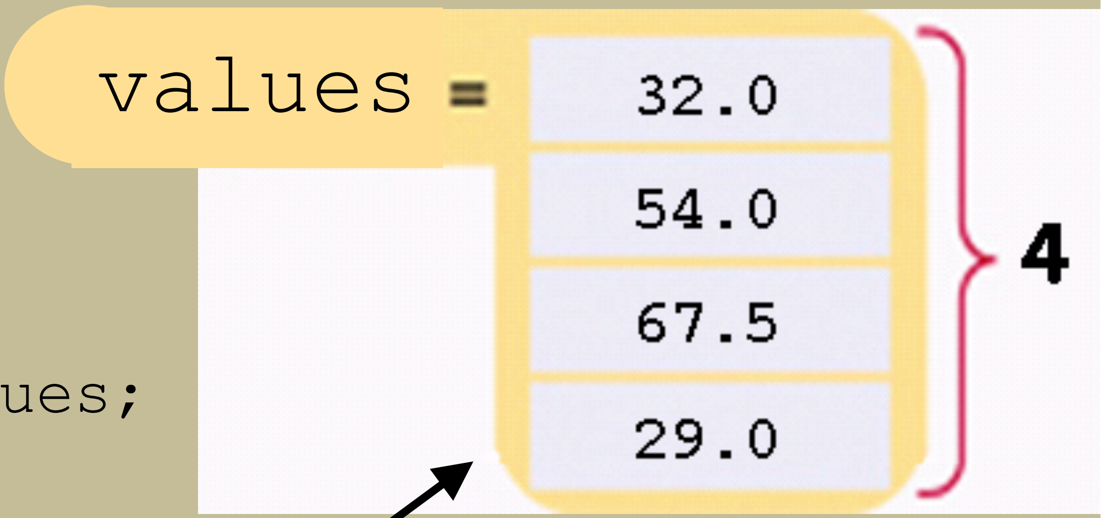


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`67.5` is placed into the vector. It now contains the elements `32.0`, `54.0` and `67.5`, and its size is `3`.

## `push_back` Adds an Element

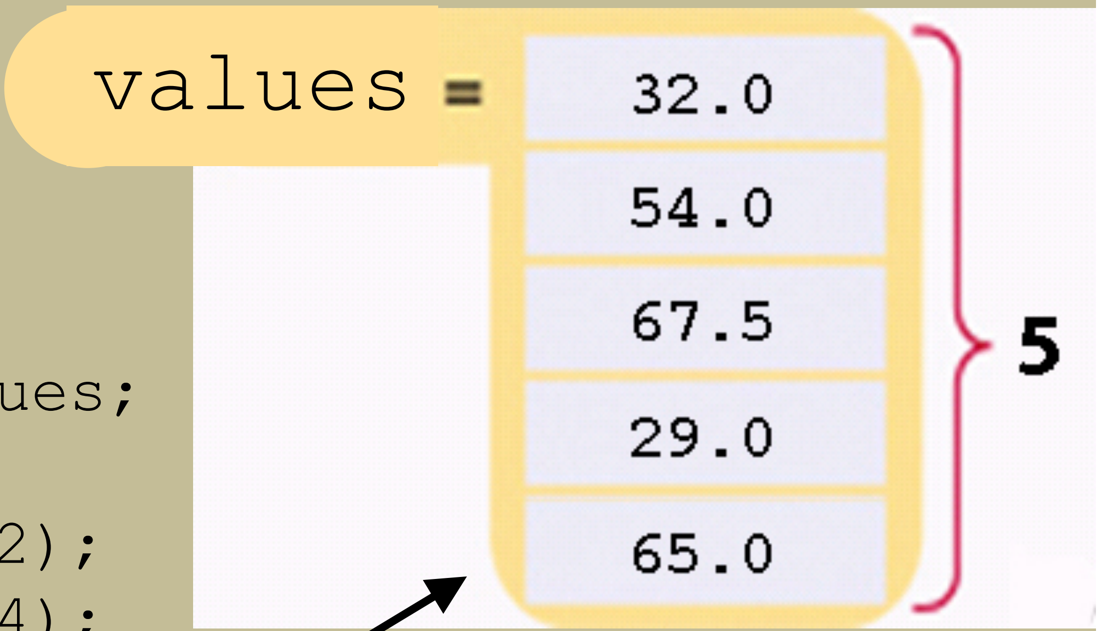


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`29` is placed into the vector. It now contains the elements `32.0`, `54.0`, `67.5` and `29.0`, and its size is `4`.

## `push_back` Adds an Element

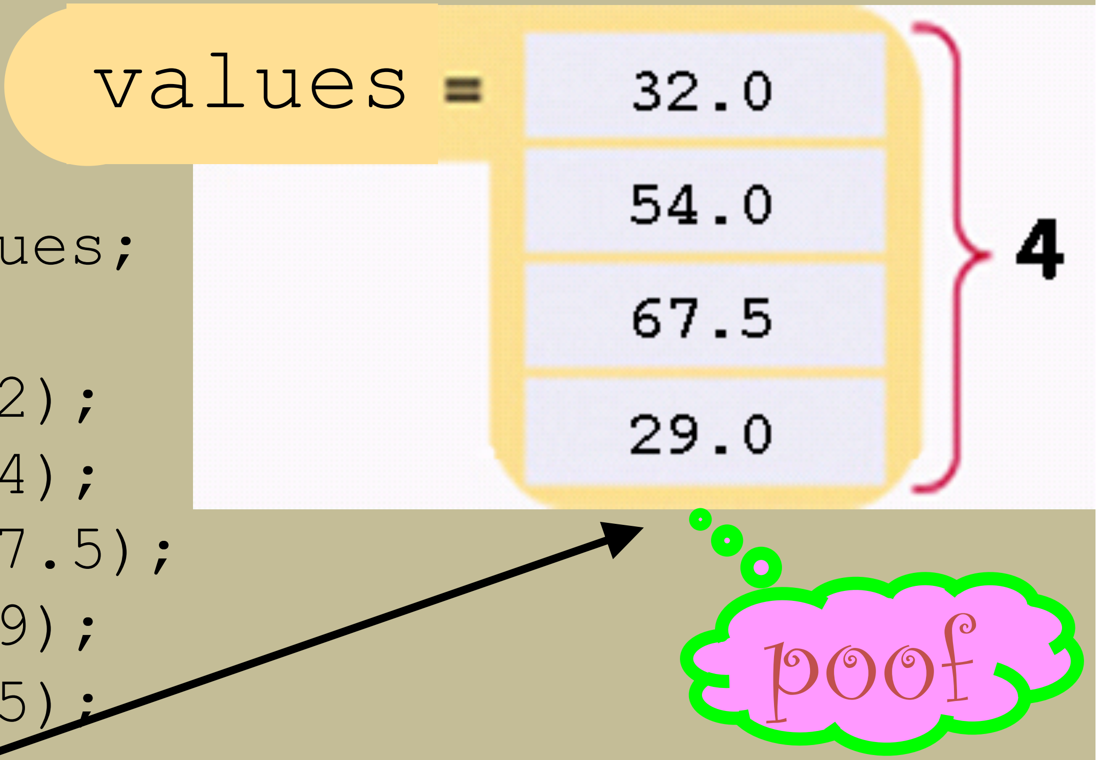


```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`65` is placed into the vector. It now contains the elements `32.0`, `54.0`, `67.5`, `29.0` and `65.0`, and its size is `5`.

# Removing the Last Element with `pop_back`



```
vector<double> values;

values.push_back(32);
values.push_back(54);
values.push_back(67.5);
values.push_back(29);
values.push_back(65);
values.pop_back();
```

`65` is no longer in the vector.
It now contains only the elements `32.0`, `54.0`, `67.5` and `29.0`, and its size is `4`.

## `push_back` and `pop_back`

You can use `push_back` to put user input into a vector:

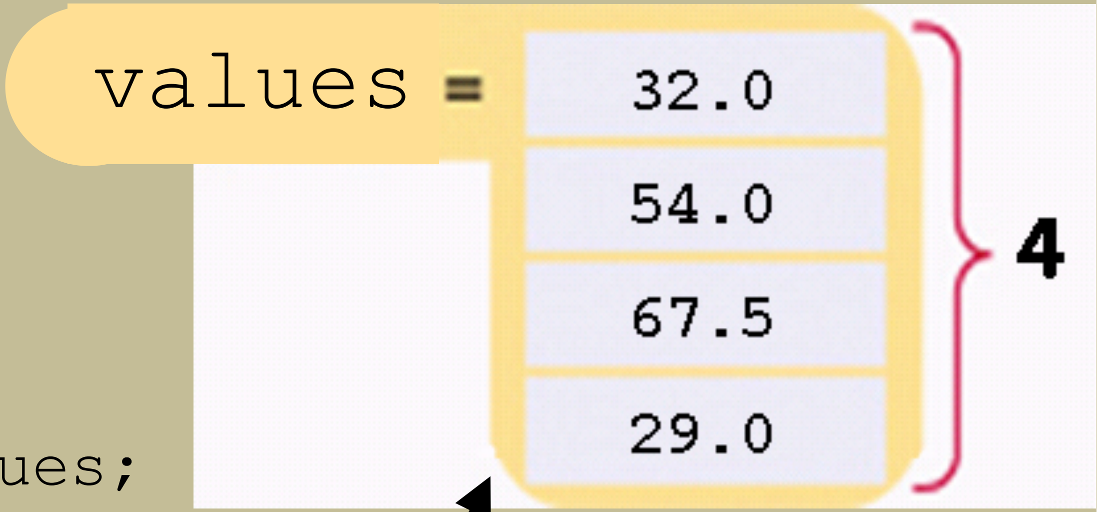
```
double input;
while (cin >> input)
{
   values.push_back(input);
}
```

## `push_back` Adds an Element

```
vector<double> values;

double input;
while (cin >> input)
{
   values.push_back(input);
}
```

# `push_back` Adds an Element



```
vector<double> values;

double input;
while (cin >> input)
{
   values.push_back(input);
}
```

Assuming the user typed `32`, then `54`, then `67.5`, then `29`. The vector size is now 4.

## Using Vectors – `size_of`

How do you visit every element in an vector?

Recall arrays.

## Using Vectors – `size_of`

With arrays, to display every element, it would be:

?

```
for (int i = 0; i < 10; i++)
{
   cout << values[i] << endl;
}
```

But with vectors, we don't know about that 10!

## Using Vectors – `size_of`

Vectors have the `size` member function which returns the current size of a vector.

The vector always knows how many are in it and you can always ask it to give you that quantity by calling the `size` method:

```
for (int i = 0; i < values.size(); i++)
{
   cout << values[i] << endl;
}
```

## Using Vectors – `size_of`

Recall all those array algorithms you learned?

```
for (int i = 0; i < size of array; i++)
   {
      ... // use array[i]
```

To make them work with vectors, you still use a `for` statement, but instead of looping until size of *array*,

you loop until *vector*`.size()`:

```
for (int i = 0; i < vector.size(); i++)
   {
      ... // use vector[i]
```

## Vectors As Parameters In Functions

How can you pass vectors as parameters?

You use vectors as function parameters in exactly the same way as any parameters.

## Vectors Parameters – Without Changing the Values

For example, the following function computes
the sum of a vector of floating-point numbers:

```
double sum(vector<double> values)
{
   double total = 0;
   for (int i = 0; i < values.size(); i++)
   {
      total = total + values[i];
   }
   return total;
}
```

This function *visits* the vector elements,
but it does *not change* them.

## Vectors Parameters – Changing the Values

What about here?

```
void multiply(vector<double> values, double factor)
{
 for (int i = 0; i < values.size(); i++)
 {
    values[i] = values[i] * factor;
 }
}
```

This function *visits* the vector elements,
but it does *not change* them

# Vectors Returned from Functions

Sometimes the function should ***`return`*** a vector.

Vectors are no different from any other values in this regard.

Simply build up the result in the function and return it:

```
vector<int> squares(int n)
{
   vector<int> result;
   for (int i = 0; i < n; i++)
   {
      result.push_back(i * i);
   }
   return result;
}
```

The function returns the squares from $0^2$ up to $(n – 1)^2$ by returning a vector.