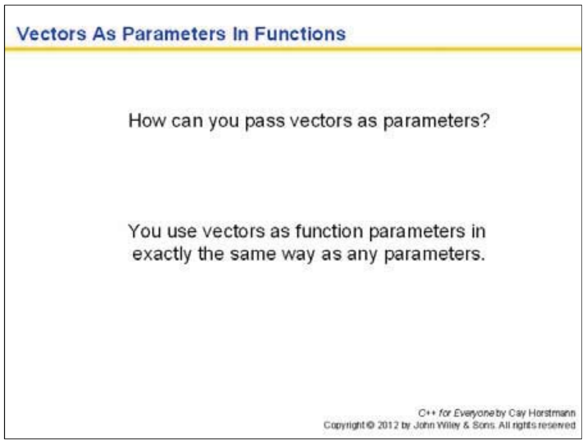
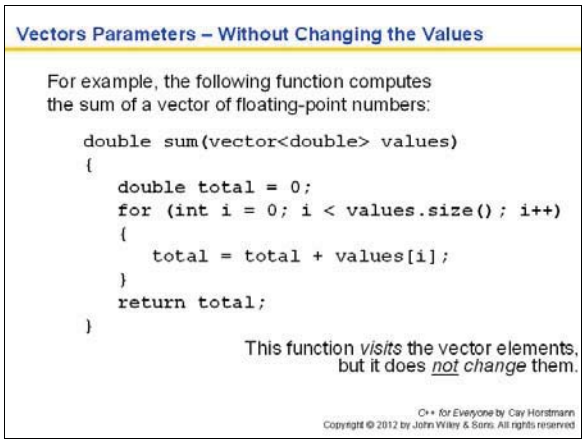


Vectors As Parameters In Functions

How can you pass vectors as parameters?

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

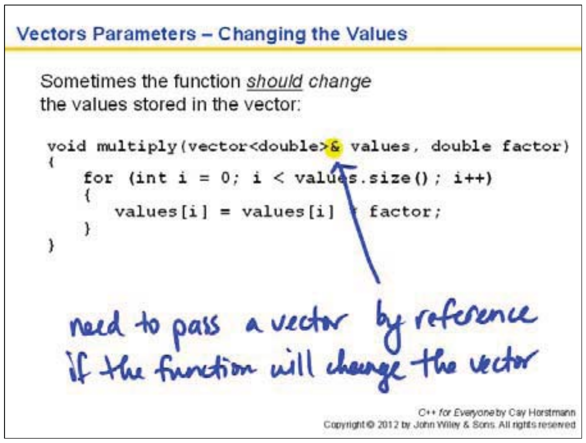


Vectors Parameters – Without Changing the Values

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.

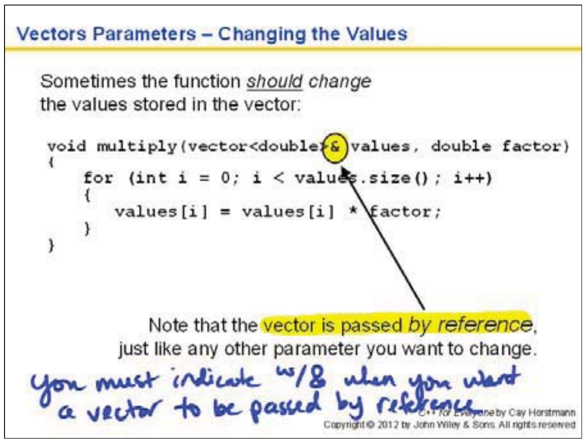


Vectors Parameters – Changing the Values

Sometimes the function *should change* the values stored in the vector:

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

need to pass a vector by reference
if the function will change the vector



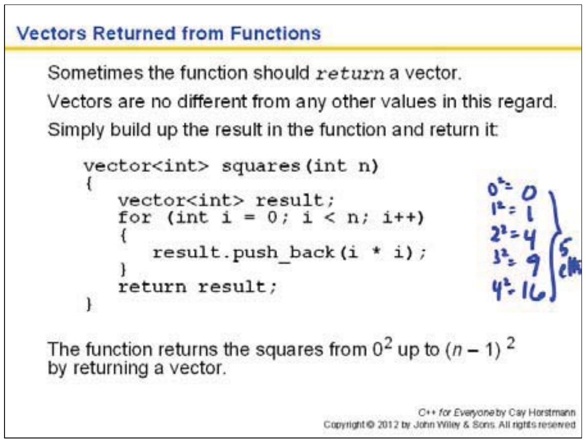
Vectors Parameters – Changing the Values

Sometimes the function *should change* the values stored in the vector:

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

Note that the vector is passed *by reference*, just like any other parameter you want to change.

you must indicate w/& when you want
a vector to be passed by reference



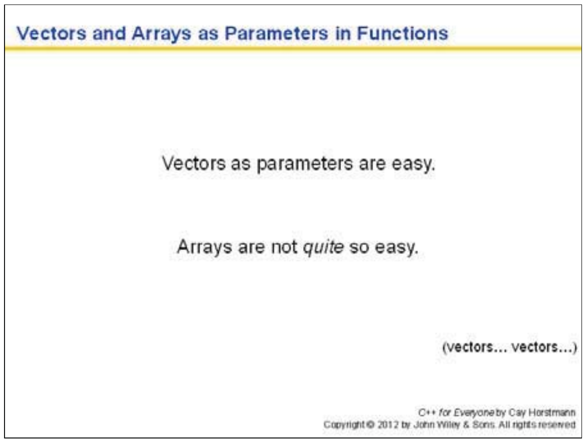
Vectors Returned from Functions

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;
}
```

$0^2 = 0$
$1^2 = 1$
$2^2 = 4$
$3^2 = 9$
$4^2 = 16$
5 ele

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

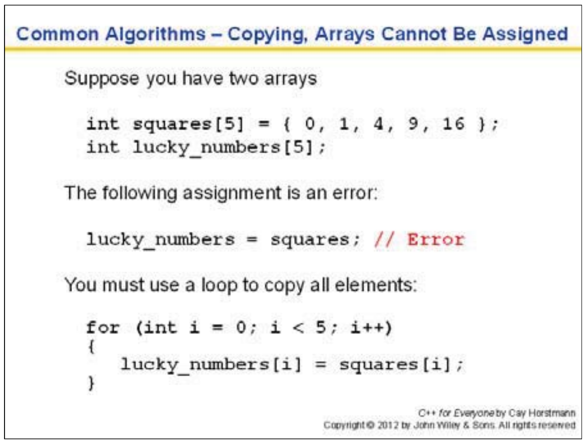


Vectors and Arrays as Parameters in Functions

Vectors as parameters are easy.

Arrays are not *quite* so easy.

(vectors… vectors…)



Common Algorithms – Copying, Arrays Cannot Be Assigned

Suppose you have two arrays

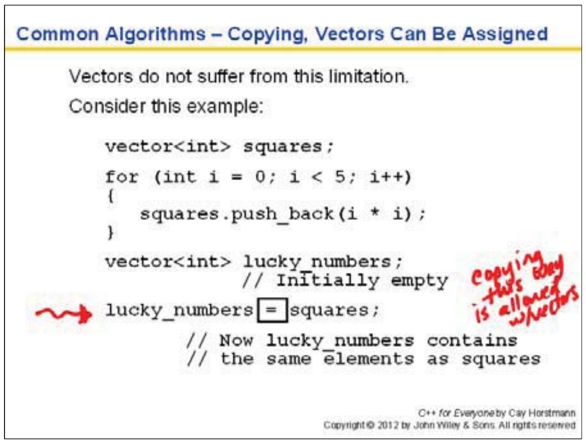
```
int squares[5] = { 0, 1, 4, 9, 16 };
int lucky_numbers[5];
```

The following assignment is an error:

```
lucky_numbers = squares; // Error
```

You must use a loop to copy all elements:

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



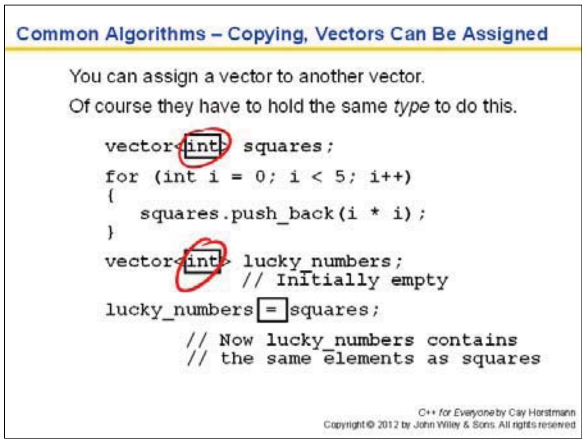
Common Algorithms – Copying, Vectors Can Be Assigned

Vectors do not suffer from this limitation.

Consider this example:

```
vector<int> squares;
for (int i = 0; i < 5; i++)
{
   squares.push_back(i * i);
}
vector<int> lucky_numbers;
            // Initially empty
lucky_numbers = squares;
        // Now lucky_numbers contains
        // the same elements as squares
```

copying this way is allowed w/ vectors

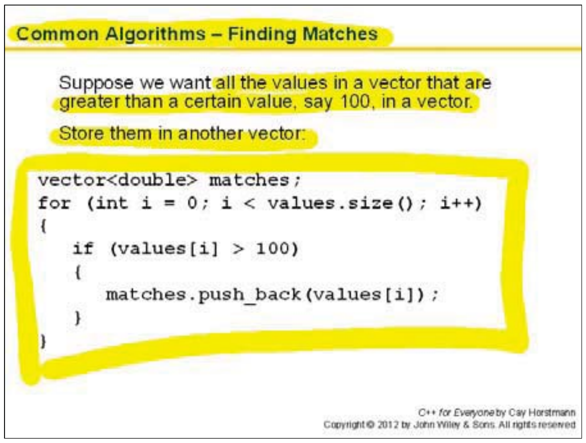


Common Algorithms – Copying, Vectors Can Be Assigned

You can assign a vector to another vector.

Of course they have to hold the same *type* to do this.

```
vector<int> squares;
for (int i = 0; i < 5; i++)
{
   squares.push_back(i * i);
}
vector<int> lucky_numbers;
            // Initially empty
lucky_numbers = squares;
        // Now lucky_numbers contains
        // the same elements as squares
```

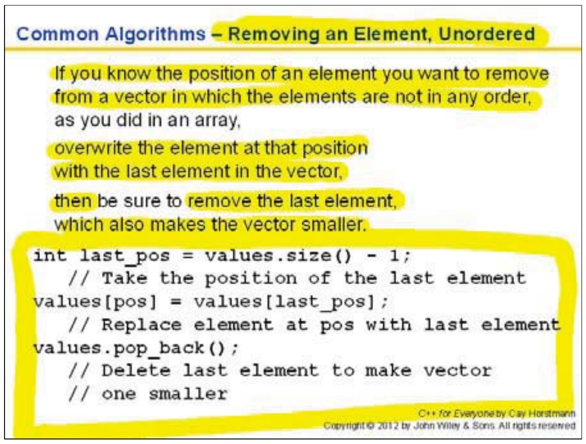


Common Algorithms – Finding Matches

Suppose we want all the values in a vector that are greater than a certain value, say 100, in a vector.

Store them in another vector:

```
vector<double> matches;
for (int i = 0; i < values.size(); i++)
{
   if (values[i] > 100)
   {
      matches.push_back(values[i]);
   }
}
```



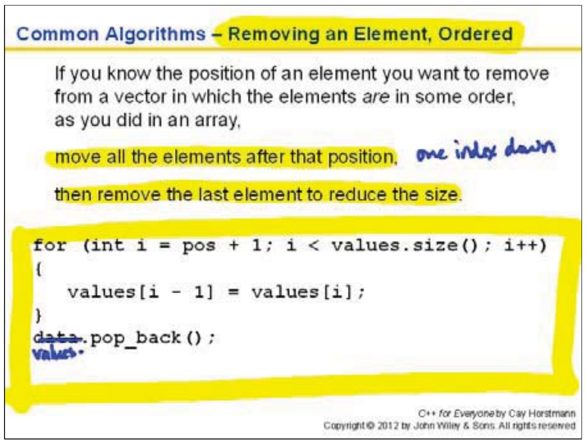
Common Algorithms – Removing an Element, Unordered

If you know the position of an element you want to remove from a vector in which the elements are not in any order, as you did in an array,

overwrite the element at that position with the last element in the vector,

then be sure to remove the last element, which also makes the vector smaller.

```
int last_pos = values.size() - 1;
   // Take the position of the last element
values[pos] = values[last_pos];
   // Replace element at pos with last element
values.pop_back();
   // Delete last element to make vector
   // one smaller
```



Common Algorithms – Removing an Element, Ordered

If you know the position of an element you want to remove from a vector in which the elements *are* in some order, as you did in an array,

move all the elements after that position, one index down

then remove the last element to reduce the size.

```
for (int i = pos + 1; i < values.size(); i++)
{
   values[i - 1] = values[i];
}
~~data~~ values.pop_back();
```

Common Algorithms – Inserting an Element, Unordered

When you need to insert an element into a vector whose elements are not in any order…

…oh, this is going to be so easy:

```
values.push_back(new_element);
```

one simple step

Common Algorithms – Inserting an Element, Ordered

However when the elements in a vector are in some order, it's a bit more complicated, just like it was in the array version.

Of course you must know the position, say `pos`, where you will insert the new element.

As in the array version, you need to move all the elements "up". (index)

```
for (int i = last_pos; i > pos; i--)
{
   values[i] = values[i - 1];
}
```

WAIT!!!

Common Algorithms – Inserting an Element, Ordered

You can't do that!

In a vector you cannot assign to the position after the last one!

You cannot assign to any position bigger than `values() - 1.`

OH DEAR!!!

Common Algorithms – Inserting an Element, Ordered

Somehow you need to make the vector one bigger

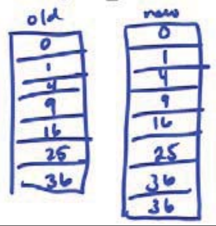
before you do the moving.

Common Algorithms – Inserting an Element, Ordered

Be clever.

If you `push_back` the last element:

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
```



…but, but…

Common Algorithms – Inserting an Element, Ordered

Yes, it will be in the vector twice,

but why care?

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
```

You will overwrite it by doing the moving.

Common Algorithms – Inserting an Element, Ordered

And, more importantly,
the vector is now one larger after the `push_back`.
Congratulations, it's to safe go ahead and start moving.

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
for (int i = last_pos; i > pos; i--)
{
   values[i] = values[i - 1];
}
values[pos] = new_element;
```

And don't forget to insert the new element.
That's what you've been trying to do all along!

Common Algorithms – Inserting an Element, Ordered

Ah.

```
int last_pos = values.size() - 1;
values.push_back(values[last_pos]);
for (int i = last_pos; i > pos; i--)
{
   values[i] = values[i - 1];
}
values[pos] = new_element;
```

Common Algorithms – Inserting an Element, Ordered

But don't be too clever,
if the position to insert the new element
is after the last element...

...oh, this is going to be so easy,
don't do any moving, just put it there:

```
values.push_back(new_element);
```

if need to insert
at the end
for some reason

Inserting into an ordered vector means
inserting into the *middle* of the vector!

Sorting with the C++ Library

Recall that you call the `sort` function
to do your sorting for you.
This can be used on vectors also.

The syntax for vectors is even more unusual than arrays:

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

name of vector goes here

Go ahead and use it as you like.
But don't forget to `#include <algorithm>`

sort function located here

Arrays or Vectors? That Is the Question

Should you use arrays or vectors?

(you know you want to say vectors...)

Arrays or Vectors? That Is the Question

For most programming tasks,
vectors are easier to use than arrays.

(say vectors, say vectors...)

Arrays or Vectors? That Is the Question

Vectors can grow and shrink.

(grow, shrink - *think:* vectors…)

Arrays or Vectors? That Is the Question

Even if a vector always stays the same size,
it is convenient that a vector remembers its size.

No chance of missing auxiliaries.

Vectors are smarter then arrays!

(size matters and vectors know their own - vectors…)

Arrays or Vectors? That Is the Question

For a beginner, the sole advantage of
an array is the initialization syntax.

(syntax, shmyntax – it's easy too with vectors…)

Arrays or Vectors? That Is the Question

Advanced programmers sometimes prefer arrays
because they are a bit more efficient.

Moreover, you need to know how to use
arrays if you work with older programs

(only a bit? and *older*? why not be current by using vectors…)

Prefer Vectors over Arrays

So:

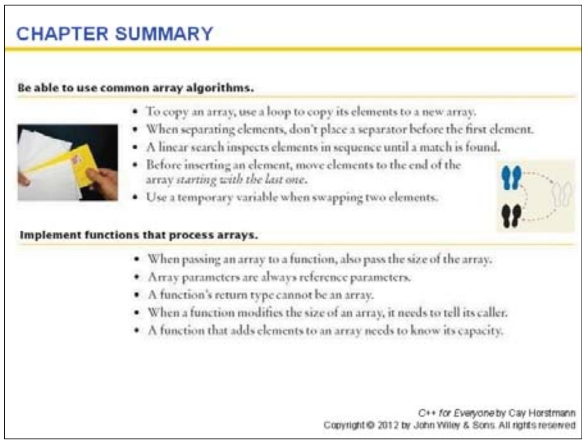
Prefer Vectors over Arrays

(it's so nice when the moral of the story is: vectors!!!)

CHAPTER SUMMARY

Use arrays for collecting values.

- Use an array to collect a sequence of values of the same type.
- Individual elements in an array *values* are accessed by an integer index i, using the notation *values*[i].
- An array element can be used like any variable.
- An array index must be at least zero and less than the size of the array.
- A bounds error, which occurs if you supply an invalid array index, can corrupt data or cause your program to terminate.
- With a partially filled array, keep a companion variable for the current size.



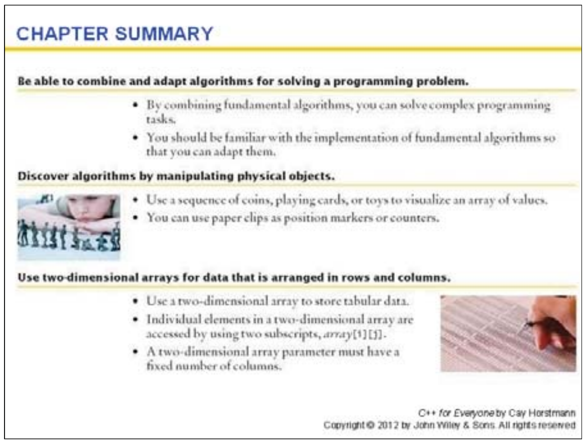
CHAPTER SUMMARY

Be able to use common array algorithms.

- To copy an array, use a loop to copy its elements to a new array.
- When separating elements, don't place a separator before the first element.
- A linear search inspects elements in sequence until a match is found.
- Before inserting an element, move elements to the end of the array *starting with the last one*.
- Use a temporary variable when swapping two elements.

Implement functions that process arrays.

- When passing an array to a function, also pass the size of the array.
- Array parameters are always reference parameters.
- A function's return type cannot be an array.
- When a function modifies the size of an array, it needs to tell its caller.
- A function that adds elements to an array needs to know its capacity.



CHAPTER SUMMARY

Be able to combine and adapt algorithms for solving a programming problem.

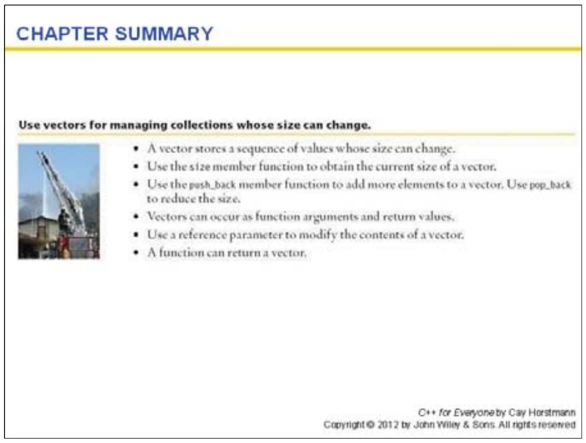
- By combining fundamental algorithms, you can solve complex programming tasks.
- You should be familiar with the implementation of fundamental algorithms so that you can adapt them.

Discover algorithms by manipulating physical objects.

- Use a sequence of coins, playing cards, or toys to visualize an array of values.
- You can use paper clips as position markers or counters.

Use two-dimensional arrays for data that is arranged in rows and columns.

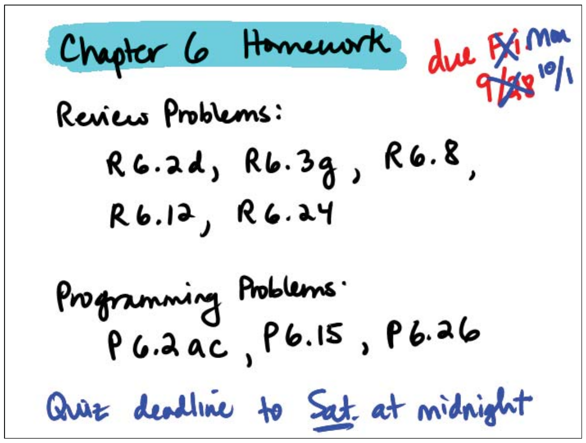
- Use a two-dimensional array to store tabular data.
- Individual elements in a two-dimensional array are accessed by using two subscripts, *array*[i][j].
- A two-dimensional array parameter must have a fixed number of columns.



CHAPTER SUMMARY

Use vectors for managing collections whose size can change.

- A vector stores a sequence of values whose size can change.
- Use the size member function to obtain the current size of a vector.
- Use the push_back member function to add more elements to a vector. Use pop_back to reduce the size.
- Vectors can occur as function arguments and return values.
- Use a reference parameter to modify the contents of a vector.
- A function can return a vector.



Chapter 6 Homework

due ~~Fri~~ Mon ~~9/28~~ 10/1

Review Problems:

R6.2d, R6.3g, R6.8, R6.12, R6.24

Programming Problems:

P6.2ac, P6.15, P6.26

Quiz deadline to Sat. at midnight