



## Chapter Seven: Pointers, Part I

Slides by Evan Gallagher

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers



No, that one – the one I'm **pointing** at!

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Chapter Goals

- To be able to declare, initialize, and use pointers
- To understand the relationship between arrays and pointers
- To be able to convert between string objects and character pointers
- To become familiar with dynamic memory allocation and deallocation

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

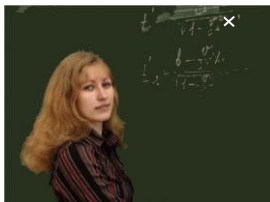
## Pointers

A variable **contains** a value,  
but a **pointer** specifies **where** a value is located.

A pointer denotes the  
*memory location* of a variable

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

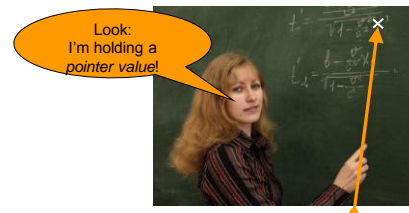
## Pointers



What's stored in that variable?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers



Yes, I mean x

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers

- In C++, pointers are important for several reasons.
  - Pointers allow sharing of values stored in variables in a uniform way
  - Pointers can refer to values that are allocated on demand (*dynamic memory allocation*)
  - Pointers are necessary for implementing *polymorphism*, an important concept in object-oriented programming (later)

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry Needs a Banking Program

Harry wants a program for making bank deposits and withdrawals.

(You can write that code by now!)

```
... balance += depositAmount ...
... balance -= withdrawalAmount ...
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## A Banking Problem

Consider a person.  
A chef.



(Harry)

Hi. Nice to see you again.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry Needs a Multi-Bank Banking Program

But not all deposits and withdrawals should be from the same bank.

```
... balance += depositAmount ...
... balance -= withdrawalAmount ...
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry Needs a Banking Program

Harry has more than one bank account.



Business is GREAT with those algorithms!

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Good Design

But withdrawing is withdrawing  
– no matter which bank it is.

Same with depositing.

Same problem – same code, right?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

By using a *pointer*,  
it is possible to *switch* to a different account  
*without* modifying the code for  
deposits and withdrawals.

(Ah, code reuse!)

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

A *pointer to double* type can hold the address of a `double`.

So what's an address?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

Harry starts with a variable for storing an account balance.  
It should be initialized to 0 since there is no money yet.

```
double harrys_account = 0;
```

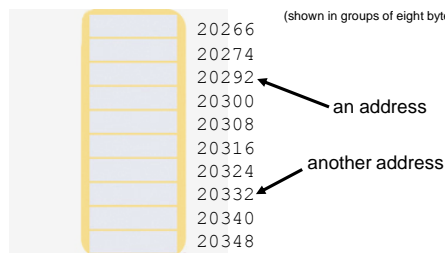


C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

Here's a picture of RAM.

Every byte in RAM  
has an *address*.  
(shown in groups of eight bytes)



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

If Harry anticipates that he may someday use other  
accounts, he can use a pointer to access any accounts.

So Harry also declares a pointer variable  
named `account_pointer` :

```
double* account_pointer
```

The type of this variable is "*pointer to double*".

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

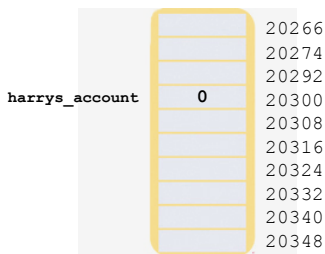
Here's how we have pictured a variable in the past:

harrys\_account    0

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

But really it's been like this all along:



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

So when Harry declares a pointer variable, he also initializes it to point to `harrys_account`:

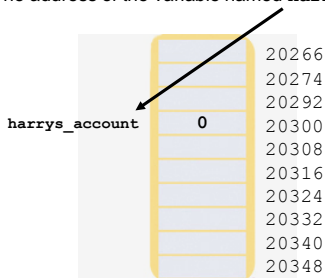
```
double harrys_account = 0;
double* account_pointer = &harrys_account;
```

The `&` operator yields the location (or address) of a variable. Taking the address of a `double` variable yields a value of type `double*` so everything fits together nicely.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

The address of the variable named `harrys_account`

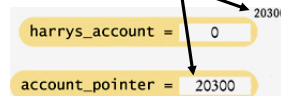


C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

`account_pointer` now contains the address of `harrys_account`

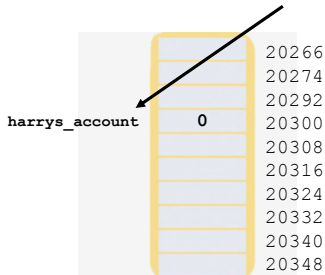
```
double harrys_account = 0;
double* account_pointer = &harrys_account;
```



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

The address of the variable named `harrys_account` is 20300

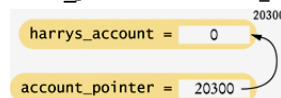


C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers to the Rescue

`account_pointer` now "points to" `harrys_account`

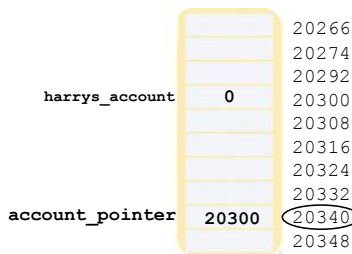
```
double harrys_account = 0;
double* account_pointer = &harrys_account;
```



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

And, of course, `account_pointer` is *somewhere* in RAM:

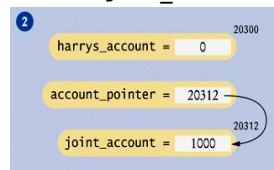


C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

To access a different account, like `joint_account`, Harry (and you) would change the pointer value stored in `account_pointer` and similarly use `account_pointer`.

```
double harrys_account = 0;
account_pointer = &harrys_account;
double joint_account = 1000;
account_pointer = &joint_account;
```



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers – and ARROWS

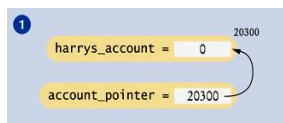
Do note that the computer stores numbers,  
not arrows.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Addresses and Pointers

To access a different account, Harry (and you) would change the pointer value stored in `account_pointer`:

```
double harrys_account = 0;
account_pointer = &harrys_account;
```



Harry (and you) would use `account_pointer` to access `harrys_account`.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry Sells An ALGORITHMMMMMCAKE

Harry makes his first ALGORITHMMMMMCAKE sale.



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

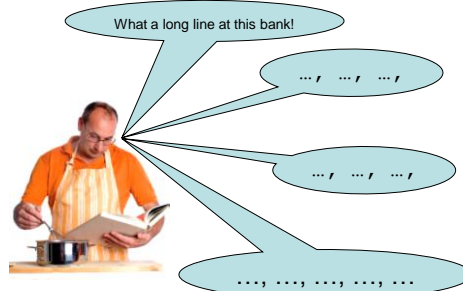
## – And Deposits the Money

Harry needs to deposit this cash into his account  
– into the `harrys_account` variable



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry at the Bank ...



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Accessing the Memory Pointed to by A Pointer Variable

When you have a pointer to a variable,  
you will want to access the value to which it points.

... `*account_pointer` ...

In C++ the `*` operator is used to indicate  
the memory location associated with a pointer.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry at the Bank ...



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Accessing the Memory Pointed to by A Pointer Variable

An expression such as `*account_pointer` can be used  
wherever a variable name of the same type can be used:

```
// display the current balance
cout << *account_pointer << endl;
```

It can be used on the left or the right of an assignment:

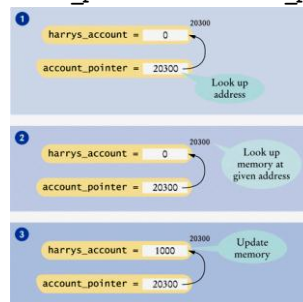
```
// withdraw $100
*account_pointer = *account_pointer - 100;
```

(or both)

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Harry Makes the Deposit

```
// deposit $1000
*account_pointer = *account_pointer + 1000;
```



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Accessing the Memory Pointed to by A Pointer Variable

Of course, this only works  
if `account_pointer` is pointing  
to `harrys_account`!

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## NULL

There is a special value  
that you can use  
to indicate a pointer  
that doesn't point anywhere:

**NULL**

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Errors Using Pointers – Uninitialized Pointer Variables

When a pointer variable is first defined,  
it contains a random address.

Using that random address is an **error**.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## NULL

If you define a pointer variable  
and are not ready to initialize it quite yet,  
it is a good idea to set it to **NULL**.

You can later test whether the pointer is **NULL**.  
If it is, don't use it.

```
double* account_pointer = NULL; // Will set later
if (account_pointer != NULL)    // OK to use
{
    cout << *account_pointer;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Errors Using Pointers – Uninitialized Pointer Variables

In practice, your program will likely crash or mysteriously  
misbehave if you use an uninitialized pointer:

```
double* account_pointer; // No initialization
```

```
*account_pointer = 1000;
```

NO!  
`account_pointer` contains an **unpredictable** value!  
Where is the 1000 going?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## NULL

Trying to access data through a NULL pointer is still illegal,  
and  
it will cause your program to crash.

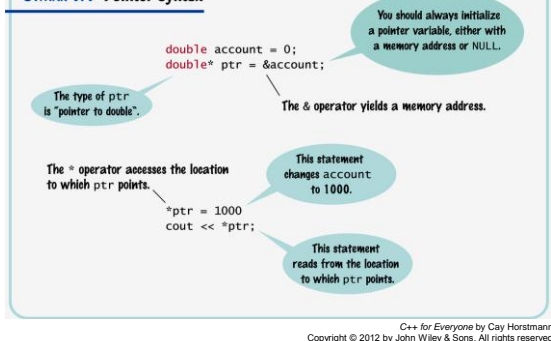
```
double* account_pointer = NULL;
cout << *account_pointer;
```

**CRASH!!!**

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntax of Pointers

## SYNTAX 7.1 Pointer Syntax



## Harry's Banking Program

ch07/accounts.cpp

```
// Withdraw $100
*account_pointer = *account_pointer - 100;

// Print balance
cout << "Balance: " << *account_pointer
    << endl;

// Change the pointer value so that the same
// statements now affect a different account
account_pointer = &joint_account;

// Withdraw $100
*account_pointer = *account_pointer - 100;

// Print balance
cout << "Balance: " << *account_pointer
    << endl;

return 0;
```

Copyright © 2012 by John Wiley & Sons. All rights reserved.

## Pointer Syntax Examples

Assume the following declarations: <code>int n = 10; // Assumed to be at address 20300</code> <code>int* p = 20; // Assumed to be at address 20304</code> <code>int* q = &amp;n;</code>		
Expression	Value	Comment
<code>p</code>	20300	The address of <code>n</code> .
<code>*p</code>	10	The value stored at that address.
<code>&amp;n</code>	20304	The address of <code>n</code> .
<code>p = &amp;n;</code>		Set <code>p</code> to the address of <code>n</code> .
<code>*p</code>	20	The value stored at the changed address.
<code>n = *p;</code>		Stores 20 into <code>n</code> .
<code>n = p;</code>	Error	<code>n</code> is an <code>int</code> value; <code>p</code> is an <code>int*</code> pointer. The types are not compatible.
<code>&amp;10</code>	Error	You can only take the address of a variable.
<code>&amp;p</code>		This is the location of a pointer variable, not the location of an integer.
<code>double x = 0;</code> <code>p = &amp;x;</code>	Error	<code>p</code> has type <code>int*</code> , & <code>x</code> has type <code>double*</code> . These types are incompatible.

Copyright © 2012 by John Wiley & Sons. All rights reserved.

## Common Error: Confusing Data And Pointers

A pointer is a memory address

– a number that tells where a value is located in memory.

It is a common error to confuse the pointer with the variable to which it points.

Copyright © 2012 by John Wiley & Sons. All rights reserved.

## Harry's Banking Program

Here is the complete banking program that Harry wrote. It demonstrates the use of a pointer variable to allow *uniform access* to variables.

```
#include <iostream>
using namespace std;

int main()
{
    double harrys_account = 0;
    double joint_account = 2000;
    double* account_pointer = &harrys_account;
    *account_pointer = 1000; // Initial deposit
```

ch07/accounts.cpp

Copyright © 2012 by John Wiley & Sons. All rights reserved.

## Common Error: Where's the \*?

```
double* account_pointer = &joint_account;
account_pointer = 1000;
```

The assignment statement does *not* set the joint account balance to 1000.

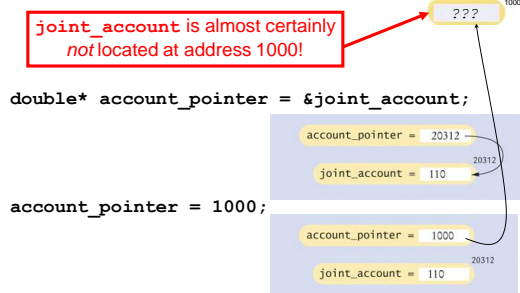
It sets the pointer variable, `account_pointer`, to point to memory address 1000.

ERROR

Copyright © 2012 by John Wiley & Sons. All rights reserved.



## Common Error: Where's the \*?



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Pointers and References

& == \*

?

What are you asking?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Common Error: Where's the \*?

Most compilers will report an error for this kind of error.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers and References

Recall that the & symbol is used for reference parameters:

```
void withdraw(double& balance, double amount)
{
    if (balance >= amount)
    {
        balance = balance - amount;
    }
}
```

a call would be:

```
withdraw(harrys_checking, 1000);
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Confusing Definitions

It is legal in C++ to define multiple variables together, like this:

```
int i = 0, j = 1;
```

This style is confusing when used with pointers:

```
double* p, q;
```

The \* associates only with the first variable.

That is, *p* is a *double\** pointer, and *q* is a *double* value.

To avoid any confusion, it is best to define each pointer variable separately:

```
double* p;
double* q;
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointers and References

We can accomplish the same thing using pointers:

```
void withdraw(double* balance, double amount)
{
    if (*balance >= amount)
    {
        *balance = *balance - amount;
    }
}
```

but the call will have to be:

```
withdraw(&harrys_checking, 1000);
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers

In C++, there is a deep relationship between pointers and arrays.

This relationship explains a number of special properties and limitations of arrays.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers

Consider this declaration:

```
int a[10];
```

(Assume we have filled it as shown.)

You can capture the pointer to the first element in the array in a variable:

```
int* p = a; // Now p points to a[0]
```

Index	Value	Address
0	20300	20300
1		20308
4		20316
9		20324
16		20332
25		20340
36		20348
49		20356
64		20364
81		20372

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers

Pointers are particularly useful for understanding the peculiarities of arrays.

The *name* of the array denotes a pointer to the starting element.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers – Same Use

You can use the array name `a` as you would a pointer:

These output statements are equivalent:

```
cout << *a;  
cout << a[0];
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers

Consider this declaration:

```
int a[10];
```

(Assume we have filled it as shown.)

You can capture the pointer to the first element in the array in a variable:

Index	Value	Address
0	20300	20300
1		20308
4		20316
9		20324
16		20332
25		20340
36		20348
49		20356
64		20364
81		20372

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Pointer Arithmetic

*Pointer arithmetic* allows you to add an integer to an array name.

```
int* p = a;
```

`p + 3` is a pointer to the array element with index 3

The expression: `*(p + 3)`

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

The *array/pointer duality law* states:

$a[n]$  is identical to  $*(a + n)$ ,

where  $a$  is a pointer into an array  
and  $n$  is an integer offset.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

Consider this function that computes the sum of all values in an array: *Look at this*

```
double sum(double a[], int size)
{
    double total = 0;
    for (int i = 0; i < size; i++)
    {
        total = total + a[i];
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

This law explains why all C++ arrays start with an index of zero.

The pointer  $a$  (or  $a + 0$ ) points to the starting element of the array.

That element must therefore be  $a[0]$ .

You are adding 0 to the start of the array, thus *correctly going nowhere!*

a	0	20300
	1	20308
	4	20316
	9	20324
	16	20332
	25	20340
	36	20348
	49	20356
	64	20364
	81	20372

p = 20300

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

Here is a call to the function.

```
double data[10];
... // Initialize data

double s = sum(data, 10);
```

data	0	20300
	1	
	4	
	9	20324
	16	
	25	
	36	
	49	
	64	
	81	

s =

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

Now it should be clear why array parameters are different from other parameter types.

(if not, we'll show you)

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

After the loop has run to the point when  $i$  is 3:

```
double sum(double a[], int size)
{
    double total = 0;
    for (int i = 0; i < size; i++)
    {
        total = total + a[i];
    }
    return total;
}
```

data	0	20300
	1	
	4	
	9	20324
	16	
	25	
	36	
	49	
	64	
	81	

s =

a = 20300

size = 10

total = 5

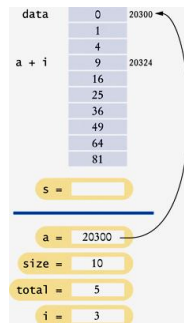
i = 3

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## The Array/Pointer Duality Law

The C++ compiler considers **a** to be a pointer, not an array.

The expression **a[i]** is *syntactic sugar* for **\*(a + i)**.



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntactic Sugar



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntactic Sugar



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntactic Sugar

That masked complex implementation detail:

**double sum(double\* a, int size)**  
is how we *should* define the first parameter

but

**double sum(double a[], int size)**  
looks a lot more like we are passing an array.

(yummy!)

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntactic Sugar

Computer scientists use the term

*"syntactic sugar"*

to describe a notation that is easy to read for humans  
and that masks a complex implementation detail.

*Yum!*

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Syntactic Sugar



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Arrays and Pointers

Expression	Value	Comment
a	20300	The starting address of the array, here assumed to be 20300.
*a	0	The value stored at that address. (The array contains values 0, 1, 4, 9, ....)
a + 1	20308	The address of the next double value in the array. A double occupies 8 bytes.
a + 3	20324	The address of the element with index 3, obtained by skipping past 3 × 8 bytes.
*(a + 3)	9	The value stored at address 20324.
a[3]	9	The same as *(a + 3) by array/pointer duality.
*a + 3	3	The sum of *a and 3. Since there are no parentheses, the * refers only to a.
&a[3]	20324	The address of the element with index 3, the same as a + 3.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array

Watch variable p as this code is executed.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

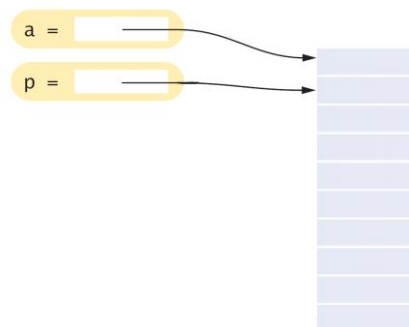
## Using a Pointer to Step Through an Array

Watch variable p as this code is executed.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

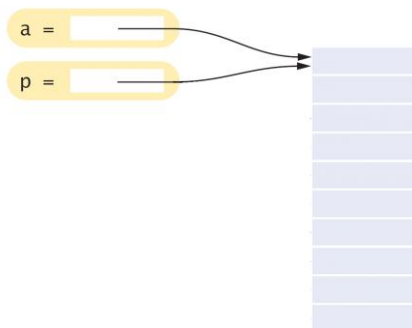
C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

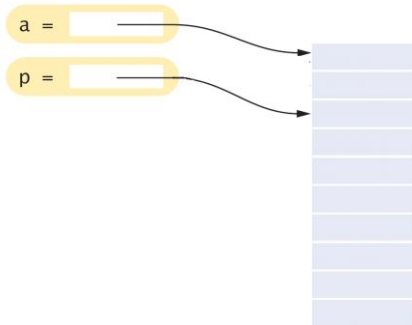
## Using a Pointer to Step Through an Array

Watch variable p as this code is executed.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

### Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

### Using a Pointer to Step Through an Array

Add, then again move `p` to the next position by incrementing.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

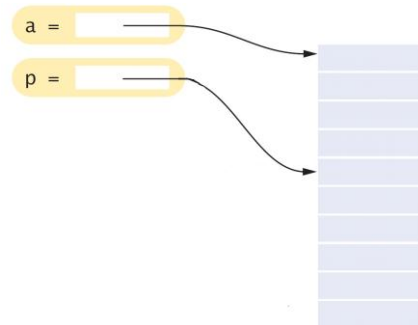
### Using a Pointer to Step Through an Array

Add, then move `p` to the next position by incrementing.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

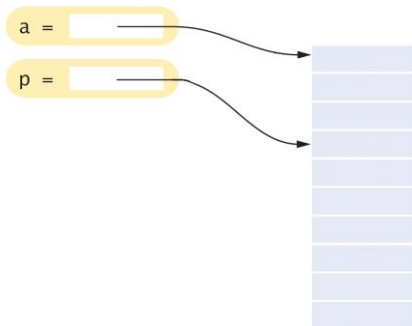
C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

### Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

### Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

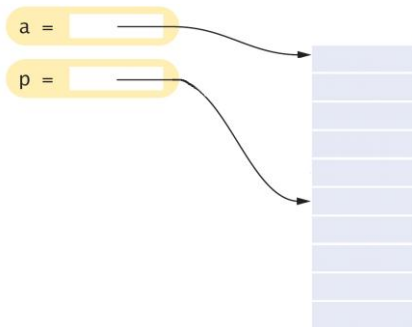
### Using a Pointer to Step Through an Array

Add, then move `p`.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array

And so on until every single position in the array has been added.

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

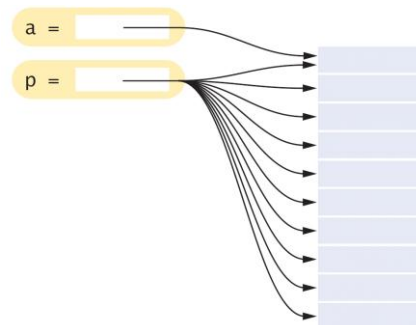
## Using a Pointer to Step Through an Array

Again...

```
double sum(double* a, int size)
{
    double total = 0;
    double* p = a;
    // p starts at the beginning of the array
    for (int i = 0; i < size; i++)
    {
        total = total + *p;
        // Add the value to which p points
        p++;
        // Advance p to the next array element
    }
    return total;
}
```

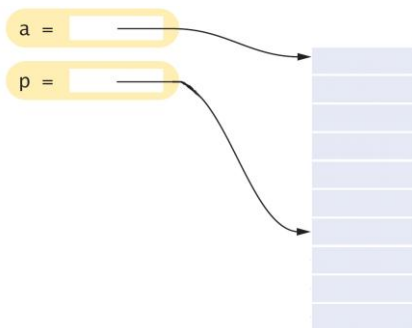
C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array



C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Using a Pointer to Step Through an Array

It is a tiny bit more efficient to use and increment a pointer than to access an array element.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Program Clearly, Not Cleverly

Some programmers take great pride in minimizing the number of instructions, even if the resulting code is hard to understand.

```
while (size-- > 0) // Loop size times
{
    total = total + *p;
    p++;
}
```

could be written as:

```
total = total + *p++;
```

Ah, so much better?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Common Error: Returning a Pointer to a Local Variable

What would it mean to  
"return an array"  
?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Program Clearly, Not Cleverly

```
while (size > 0)
{
    total = total + *p;
    p++;
    size--;
}
```

could be written as:

```
while (size-- > 0)
    total = total + *p++;
```

Ah, so much better?

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Common Error: Returning a Pointer to a Local Variable

Consider this function that tries to return a pointer to an array containing two elements, the first and last values of an array:

```
double* firstlast(double a[], int size)
{
    double result[2];
    result[0] = a[0];
    result[1] = a[size - 1];
    return result;
}
```

*Local memory is invalid after the function call has ended!*

*What would the value the caller gets be pointing to?*

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Program Clearly, Not Cleverly

Please do not use this programming style.

Your job as a programmer is not to dazzle other programmers with your cleverness, but to write code that is easy to understand and maintain.

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved

## Common Error: Returning a Pointer to a Local Variable

A solution would be to pass in an array to hold the answer:

```
double* firstlast(double a[], int size,
                  double result)
{
    result[0] = a[0];
    result[1] = a[size - 1];
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2012 by John Wiley & Sons. All rights reserved



## C and C++ Strings, POP QUIZ

"Q: What?"

Really we mean:

"Q: What is this?"

A *C string*, of course!  
(notice the double quotes: "Like this")

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## char Type and Some Famous Characters

Some of these characters are plain old letters and such:

```
char yes = 'y';
char no = 'n';
char maybe = '?';
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## C and C++ Strings

C++ has two mechanisms for manipulating strings.

The **string** class

- Supports character sequences of arbitrary length.
- Provides convenient operations such as concatenation and string comparison.

C strings

- Provide a more primitive level of string handling.
- Are from the C language (C++ was built from C).
- Are represented as arrays of **char** values.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## char Type and Some Famous Characters

Some are numbers masquerading as digits:

```
char theThreeChar = '3';
```

That is not the number three – it's the *character* 3.

'3' is what is actually stored in a disk file  
when you write the **int** 3.

Writing the variable **theThreeChar** to a file  
would put the same '3' in a file.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## char Type and Some Famous Characters

The type **char** is used to store an individual character.

## char Type and Some Famous Characters

Recall that a stream is a  
sequence of characters – **chars**.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## char Type and Some Famous Characters

So some characters are literally what they are:

'A'

Some represent digits:

'3'

Some are other things that can be typed:

'C'

'+'

'+'

but...

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Some Famous Characters

And there is one special character that is especially special to C strings:

The null terminator character:

'\0'

That is an escaped zero.

It's in ASCII position zero.

It is the value 0 (not the character zero, '0')

If you output it to screen nothing will appear.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Some Famous Characters

Some of these characters are true individuals.  
"Characters" you might say (if they were human).

They are quite "special":

'\n'


'\t'

These are still single (individual) characters:  
the **escape sequence** characters.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Some Famous Characters

**Table 3 Character Literals**

'y'	The character y
'0'	The character for the digit 0. In the ASCII code, '0' has the value 48.
' '	The space character
'\n'	The newline character
'\t'	The tab character
'\0'	The null terminator of a string
 "y"	<b>Error:</b> Not a char value

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Some Famous Characters

And one you can output to the screen  
in order to annoy those around you  
(if you were naughty and didn't mute your  
computer when you entered the classroom)

'\a'

– the *alert* character.

Don't try this at home

– no we mean

**ONLY** try this at home!!!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## The Null Terminator Character and C Strings

The null character is special to C strings because  
it is always the last character in them:

"CAT" is really this sequence of characters:

'C' 'A' 'T' '\0'

The null terminator character  
indicates the end of the C string

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## The Null Terminator Character and C Strings

The literal C string "CAT" is actually an array of **four** chars stored somewhere in the computer.

In the C programming language, literal strings are always stored as character arrays.

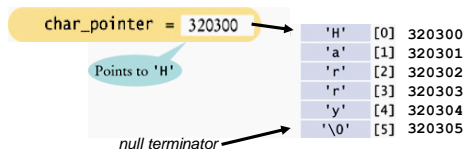
Now you know why C++ programmers often refer to arrays of char values as "C strings".

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays as Storage for C Strings

As with all arrays, a string literal can be assigned to a pointer variable that points to the initial character in the array:

```
char* char_pointer = "Harry";  
// Points to the 'H'
```



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Pop Quiz #2.

Q:

Is "C strings" a string?

Yes

...wait...

No

...wait...

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Using the Null Terminator Character

Functions that operate on C strings rely on this terminator. The **strlen** function returns the length of a C string.

```
#include <cstring>  
int strlen(const char s[])  
{  
    int i = 0;  
    // Count characters before  
    // the null terminator  
    while (s[i] != '\0') { i++; }  
    return i;  
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Pop Quiz #2

Answer:

"C strings" is NOT an object of **string** type.

"C strings" IS an array of **chars** with a null terminator character at the end.

(and that English was correct!)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Using the Null Terminator Character

The call **strlen("Harry")** returns 5.

The null terminator character is not counted as part of the "length" of the C string – but it's there.

Really, it is.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays

Literal C strings are considered constant.

You are not allowed to modify its characters.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays

The compiler counts the characters in the string that is used for initializing the array, including the null terminator.

```
char char_array[] = "Harry";
```

↑  
(6)

*I'm the compiler && I put that 6 there*

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays

If you want to modify the characters in a C string, define a character array to hold the characters instead.

For example:

```
// An array of 6 characters  
char char_array[] = "Harry";
```

↑  
Isn't something missing?

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays

You can modify the characters in the array:

```
char char_array[] = "Harry";  
char_array[0] = 'L';
```

*I'm the programmer && I changed Harry into Larry!*

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Character Arrays

The compiler counts the characters in the string that is used for initializing the array, including the null terminator.

```
char char_array[] = "Harry";
```

↑  
(6)

*I'm the compiler && I can count to 6  
&& I wasn't fooled by that null terminator*

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

The `cstdlib` header declares a useful function:

```
int atoi(const char s[])
```

The `atoi` function converts a character array containing digits into its integer value:

```
char* year = "2012";  
int y = atoi(year);
```

*y* is the integer 2012

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

Unfortunately there is nothing like this for the `string` class!  
(can you believe that?!)

The `c_str` member function offers an "escape hatch":

```
string year = "2012";
int y = atoi(year.c_str());
```

Again, `y` is the integer 2012

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

You can access individual characters with the `[]` operator:

```
string name = "Harry";
name[3] = 'd';
```

*I'm the programmer && I changed Harry into Hardy!*

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

Converting from a C string to a C++ string is very easy:

```
string name = "Harry";
```

`name` is initialized with the C string `"Harry"`.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

You can write a function that will return  
the uppercase version of a `string`.

The `toupper` function is defined in the `cctype` header.

It converts lowercase characters to uppercase.  
(The `tolower` function does the opposite.)

```
char ch = toupper('a');
```

`ch` contains `'A'`

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

Up to this point, we have always used the  
`substr` member function to access individual  
characters in a C++ `string`:

```
string name = "Harry";
```

```
...name.substr(3, 1)...
```

yields a string of length 1  
containing the character at index 3  
(the second 'r')

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Converting Between C and C++ Strings

```
/**
 * Makes an uppercase version of a string.
 * @param str a string
 * @return a string with the characters in str converted to uppercase
 */
string uppercase(string str)
{
    string result = str; // Make a copy of str
    for (int i = 0; i < result.length(); i++)
    {
        // Convert each character to uppercase
        result[i] = toupper(result[i]);
    }
    return result;
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

C String Functions

Table 4 C String Functions	
In this table, s and t are character arrays; n is an integer.	
Function	Description
strlen(s)	Returns the length of s.
strcpy(t, s)	Copies the characters from s into t.
strncpy(t, s, n)	Copies at most n characters from s into t.
strcat(t, s)	Appends the characters from s after the end of the characters in t.
strncat(t, s, n)	Appends at most n characters from s after the end of the characters in t.
strcmp(s, t)	Returns 0 if s and t have the same contents, a negative integer if s comes before t in lexicographic order, a positive integer otherwise.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Dynamic Memory Allocation

The size of a *static* array must be known when you define it.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Dynamic Memory Allocation

In many programming situations, you know you will be working with several values.

You would normally use an array for this situation, right?

(yes)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Dynamic Memory Allocation

To solve this problem, you can use *dynamic allocation*.

Dynamic arrays are not static.

(Static, like all facts.)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Dynamic Memory Allocation

But suppose you do not know beforehand how many values you need.

So now can you use an array?

(oh dear!)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Dynamic Memory Allocation

To use dynamic arrays, you ask the C++ run-time system to create new space for an array whenever you need it.

*This is at RUN-TIME?  
On the fly?  
Arrays on demand!*

(cool)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

Where does this memory for my  
on-demand arrays come from?

The OS *keeps*  
a *heap*:  
a *Heap* O' RAM

(to give to good little programmers like you)  
(and poets)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

But just how useful is one single **double**?

(Not very)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

Yes, it's really called:

The Heap

(or sometimes the *freestore*  
– and it really is free!  
All you have to do is ask)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

How about a brand new array from that Heap O' RAM?

(Yes, please)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

To ask for more memory,  
say a **double**, you use the **new** operator:

**new double**

the runtime system seeks out room for  
a **double** on the heap, reserves it just for your  
use and returns a pointer to it.

This **double** location  
does not have a name.  
(this is run-time)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

To request a dynamic array you use the same **new**  
operator with some looks-like-an-array things added:

**new double[n]**

where **n** is the number of **doubles** you want  
and, again, you get a pointer to the array.

an array of **doubles** on demand!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

You need a pointer variable to hold the pointer you get:

```
double* account_pointer = new double;
double* account_array = new double[n];
```

Now you can use `account_array` as an array.

The magic of array/pointer duality  
lets you use the array notation  
`account_array[i]` to access the *i*th element.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

After you delete a memory block,  
you can no longer use it.  
The OS is very efficient – and quick – “your” storage  
space may already be used elsewhere.

```
delete[] account_array;
account_array[0] = 1000;
// NO! You no longer own the
// memory of account_array
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

When your program no longer needs the memory  
that you asked for with the `new` operator,  
you must return it to the heap  
using the `delete` operator for single areas of memory  
(which you would probably never use anyway).

```
delete account_pointer;
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

Unlike static arrays,  
which you are stuck with after you create them,  
you can change the size of a dynamic array.

Make a new, improved, bigger array  
and copy over the old data – but remember  
to delete what you no longer need.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

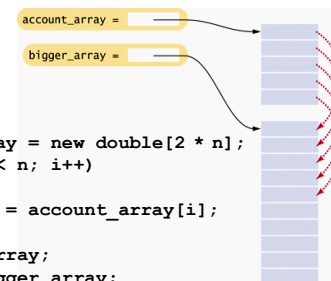
## Dynamic Memory Allocation

Or more likely, you allocated an array.  
So you must use the `delete[]` operator.

```
delete[] account_array;
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation – Resizing an Array



```
double* bigger_array = new double[2 * n];
for (int i = 0; i < n; i++)
{
    bigger_array[i] = account_array[i];
}
delete[] account_array;
account_array = bigger_array;
n = 2 * n;
```

(*n* is the variable used with the array)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved



## Dynamic Memory Allocation – Serious Business

Son,  
we need to talk.

We need to have a serious discussion about *safety*.

*Safety and security* are very important issues.

Really – THIS IS SERIOUS  
Sit down!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation – ON NO!!!

You could find  
yourself talking to  
strange arrays!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation – Serious Business

Son, heap allocation is a powerful feature,  
and you have proven yourself to be a responsible  
enough programmer to begin using dynamic arrays  
but you must be very careful to

follow these rules precisely:

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation



(not a strange array)

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation – THE RULES

1. Every call to `new` *must* be matched by exactly one call to `delete`.
2. Use `delete[]` to delete arrays.  
And always assign `NULL` to the pointer after that.
3. Don't access a memory block after it has been deleted.

If you don't follow these rules, your program can  
*crash or run unpredictably*

or worse...

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation



peas  
peas recycle  
spechilly dymambic membrities

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation



no. is not ribbly hebby.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Dangling Pointers – Serious Business

Son, there's more:

### DANGLING

*Dangling pointers* are when you use a pointer that has already been deleted or was never initialized.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation

### SYNTAX 7.2 Dynamic Memory Allocation

Capture the pointer in a variable.

Use the memory.

Delete the memory when you are done.

The new operator yields a pointer to a memory block of the given type.

Use this form to allocate an array of the given size (size need not be a constant).

Use the pointer as if it were an array.

Remember to use delete[] when deallocating the array.

```
int* var_ptr = new int;
...
*var_ptr = 1000;
...
delete var_ptr;

int* array_ptr = new int[size];
...
array_ptr[i] = 1000;
...
delete[] array_ptr;
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Dangling Pointers – Serious Business

```
int* values = new int[n];
// Process values
```

```
delete[] values;
```

```
// Some other work
values[0] = 42;
```

Good, son.  
Being responsible!

**Son!**  
**NO!!!**

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Dynamic Memory Allocation – Common Errors

Table 5 Common Memory Allocation Errors

Statements	Error
int* p; *p = 5; delete p;	There is no call to new int.
int* p = new int; *p = 5; p = new int;	The first allocated memory block was never deleted.
int* p = new int[10]; *p = 5; delete p;	The delete[] operator should have been used.
int* p = new int[10]; int* q = p; q[0] = 5; delete p; delete q;	The same memory block was deleted twice.
int n = 4; int* p = &n; *p = 5; delete p;	You can only delete memory blocks that you obtained from calling new.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Dangling Pointers – Serious Business

The value in an uninitialized or deleted pointer might point somewhere in the program you have no right to be accessing.

You can create real damage by writing to the location to which it points.

It's not yours to play with, son.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Dangling Pointers – Serious Business

Even just *reading* from that location  
can crash your program.

You've seen what's happened to other programs.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Dangling Pointers – Serious Business

Son, programming with pointers requires *iron discipline*.

- Always initialize pointer variables.
- If you can't initialize them with the return value of `new` or the `&` operator, then set them to `NULL`.
- Never use a pointer that has been deleted.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Dangling Pointers – Serious Business

Remember what happened to Jimmy?  
A dialog box with a bomb icon.

And Ralph?  
“General protection fault.”

And poor Henry's son?  
“Segmentation fault” came up,  
and the program *was terminated*.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Memory Leaks – Serious Business

And Son, I'm sorry to say, there's even more:

#### LEAKS

A *memory leak* is when use `new` to get dynamic memory but you fail to delete it when you are done.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Dangling Pointers – Serious Business

Or worse, son – you could hurt *yourself*!

If that dangling pointer points at your own data,  
and you write to it –

you may very well have messed up your own  
future,  
your own data!

Just don't do it, son!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

### Common Errors Memory Leaks – Serious Business

I know, I know, you think that a few `doubles`  
and a couple of `strings` left on the heap  
now and then doesn't really hurt anyone.

But son, what if everyone did this?  
Think of a loop – 10,000 times you grab just a few bytes  
from the heap and don't give them back!

What happens when there's no more heap  
for the OS to give you?

Just give it up, son – give back what you no longer need.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Memory Leaks – Serious Business

### Remember Rule #1.

1. Every call to `new` *must* be matched by exactly one call to `delete`.

And after deleting, set it to `NULL` so that it can be tested for danger later.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Arrays and Vectors of Pointers

When you have a sequence of pointers, you can place them into an array or vector.

An array and a vector of ten `int*` pointers are defined as

```
int* pointer_array[10];  
vector<int*> pointer_vector(10);
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Dangling Pointers – Serious Business

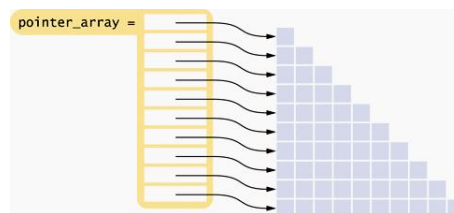
```
int* values = new int[n];  
// Process values  
  
delete[] values;  
values = NULL;  
  
later...  
if values = NULL ...
```

Very good, son.  
Being very responsible!

Great!

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Arrays and Vectors of Pointers – A Triangular Array



In this array, each row is a different length.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Common Errors Memory Leaks – Serious Business

Son, I think you are ready to go on...

## Arrays and Vectors of Pointers – A Triangular Array

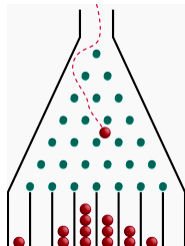
In this situation, it would not be very efficient to use a two-dimensional array, because almost half of the elements would be wasted.



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

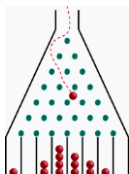
## A Galton Board Simulation

The Galton board can only show the balls in the bins, but we can do better by keeping a counter for *each* peg, incrementing it as a ball travels past it.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

We will develop a program that uses a triangular array to simulate a Galton board.



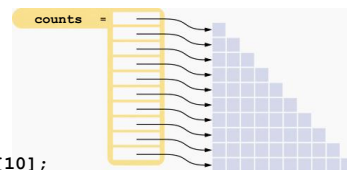
C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

We will simulate a board with ten rows of pegs. Each row requires an array of counters. The following statements initialize the triangular array:

```
int* counts[10];
for (int i = 0; i < 10; i++)
{
    counts[i] = new int[i + 1];
}
```

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved



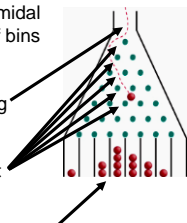
## A Galton Board Simulation

A Galton board consists of a pyramidal arrangement of pegs and a row of bins at the bottom.

Balls are dropped onto the top peg and travel toward the bins.

At each peg, there is a 50 percent chance of moving left or right.

The balls in the bins approximate a bell-curve distribution.

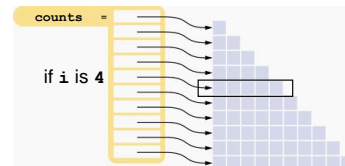


C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

We will need to print each row:

```
// print all elements in the ith row
for (int j = 0; j <= i; j++)
{
    cout << setw(4) << counts[i][j];
}
cout << endl;
```

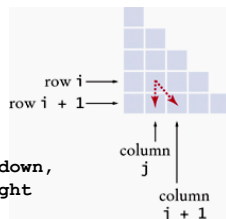


C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

We will simulate a ball bouncing through the pegs:

```
int r = rand() % 2;
// If r is even, move down,
// otherwise to the right
if (r == 1)
{
    j++;
}
counts[i][j]++;
```



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

```
// Print all counts
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j <= i; j++)
    {
        cout << setw(4) << counts[i][j];
    }
    cout << endl;
}

// Deallocate the rows
for (int i = 0; i < 10; i++)
{
    delete[] counts[i];
}

return 0;
```

ch07/galton.cpp

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
    srand(time(0));
    int* counts[10];

    // Allocate the rows
    for (int i = 0; i < 10; i++)
    {
        counts[i] = new int[i + 1];
        for (int j = 0; j <= i; j++)
        {
            counts[i][j] = 0;
        }
    }
}
```

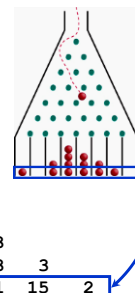
ch07/galton.cpp

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

This is the output from a run of the program:

```
1000
480 520
241 500 259
124 345 411 120
68 232 365 271 64
32 164 283 329 161 31
16 88 229 303 254 88 22
9 47 147 277 273 190 44 13
5 24 103 203 288 228 113 33 3
1 18 64 149 239 265 186 61 15 2
```



C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## A Galton Board Simulation

```
const int RUNS = 1000;
// Simulate 1,000 balls
for (int run = 0; run < RUNS; run++)
{
    // Add a ball to the top
    counts[0][0]++;
    // Have the ball run to the bottom
    int j = 0;
    for (int i = 1; i < 10; i++)
    {
        int r = rand() % 2;
        // If r is even, move down,
        // otherwise to the right
        if (r == 1)
        {
            j++;
        }
        counts[i][j]++;
    }
}
```

ch07/galton.cpp

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Chapter Summary

### Define and use pointer variables.

- A pointer denotes the location of a variable in memory.
- The type `T*` denotes a pointer to a variable of type `T`.
- The `&` operator yields the location of a variable.
- The `*` operator accesses the variable to which a pointer points.
- It is an error to use an uninitialized pointer.
- The `NULL` pointer does not point to any object.



### Understand the relationship between arrays and pointers in C++.

- The name of an array variable is a pointer to the starting element of the array.
- Pointer arithmetic means adding an integer offset to an array pointer, yielding a pointer that skips past the given number of elements.
- The array/pointer duality law states that `a[n]` is identical to `*(a + n)`, where `a` is a pointer into an array and `n` is an integer offset.
- When passing an array to a function, only the starting address is passed.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Chapter Summary

### Use C++ string objects with functions that process character arrays.

W<sub>0</sub> O<sub>1</sub> R<sub>2</sub> D<sub>3</sub>

- A value of type `char` denotes an individual character. Character literals are enclosed in single quotes.
- A literal string (enclosed in double quotes) is an array of `char` values with a zero terminator.
- Many library functions use pointers of type `char*`.
- The `c_str` member function yields a `char*` pointer from a `string` object.
- You can initialize C++ string variables with C strings.
- You can access characters in a C++ string object with the `[]` operator.

char\_pointer ←

Points to "WRD"

"W"	[0]
"R"	[1]
"D"	[2]
"\0"	[3]
"y"	[4]
"n"	[5]



End Chapter Seven, Part II

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

Slides by Evan Gallagher  
C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Chapter Summary

### Allocate and deallocate memory in programs whose memory requirements aren't known until run time.

- Use dynamic memory allocation if you do not know in advance how many values you need.
- The `new` operator allocates memory from the heap.
- You must reclaim dynamically allocated objects with the `delete` or `delete[]` operator.
- Using a dangling pointer (a pointer that points to memory that has been deleted) is a serious programming error.
- Every call to `new` should have a matching call to `delete`.

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved

## Chapter Cleanup



than you fa recyblin

C++ for Everyone by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved