Non-Exact Comparison of Floating-Point Numbers – SOLUTION

It is common to set ε to 10^{-14} when comparing `double` numbers:

```
const double EPSILON = 1E-14;
double r = sqrt(2.0);
if (fabs(r * r - 2) < EPSILON)
{
   cout << "sqrt(2) squared is approximately ";
}
```

$r^2 == 2$

Include the `<cmath>` header to use `sqrt` and the `fabs` function which gives the absolute value.

Multiple Alternatives (3.3)

if it's quicker to the candy mountain,
    we'll go that way
else
    we go that way
but what about that way?

Multiple Alternatives

Multiple `if` statements can be combined to evaluate complex decisions.

Multiple Alternatives

EX:

How would we write code to deal with Richter scale values?

Multiple Alternatives

Table 3 Richter Scale

Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

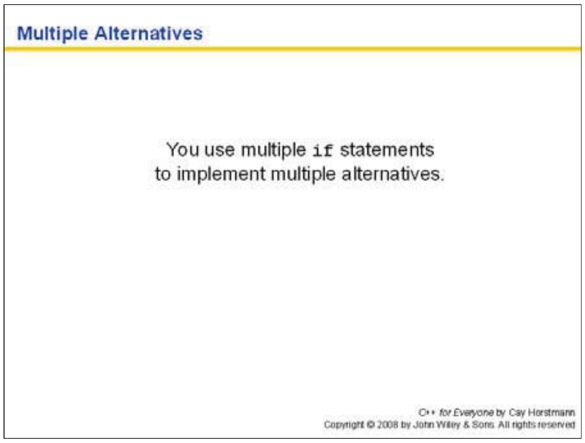
Multiple Alternatives

In this case, there are five branches:
one each for the four descriptions of damage,

Table 3 Richter Scale

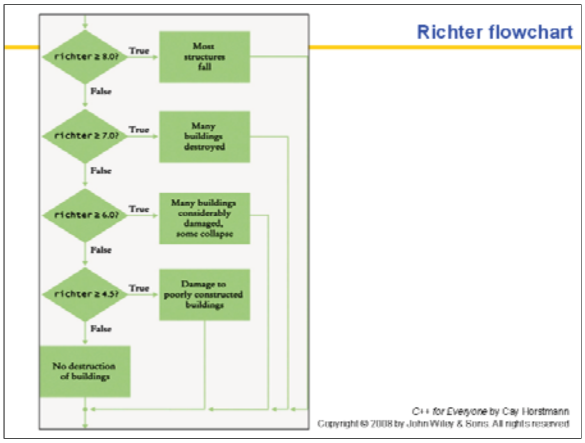
Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

and one for no destruction.

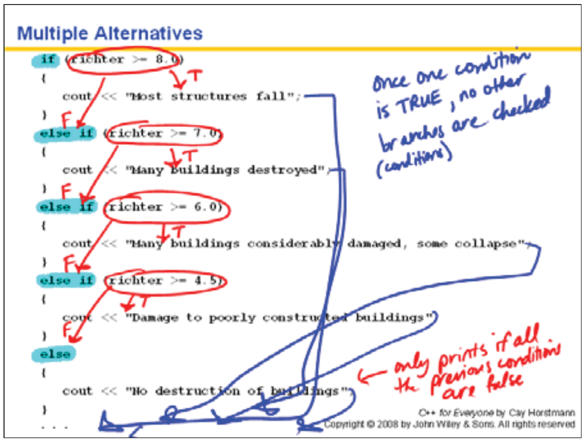


Multiple Alternatives

You use multiple `if` statements
to implement multiple alternatives.



Richter flowchart

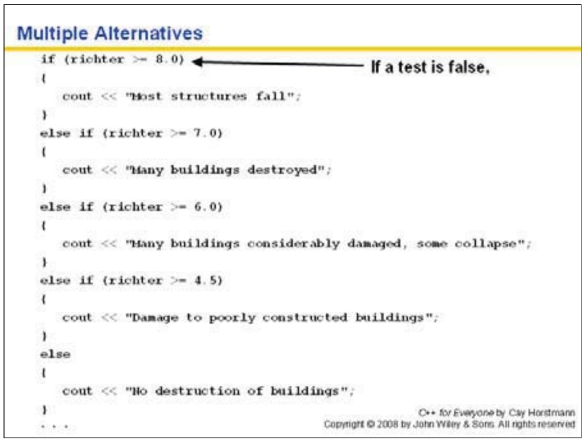


Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

Once one condition is TRUE, no other branches are checked (conditions)

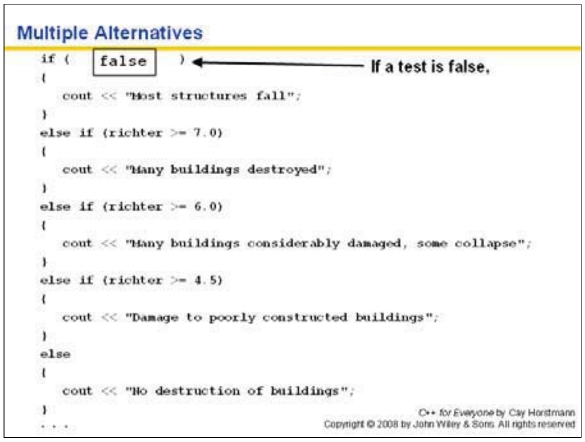
← only prints if all the previous conditions are false



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

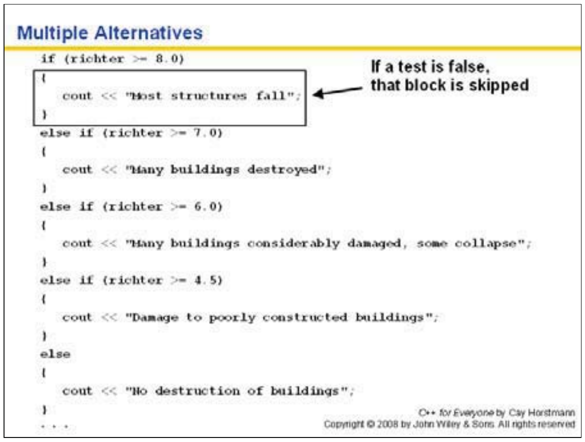
If a test is false,



Multiple Alternatives

```
if (  false  )
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

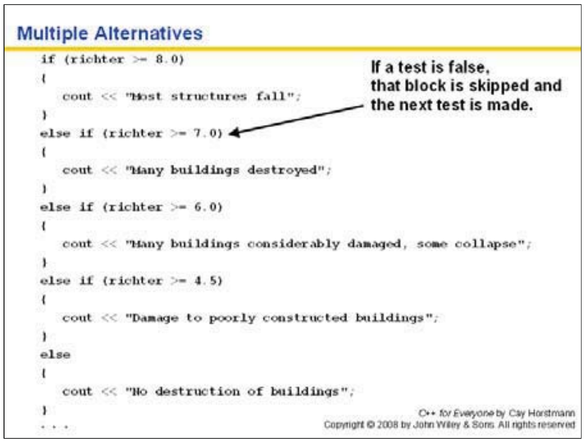
If a test is false,



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

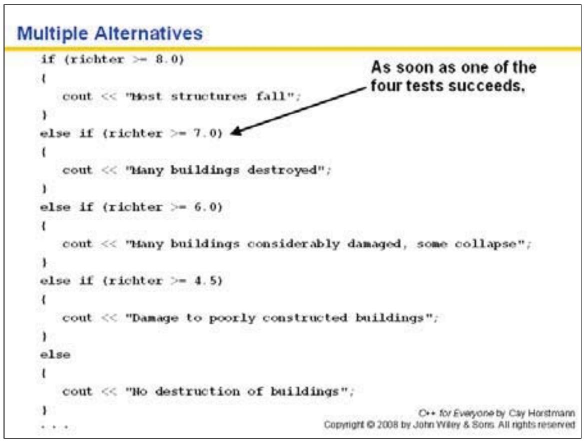
If a test is false,
that block is skipped



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

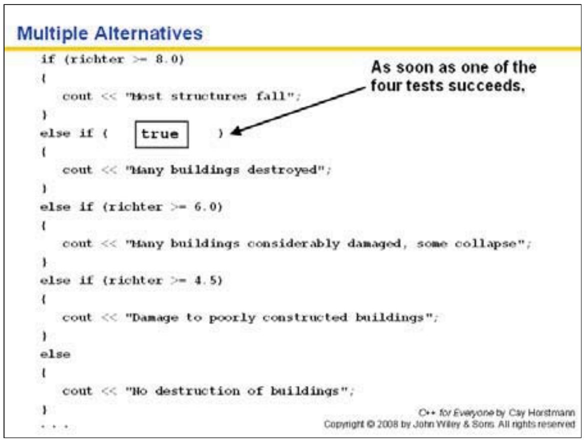
If a test is false, that block is skipped and the next test is made.



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

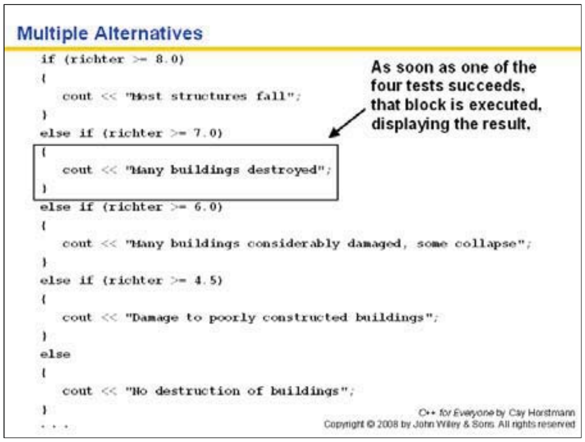
As soon as one of the four tests succeeds,



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (   true   )
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

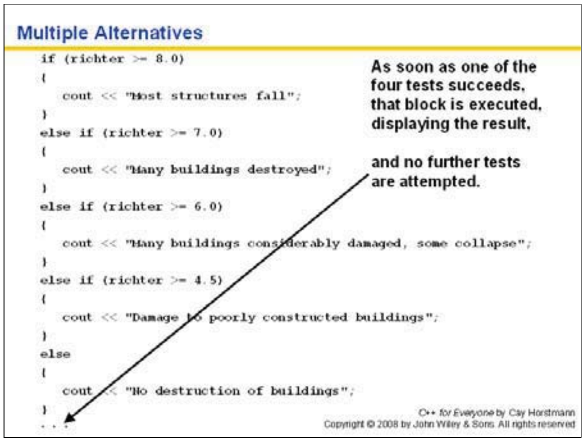
As soon as one of the four tests succeeds,



Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

As soon as one of the four tests succeeds, that block is executed, displaying the result,

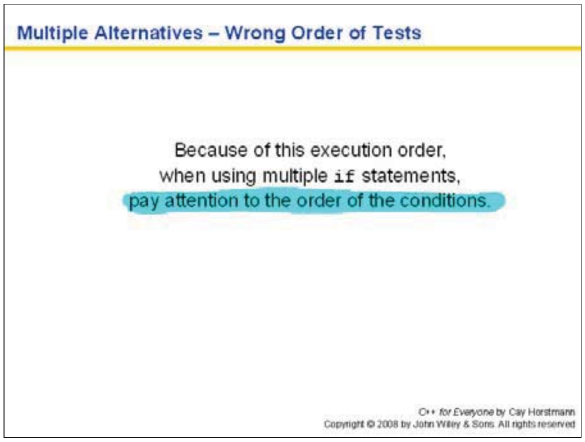


Multiple Alternatives

```
if (richter >= 8.0)
{
   cout << "Most structures fall";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 4.5)
{
   cout << "Damage to poorly constructed buildings";
}
else
{
   cout << "No destruction of buildings";
}
. . .
```

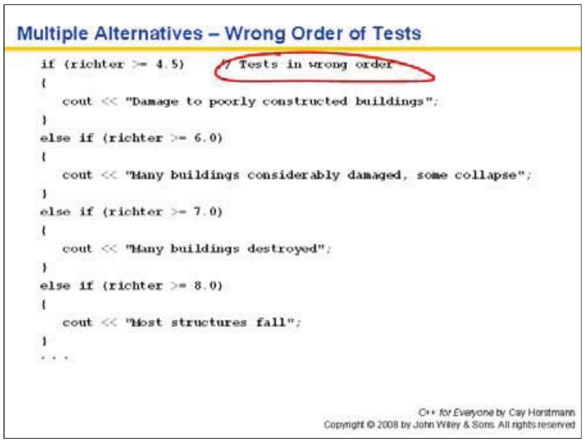
As soon as one of the four tests succeeds, that block is executed, displaying the result,

and no further tests are attempted.



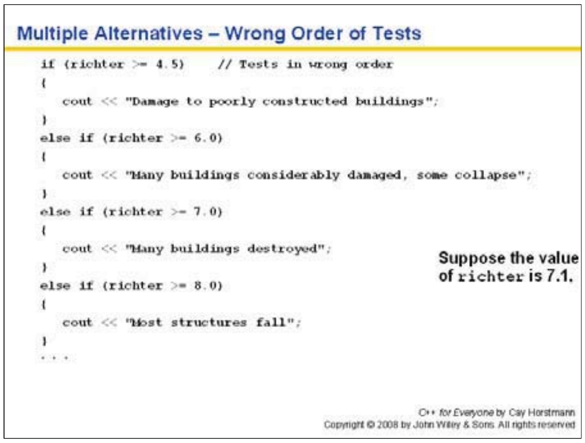
Multiple Alternatives – Wrong Order of Tests

Because of this execution order,
when using multiple `if` statements,
pay attention to the order of the conditions.



Multiple Alternatives – Wrong Order of Tests

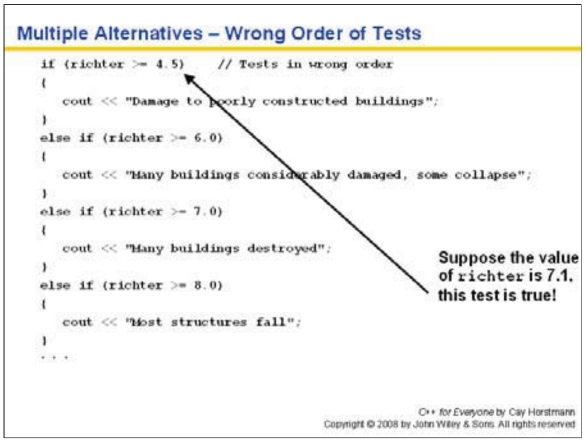
```
if (richter >= 4.5)    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```



Multiple Alternatives – Wrong Order of Tests

```
if (richter >= 4.5)    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```

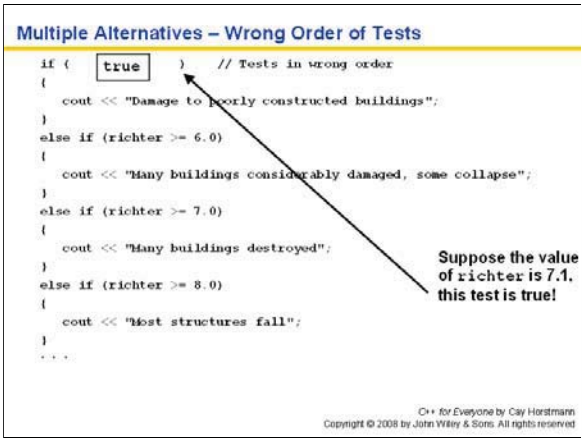
Suppose the value of `richter` is 7.1,



Multiple Alternatives – Wrong Order of Tests

```
if (richter >= 4.5)    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```

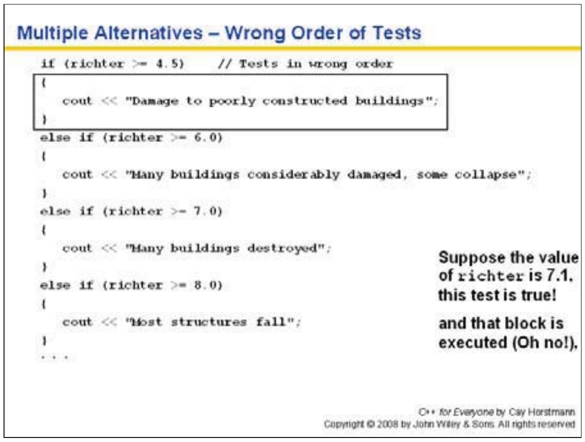
Suppose the value of `richter` is 7.1, this test is true!



Multiple Alternatives – Wrong Order of Tests

```
if (  true  )    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```

Suppose the value of `richter` is 7.1, this test is true!

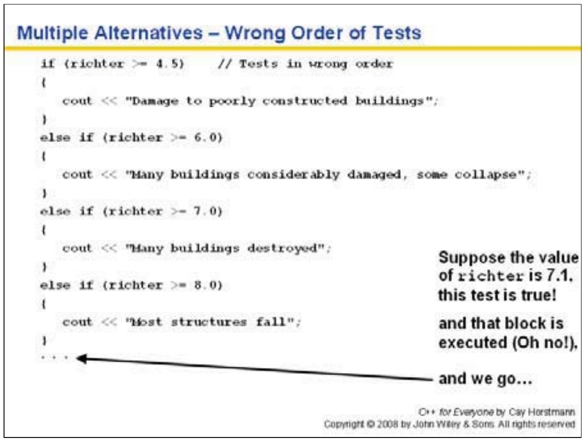


Multiple Alternatives – Wrong Order of Tests

```
if (richter >= 4.5)    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```

Suppose the value of `richter` is 7.1, this test is true!

and that block is executed (Oh no!),



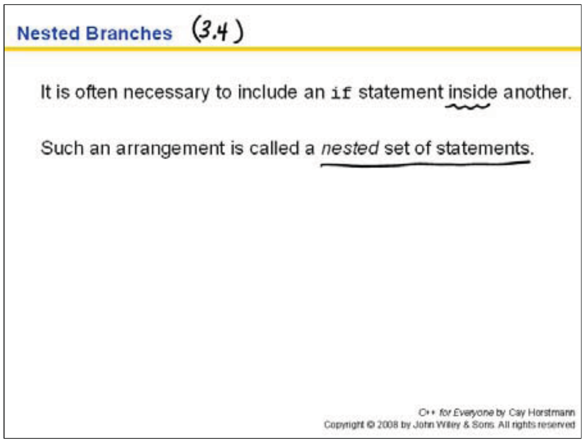
Multiple Alternatives – Wrong Order of Tests

```
if (richter >= 4.5)    // Tests in wrong order
{
   cout << "Damage to poorly constructed buildings";
}
else if (richter >= 6.0)
{
   cout << "Many buildings considerably damaged, some collapse";
}
else if (richter >= 7.0)
{
   cout << "Many buildings destroyed";
}
else if (richter >= 8.0)
{
   cout << "Most structures fall";
}
. . .
```

Suppose the value of `richter` is 7.1, this test is true!

and that block is executed (Oh no!),

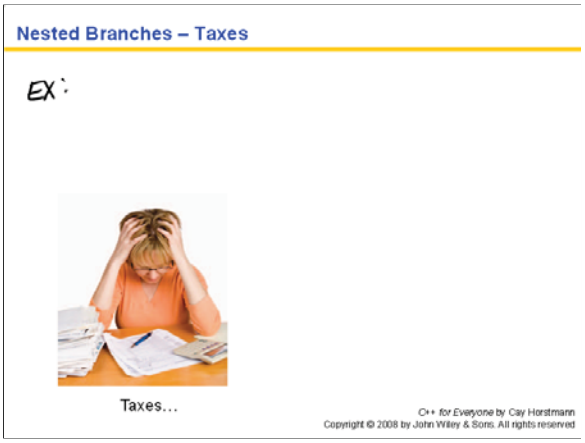
and we go…



Nested Branches (3.4)

It is often necessary to include an `if` statement inside another.

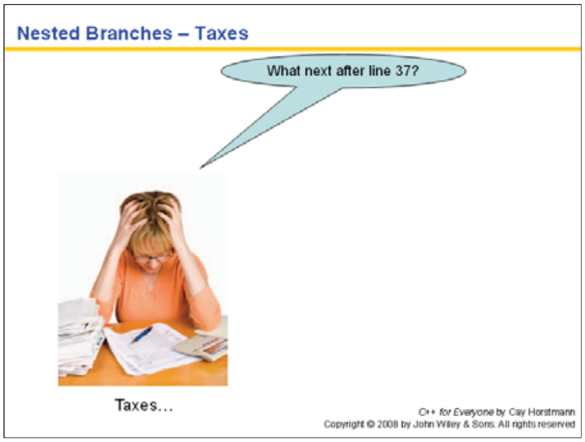
Such an arrangement is called a *nested* set of statements.



Nested Branches – Taxes

EX:

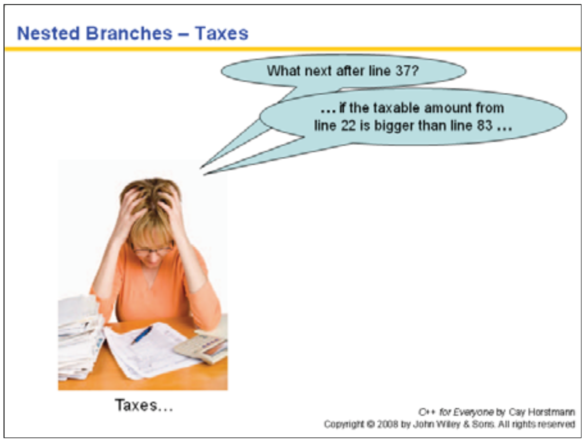
Taxes…



Nested Branches – Taxes

What next after line 37?

Taxes…

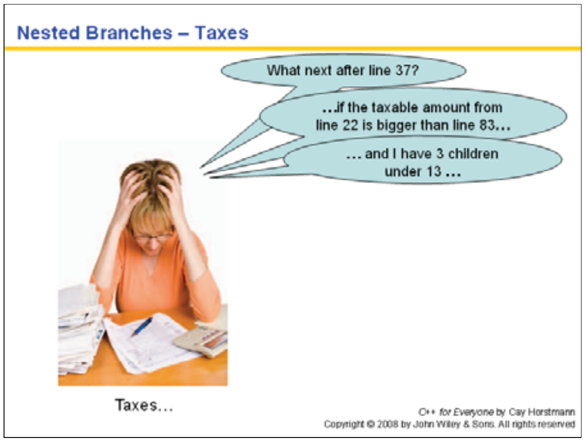


Nested Branches – Taxes

What next after line 37?

… if the taxable amount from line 22 is bigger than line 83 …

Taxes…



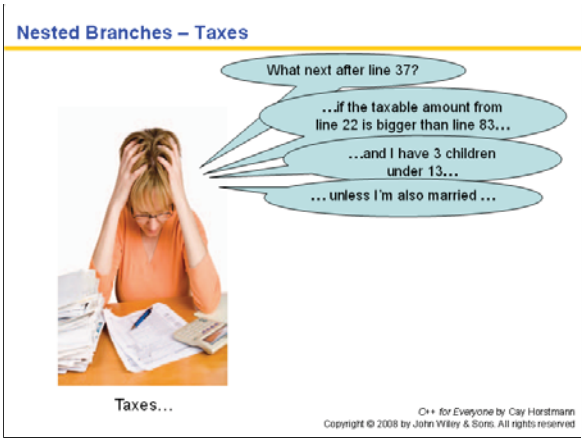
Nested Branches – Taxes

What next after line 37?

…if the taxable amount from line 22 is bigger than line 83…

… and I have 3 children under 13 …

Taxes…



Nested Branches – Taxes

What next after line 37?

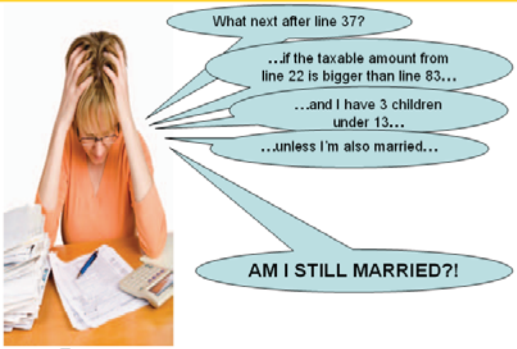
…if the taxable amount from line 22 is bigger than line 83…

…and I have 3 children under 13…

… unless I'm also married …

Taxes…

Nested Branches – Taxes


What next after line 37?
…if the taxable amount from line 22 is bigger than line 83…
…and I have 3 children under 13…
…unless I'm also married…
AM I STILL MARRIED?!

Taxes…

Nested Branches – Taxes

- In the United States different tax rates are used depending on the taxpayer's marital status.
- There are different tax schedules for single and for married taxpayers.
- Married taxpayers add their income together and pay taxes on the total.

Nested Branches – Taxes

Let's write the code.

First, as always, we analyze the problem.

Nested Branches – Taxes

Nested branching analysis is aided by drawing tables showing the different criteria.

Thankfully, the I.R.S. has done this for us.

Nested Branches – Taxes

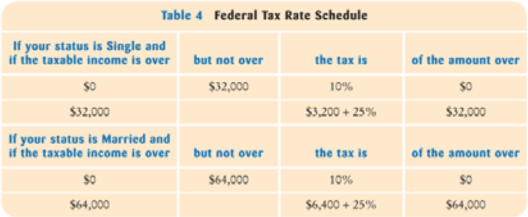


Table 4 Federal Tax Rate Schedule

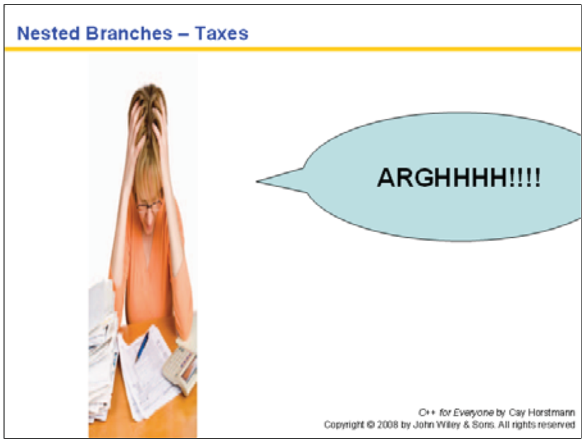
If your status is Single and if the taxable income is over	but not over	the tax is	of the amount over
$0	$32,000	10%	$0
$32,000		$3,200 + 25%	$32,000
If your status is Married and if the taxable income is over	but not over	the tax is	of the amount over
$0	$64,000	10%	$0
$64,000		$6,400 + 25%	$64,000

Tax brackets for single filers:
from $0 to $32,000
above $32,000
then tax depends on income

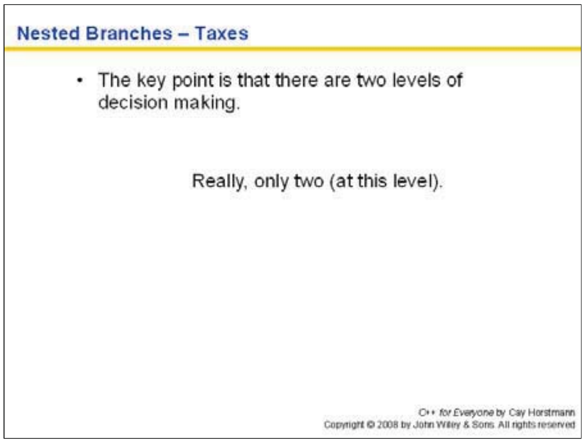
Tax brackets for married filers:
from $0 to $64,000
above $64,000
then tax depends on income

Nested Branches – Taxes

Now that you understand,
given a filing status and an income figure,
compute the taxes due.



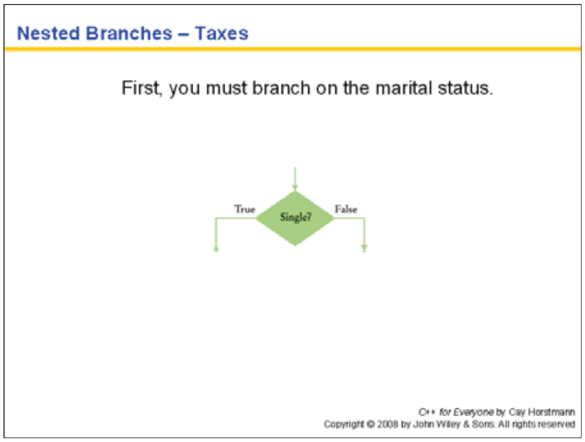
Nested Branches – Taxes



Nested Branches – Taxes

- The key point is that there are two levels of decision making.

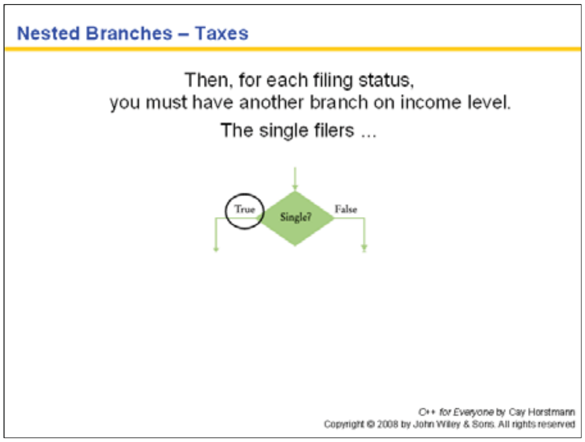
Really, only two (at this level).



Nested Branches – Taxes

First, you must branch on the marital status.

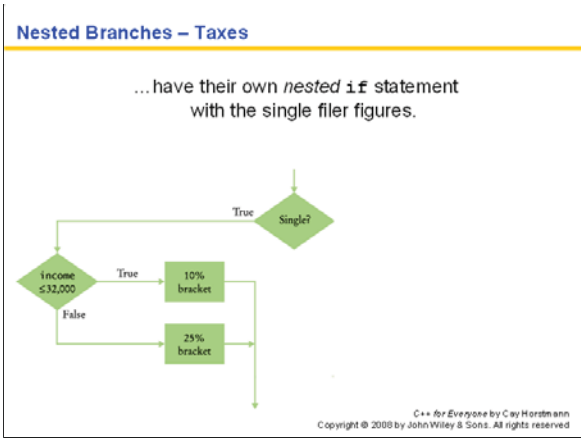
True
Single?
False



Nested Branches – Taxes

Then, for each filing status,
you must have another branch on income level.
The single filers …

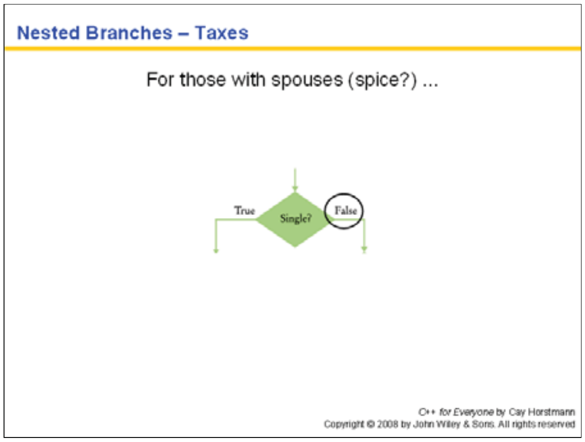
True
Single?
False



Nested Branches – Taxes

… have their own *nested* `if` statement
with the single filer figures.

True
Single?
income ≤32,000
True
10% bracket
False
25% bracket



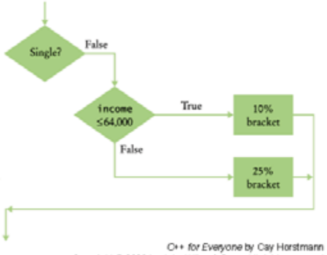
Nested Branches – Taxes

For those with spouses (spice?) …

True
Single?
False

Nested Branches – Taxes

…a different *nested* `if` for using their figures.



Nested Branches – Taxes

In theory you can have even deeper levels of nesting.

Consider:

first by state
then by filing status
then by income level

This situation requires three levels of nesting.

Nested Branches – Taxes

ch03/tax.cpp

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   const double RATE1 = 0.10;
   const double RATE2 = 0.25;
   const double RATE1_SINGLE_LIMIT = 32000;
   const double RATE1_MARRIED_LIMIT = 64000;

   double tax1 = 0;
   double tax2 = 0;

   double income;
   cout << "Please enter your income: ";
   cin >> income;

   cout << "Please enter s for single, m for married: ";
   string marital_status;
   cin >> marital_status;
```

Nested Branches – Taxes

```
   if (marital_status == "s")
   {
                                   32000
      if (income <= RATE1_SINGLE_LIMIT)
      {
         tax1 = RATE1 * income;
      }
      else
      {           10%          32000
         tax1 = RATE1 * RATE1_SINGLE_LIMIT;
         tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
                  25%       18000
      }
   }
   else
      if married
```

50000
-32000
18,000

Nested Branches – Taxes

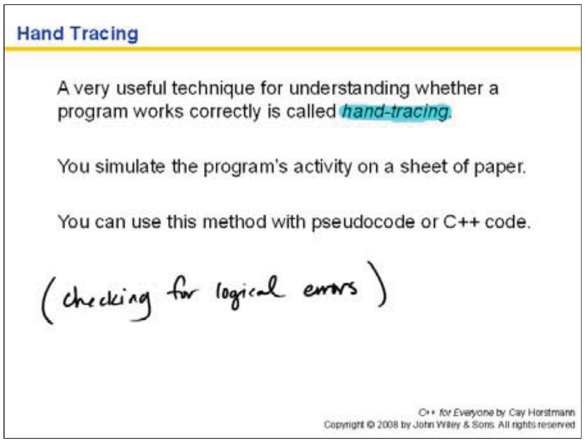
```
   {                              64000
      if (income <= RATE1_MARRIED_LIMIT)
      {
         tax1 = RATE1 * income;
      }
      else
      {
         tax1 = RATE1 * RATE1_MARRIED_LIMIT;
         tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
      }
   }

   double total_tax = tax1 + tax2;

   cout << "The tax is $" << total_tax << endl;
   return 0;
}
```

Nested Branches – Taxes

In practice two levels of nesting should be enough.
beyond that you should be calling your own functions

– but you don't know to write functions…

…yet



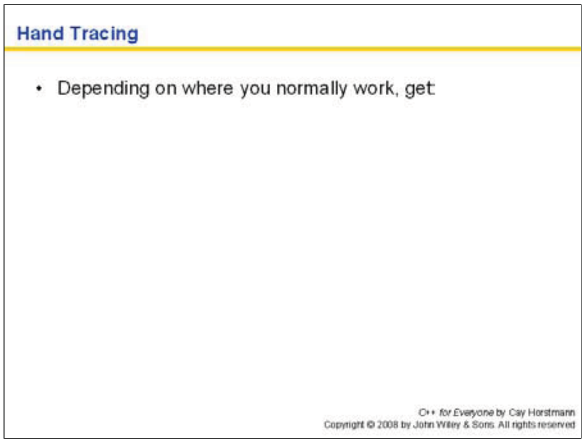
Hand Tracing

A very useful technique for understanding whether a program works correctly is called *hand-tracing*.

You simulate the program's activity on a sheet of paper.

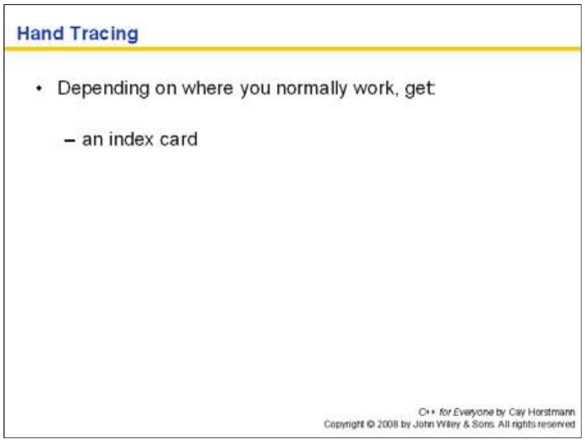
You can use this method with pseudocode or C++ code.

(checking for logical errors)



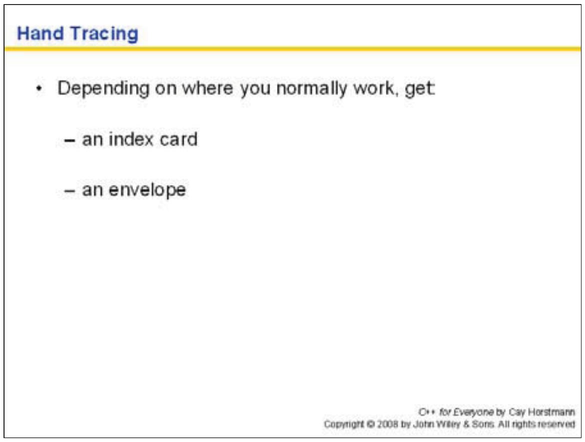
Hand Tracing

- Depending on where you normally work, get:



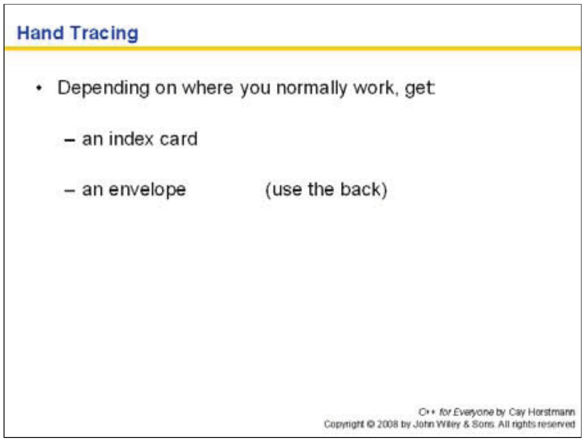
Hand Tracing

- Depending on where you normally work, get:
  - an index card



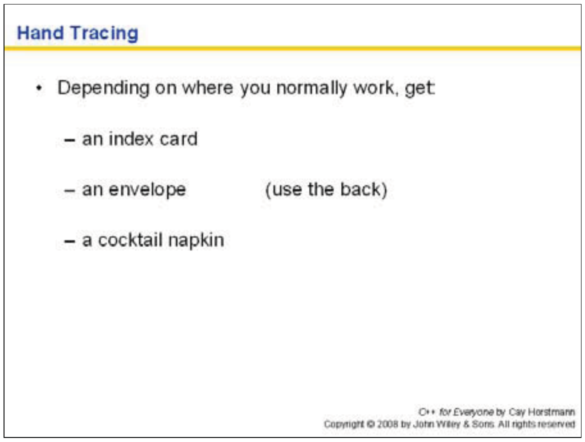
Hand Tracing

- Depending on where you normally work, get:
  - an index card
  - an envelope



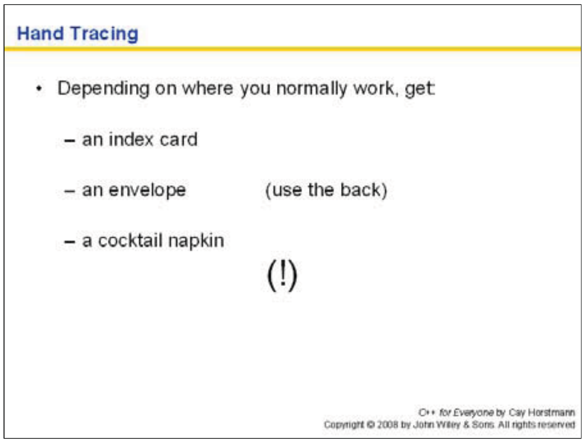
Hand Tracing

- Depending on where you normally work, get:
  - an index card
  - an envelope (use the back)



Hand Tracing

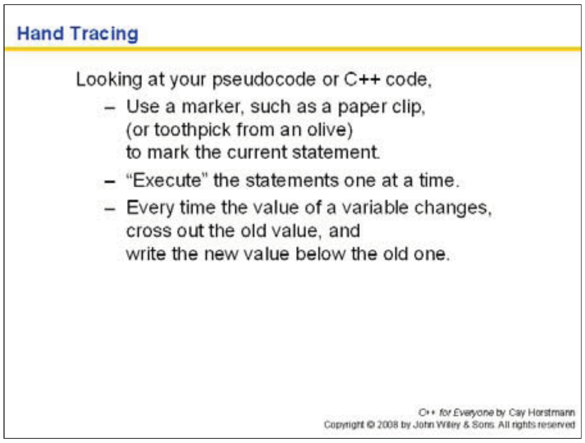
- Depending on where you normally work, get:
  - an index card
  - an envelope (use the back)
  - a cocktail napkin



Hand Tracing

- Depending on where you normally work, get:
    - an index card
    - an envelope (use the back)
    - a cocktail napkin

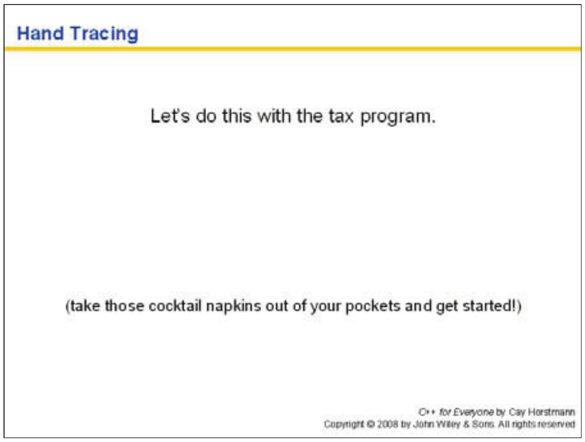
(!)



Hand Tracing

Looking at your pseudocode or C++ code,

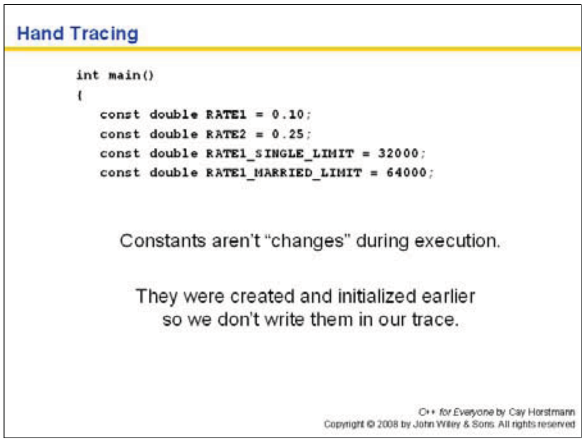
- Use a marker, such as a paper clip, (or toothpick from an olive) to mark the current statement.
- "Execute" the statements one at a time.
- Every time the value of a variable changes, cross out the old value, and write the new value below the old one.



Hand Tracing

Let's do this with the tax program.

(take those cocktail napkins out of your pockets and get started!)

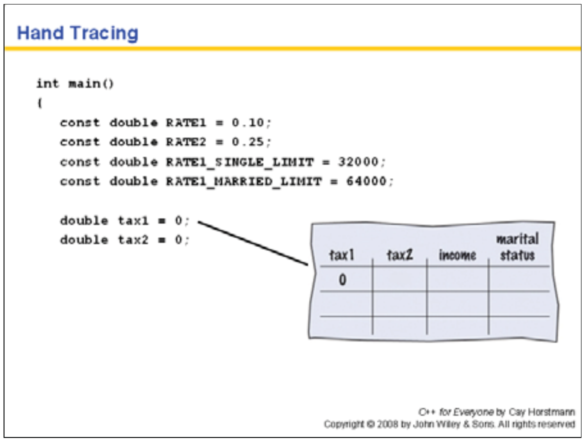


Hand Tracing

```
int main()
{
   const double RATE1 = 0.10;
   const double RATE2 = 0.25;
   const double RATE1_SINGLE_LIMIT = 32000;
   const double RATE1_MARRIED_LIMIT = 64000;
```

Constants aren't "changes" during execution.

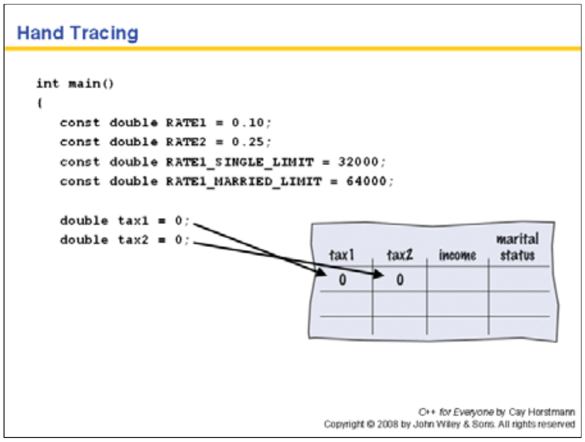
They were created and initialized earlier so we don't write them in our trace.



Hand Tracing

```
int main()
{
   const double RATE1 = 0.10;
   const double RATE2 = 0.25;
   const double RATE1_SINGLE_LIMIT = 32000;
   const double RATE1_MARRIED_LIMIT = 64000;

   double tax1 = 0;
   double tax2 = 0;
```

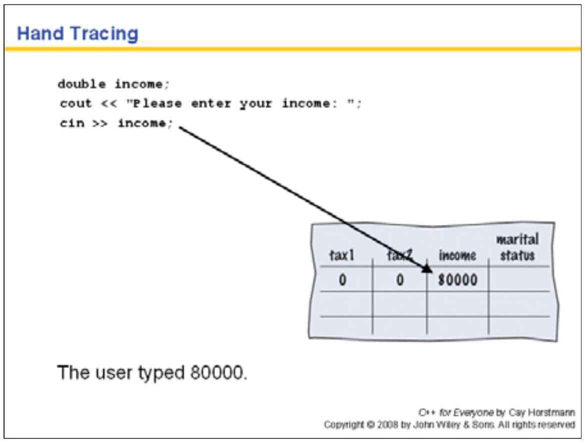
tax1	tax2	income	marital status
0			



Hand Tracing

```
int main()
{
   const double RATE1 = 0.10;
   const double RATE2 = 0.25;
   const double RATE1_SINGLE_LIMIT = 32000;
   const double RATE1_MARRIED_LIMIT = 64000;

   double tax1 = 0;
   double tax2 = 0;
```

tax1	tax2	income	marital status
0	0		

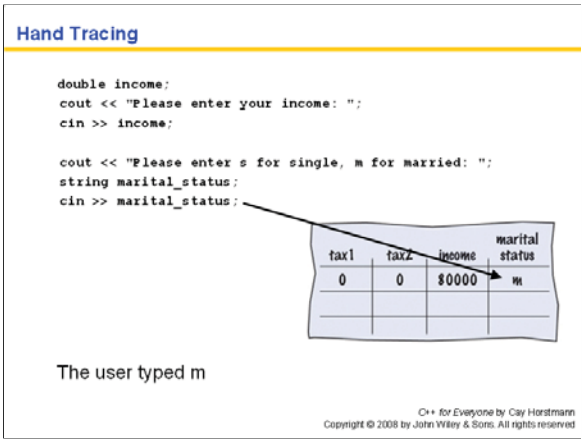


Hand Tracing

```
double income;
cout << "Please enter your income: ";
cin >> income;
```

tax1	tax2	income	marital status
0	0	80000	

The user typed 80000.

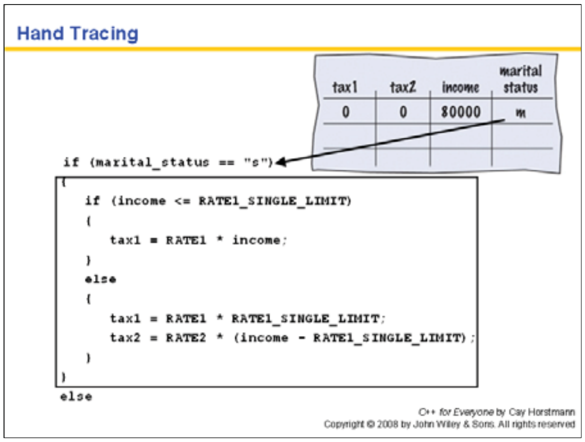


Hand Tracing

```
double income;
cout << "Please enter your income: ";
cin >> income;

cout << "Please enter s for single, m for married: ";
string marital_status;
cin >> marital_status;
```

tax1	tax2	income	marital status
0	0	80000	m

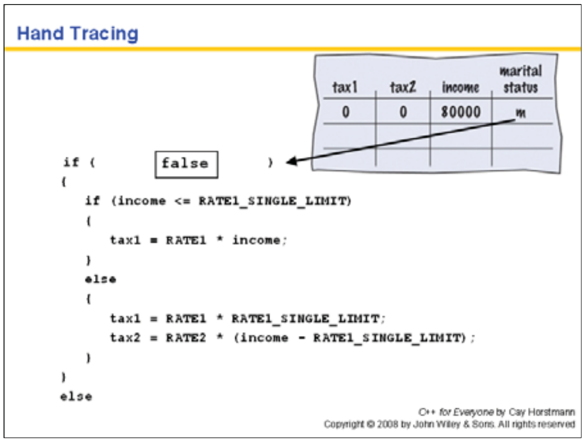
The user typed m



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

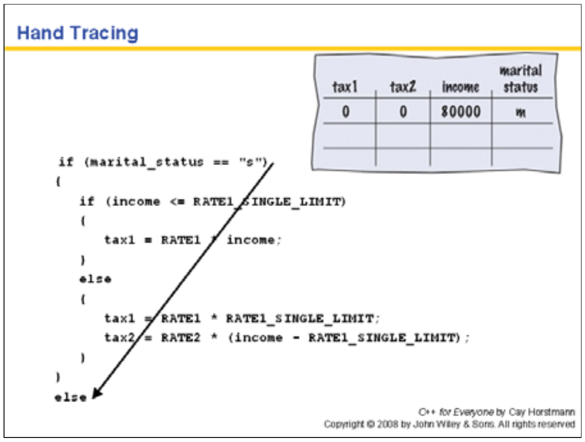
```
if (marital_status == "s")
{
   if (income <= RATE1_SINGLE_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_SINGLE_LIMIT;
      tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
   }
}
else
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

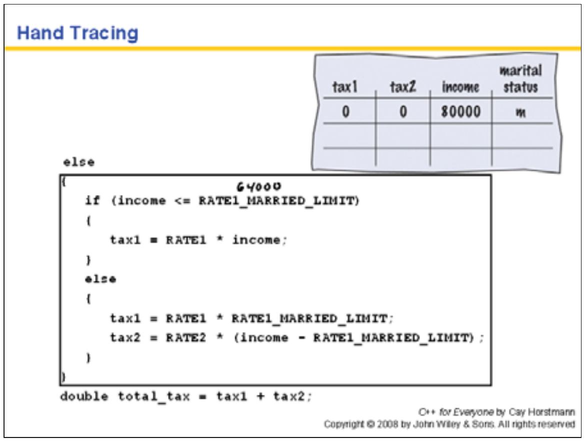
```
if (      false      )
{
   if (income <= RATE1_SINGLE_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_SINGLE_LIMIT;
      tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
   }
}
else
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

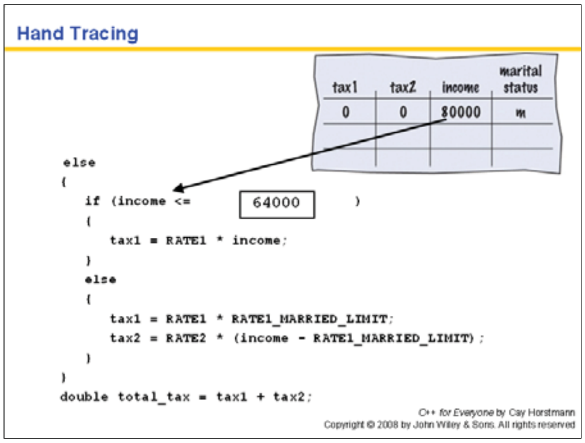
```
if (marital_status == "s")
{
   if (income <= RATE1_SINGLE_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_SINGLE_LIMIT;
      tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT);
   }
}
else
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

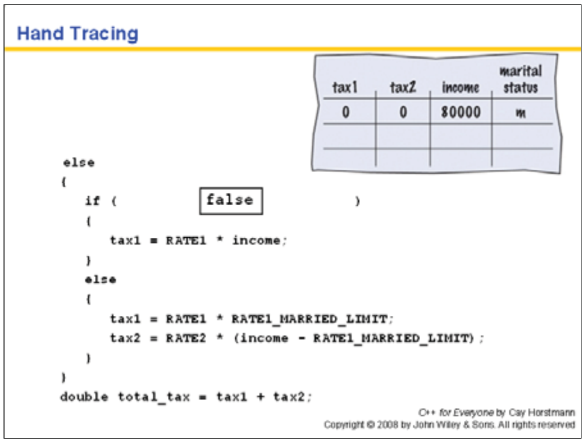
```
else
{
                        64000
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

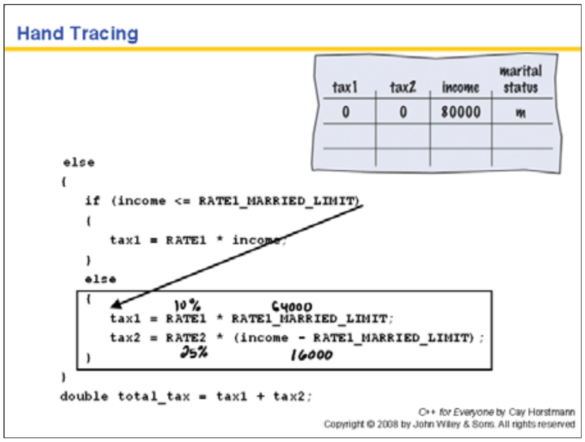
```
else
{
   if (income <=        64000        )
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

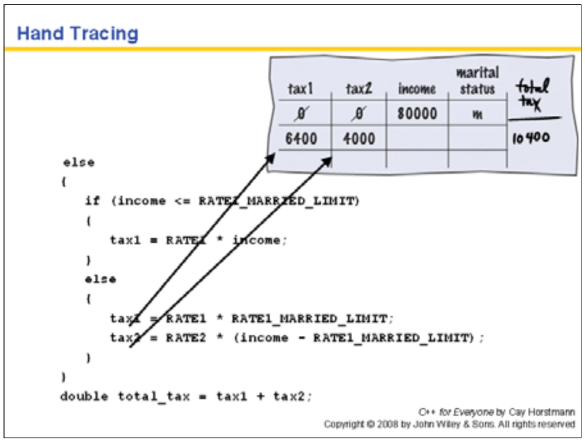
```
else
{
   if (          false            )
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status
0	0	80000	m

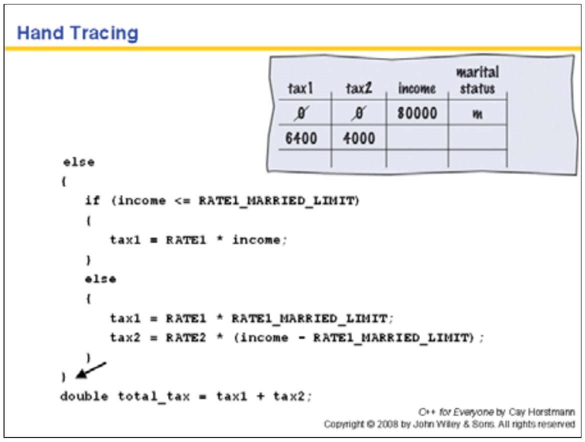
```
else
{
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {           10%          64000
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }           25%            16000
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status	total tax
~~0~~	~~0~~	80000	m	
6400	4000			10400

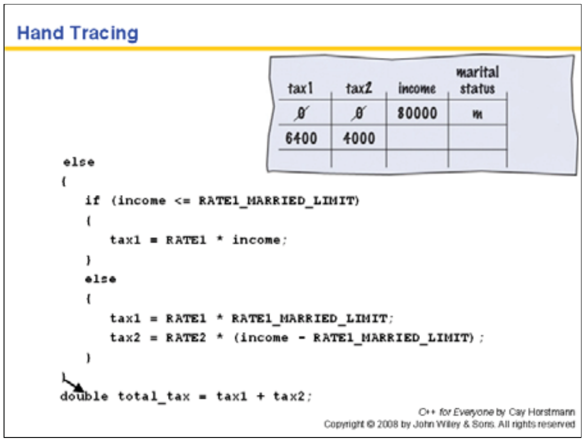
```
else
{
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status
~~0~~	~~0~~	80000	m
6400	4000		

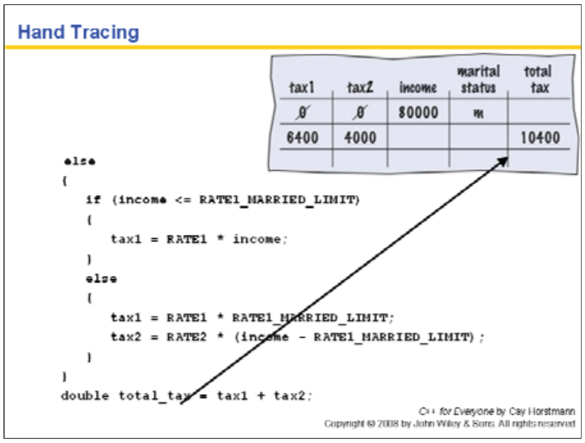
```
else
{
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```



Hand Tracing

tax1	tax2	income	marital status
~~0~~	~~0~~	80000	m
6400	4000		

```
else
{
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
   }
}
double total_tax = tax1 + tax2;
```

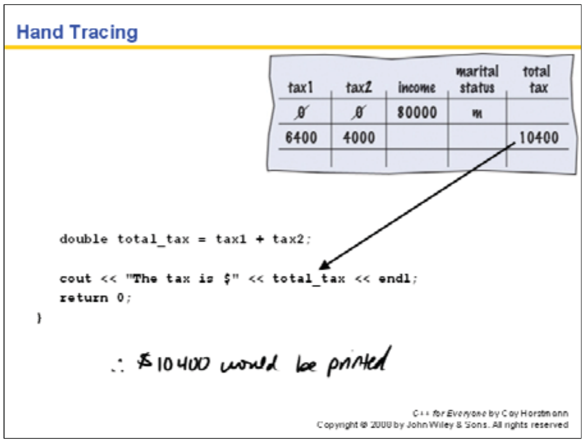


Hand Tracing

tax1	tax2	income	marital status	total tax
~~0~~	~~0~~	80000	m	
6400	4000			10400

```
else
{
   if (income <= RATE1_MARRIED_LIMIT)
   {
      tax1 = RATE1 * income;
   }
   else
   {
      tax1 = RATE1 * RATE1_MARRIED_LIMIT;
      tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT) ;
   }
}
double total_tax = tax1 + tax2;
```

C++ *for Everyone* by Cay Horstmann



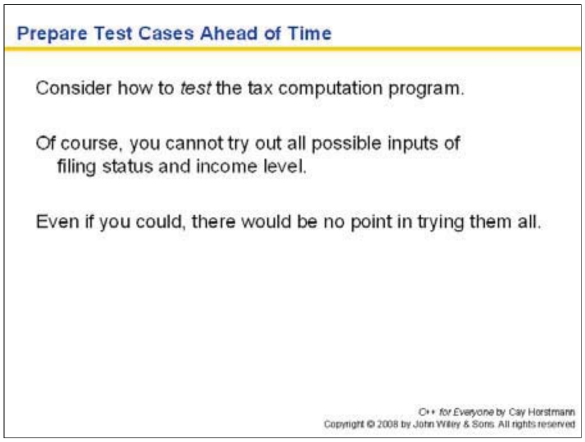
Hand Tracing

tax1	tax2	income	marital status	total tax
~~0~~	~~0~~	80000	m	
6400	4000			10400

```
   double total_tax = tax1 + tax2;

   cout << "The tax is $" << total_tax << endl;
   return 0;
}
```

∴ $10400 would be printed

C++ *for Everyone* by Cay Horstmann



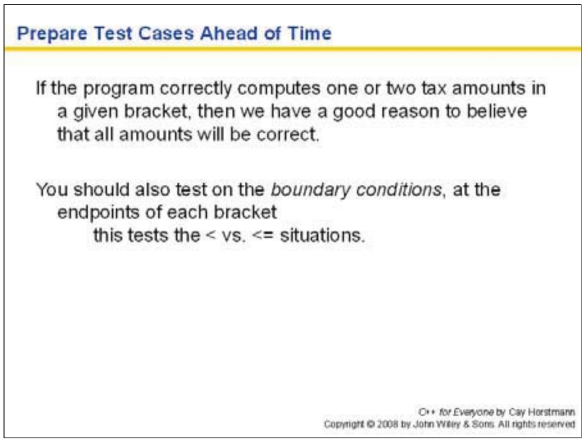
Prepare Test Cases Ahead of Time

Consider how to *test* the tax computation program.

Of course, you cannot try out all possible inputs of filing status and income level.

Even if you could, there would be no point in trying them all.

C++ *for Everyone* by Cay Horstmann



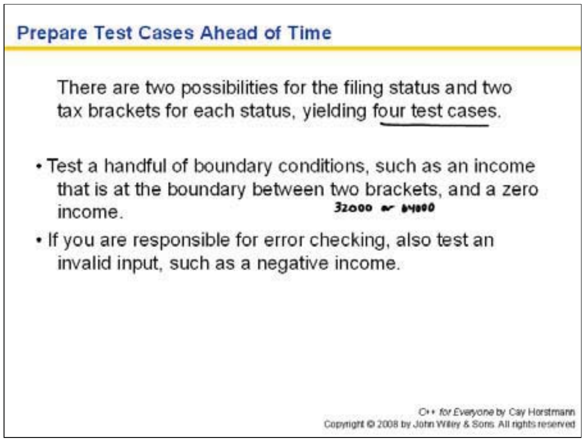
Prepare Test Cases Ahead of Time

If the program correctly computes one or two tax amounts in a given bracket, then we have a good reason to believe that all amounts will be correct.

You should also test on the *boundary conditions*, at the endpoints of each bracket

this tests the < vs. <= situations.

C++ *for Everyone* by Cay Horstmann

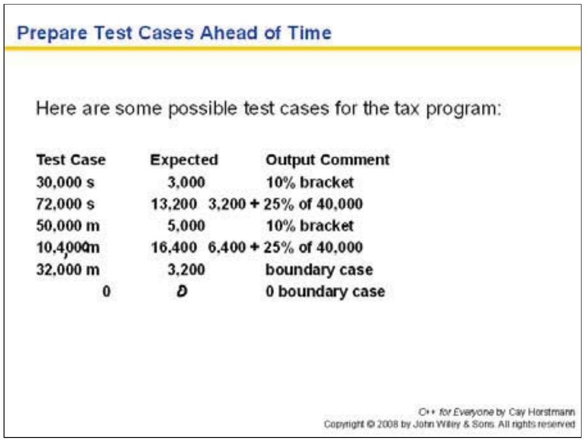


Prepare Test Cases Ahead of Time

There are two possibilities for the filing status and two tax brackets for each status, yielding four test cases.

- Test a handful of boundary conditions, such as an income that is at the boundary between two brackets, and a zero income. 32000 or 64000
- If you are responsible for error checking, also test an invalid input, such as a negative income.

C++ *for Everyone* by Cay Horstmann

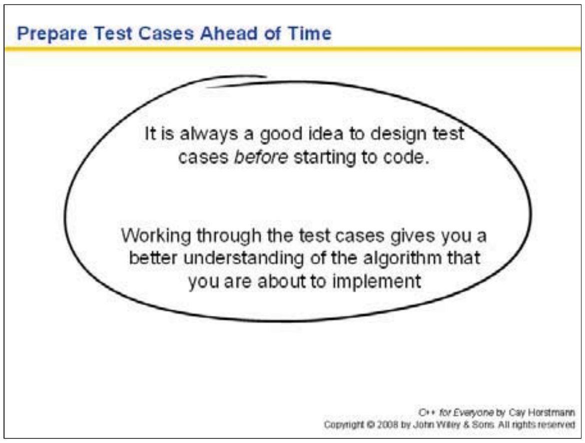


Prepare Test Cases Ahead of Time

Here are some possible test cases for the tax program:

Test Case	Expected	Output Comment
30,000 s	3,000	10% bracket
72,000 s	13,200	3,200 + 25% of 40,000
50,000 m	5,000	10% bracket
10,4000m	16,400	6,400 + 25% of 40,000
32,000 m	3,200	boundary case
0	0	0 boundary case

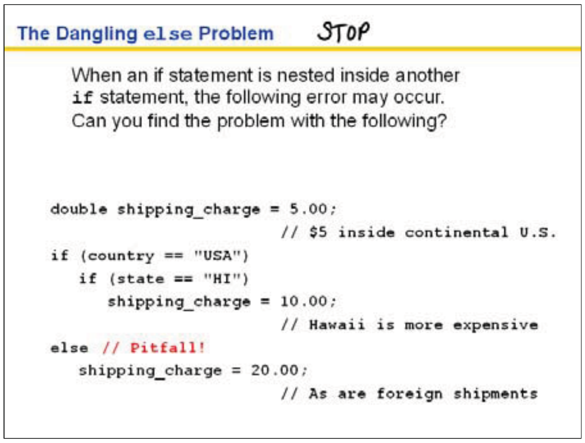
C++ *for Everyone* by Cay Horstmann



Prepare Test Cases Ahead of Time

It is always a good idea to design test cases *before* starting to code.

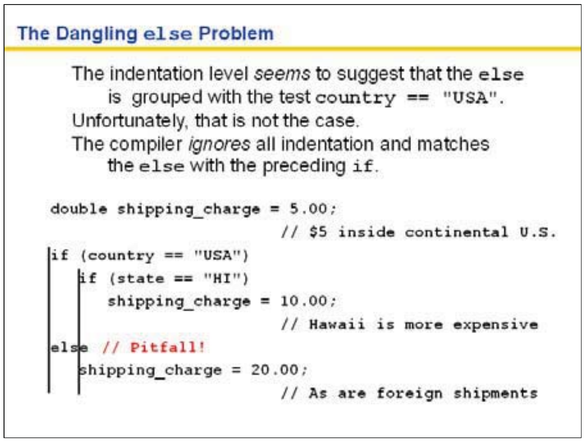
Working through the test cases gives you a better understanding of the algorithm that you are about to implement



The Dangling `else` Problem STOP

When an if statement is nested inside another **`if`** statement, the following error may occur.
Can you find the problem with the following?

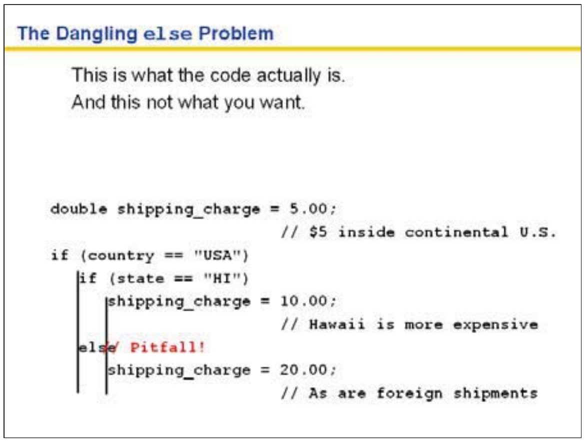
```
double shipping_charge = 5.00;
                                // $5 inside continental U.S.
if (country == "USA")
   if (state == "HI")
      shipping_charge = 10.00;
                                // Hawaii is more expensive
else // Pitfall!
   shipping_charge = 20.00;
                                // As are foreign shipments
```



The Dangling `else` Problem

The indentation level *seems* to suggest that the `else` is grouped with the test `country == "USA"`.
Unfortunately, that is not the case.
The compiler *ignores* all indentation and matches the `else` with the preceding `if`.

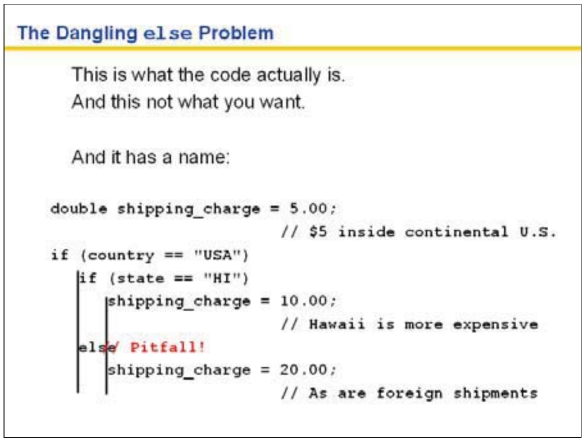
```
double shipping_charge = 5.00;
                                // $5 inside continental U.S.
if (country == "USA")
   if (state == "HI")
      shipping_charge = 10.00;
                                // Hawaii is more expensive
else // Pitfall!
   shipping_charge = 20.00;
                                // As are foreign shipments
```



The Dangling `else` Problem

This is what the code actually is.
And this not what you want.

```
double shipping_charge = 5.00;
                                // $5 inside continental U.S.
if (country == "USA")
   if (state == "HI")
      shipping_charge = 10.00;
                                // Hawaii is more expensive
   else/ Pitfall!
      shipping_charge = 20.00;
                                // As are foreign shipments
```

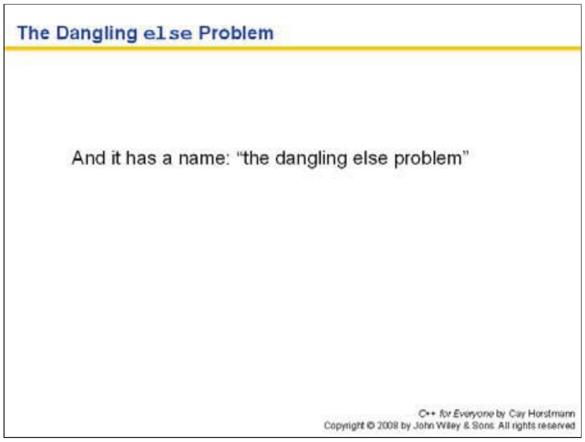


The Dangling `else` Problem

This is what the code actually is.
And this not what you want.

And it has a name:

```
double shipping_charge = 5.00;
                                // $5 inside continental U.S.
if (country == "USA")
   if (state == "HI")
      shipping_charge = 10.00;
                                // Hawaii is more expensive
   else/ Pitfall!
      shipping_charge = 20.00;
                                // As are foreign shipments
```



The Dangling `else` Problem

And it has a name: “the dangling else problem”