CS111

Introduction to Computing Science

Previously

- If-then-else decisions.
- Multiple If-then-else decisions.
- Switches.

Nested Branches

Your Problem

If have more than 10 dollars, consider going to village six. When you arrive there if movie “Idiot Monkey” is showing, watch that movie else watch “The return of cockroaches”. Otherwise not enough money, buy a DVD.

Nested Branches

Your pseudo code

```
If (money greater than $10)
        Go to village six
        If (movie == "Idiot monkey")
                Watch that movie
        Else
                Watch "The return of cockroaches"
Else
        Buy a DVD.
```

Nested Branches

How to make decisions

- that influence other decisions,
- that influence other decisions,
- that influence other decisions,
- ….

Nested Branches – Taxes

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

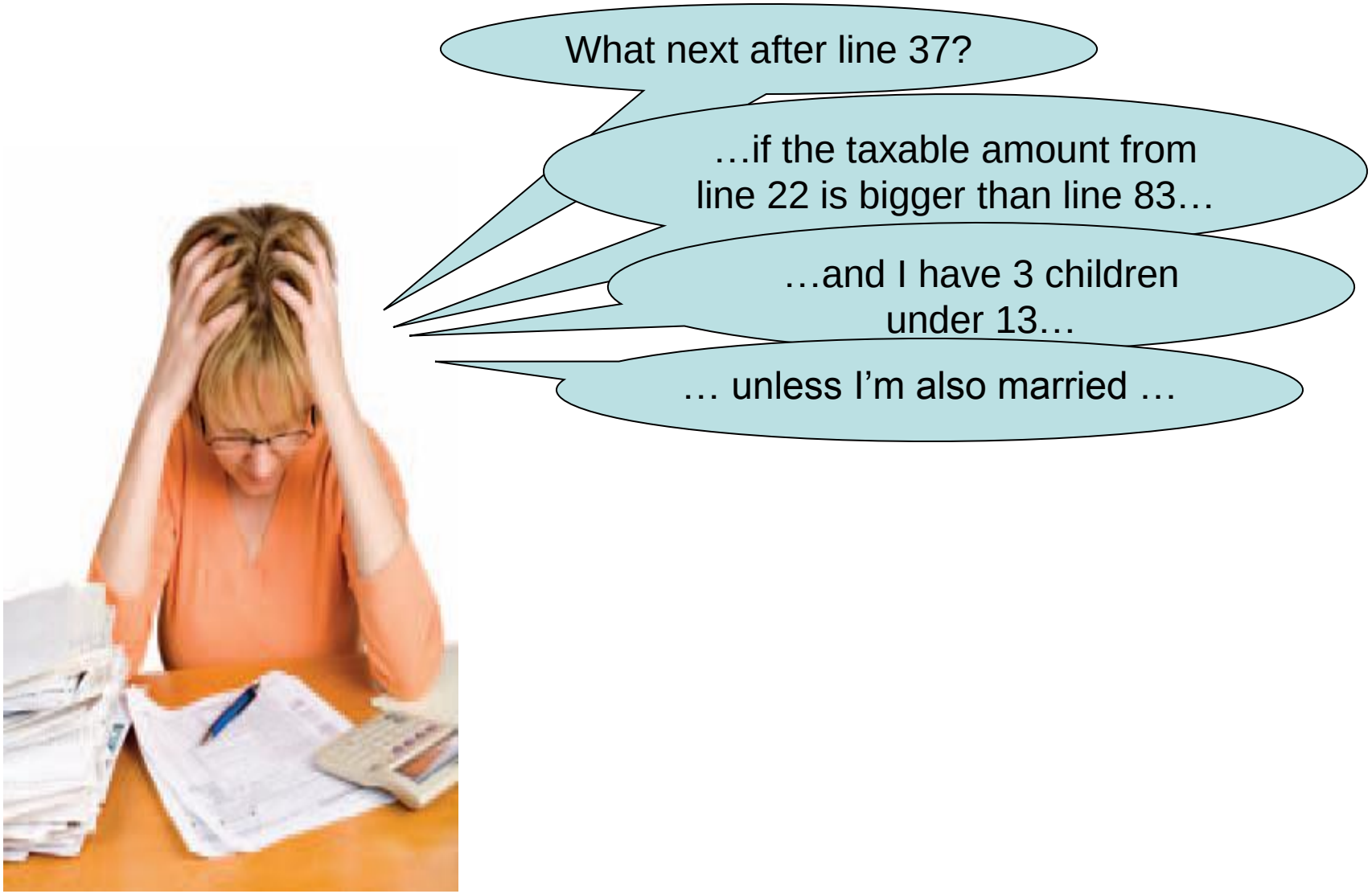
What next after line 37?

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

… and I have 3 children under 13 …

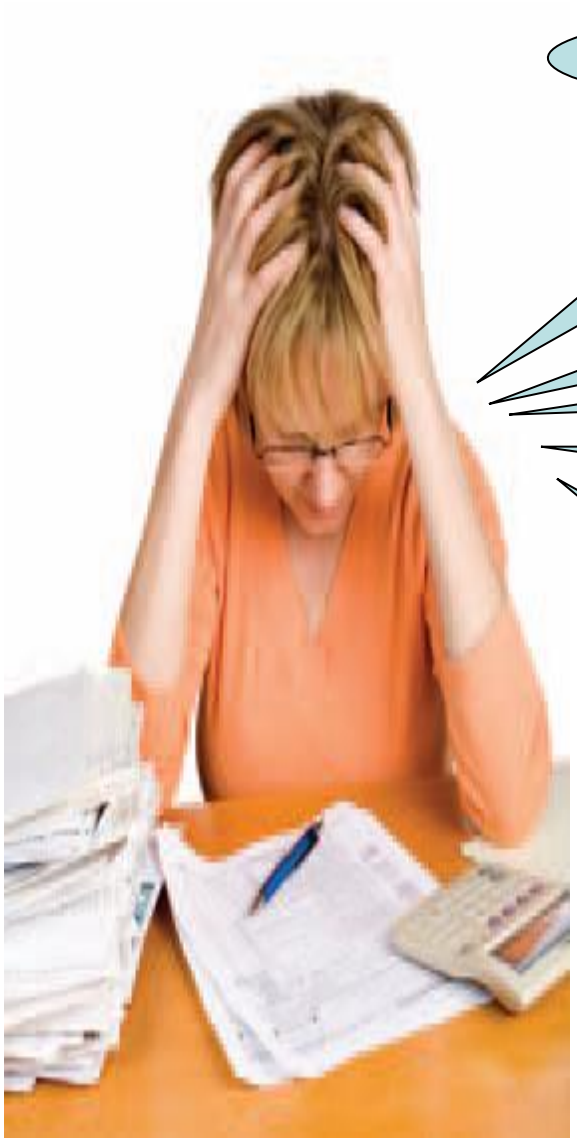
Taxes…

Nested Branches – Taxes



Taxes…

Nested Branches – Taxes



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?!

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.

Before we write code

First, as always, we analyze the problem.

Nested Branches – Taxes

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

The US tax office, the I.R.S. has done this for us.

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

Nested Branches – Taxes

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

Nested Branches – Taxes

ARGHHHH!!!!

Nested Branches – Taxes

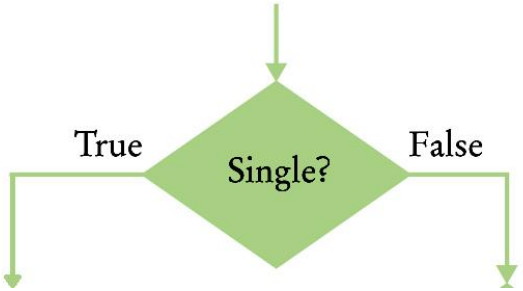
The key point is that there are two levels of decision making.

- Marital status
- Income

Really, only two (at this level).

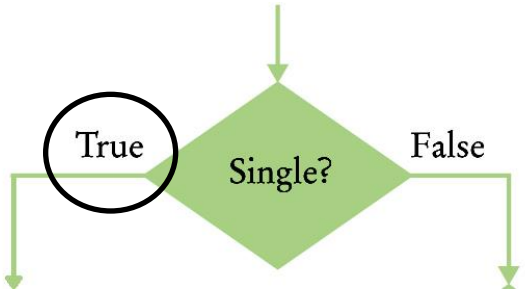
Nested Branches – Taxes

First, you must branch on the marital status.



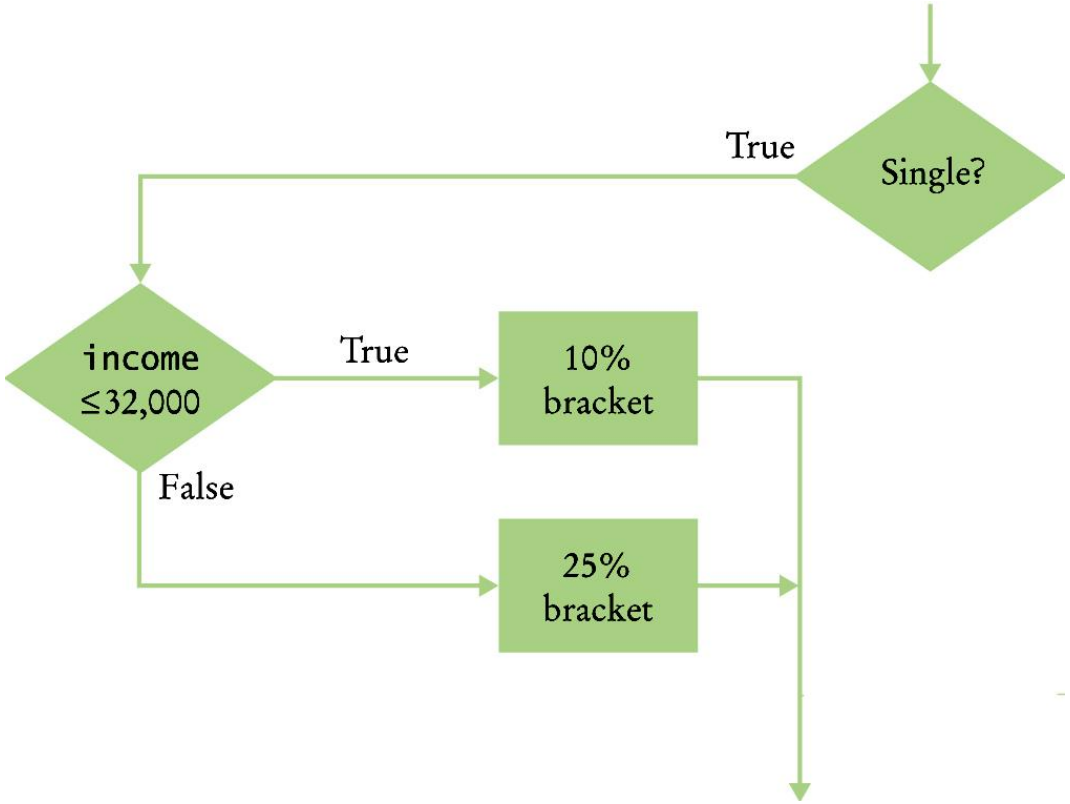
Nested Branches – Taxes

Then you must have another branch on income level.



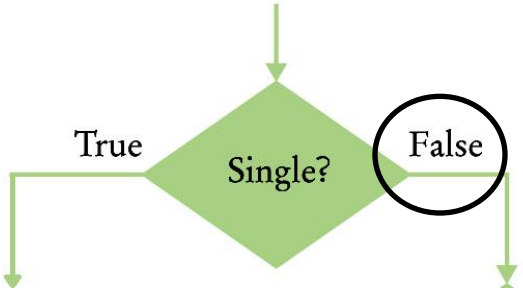
Nested Branches – Taxes

One *nested if-statement* for the singles.
Depending on their income.



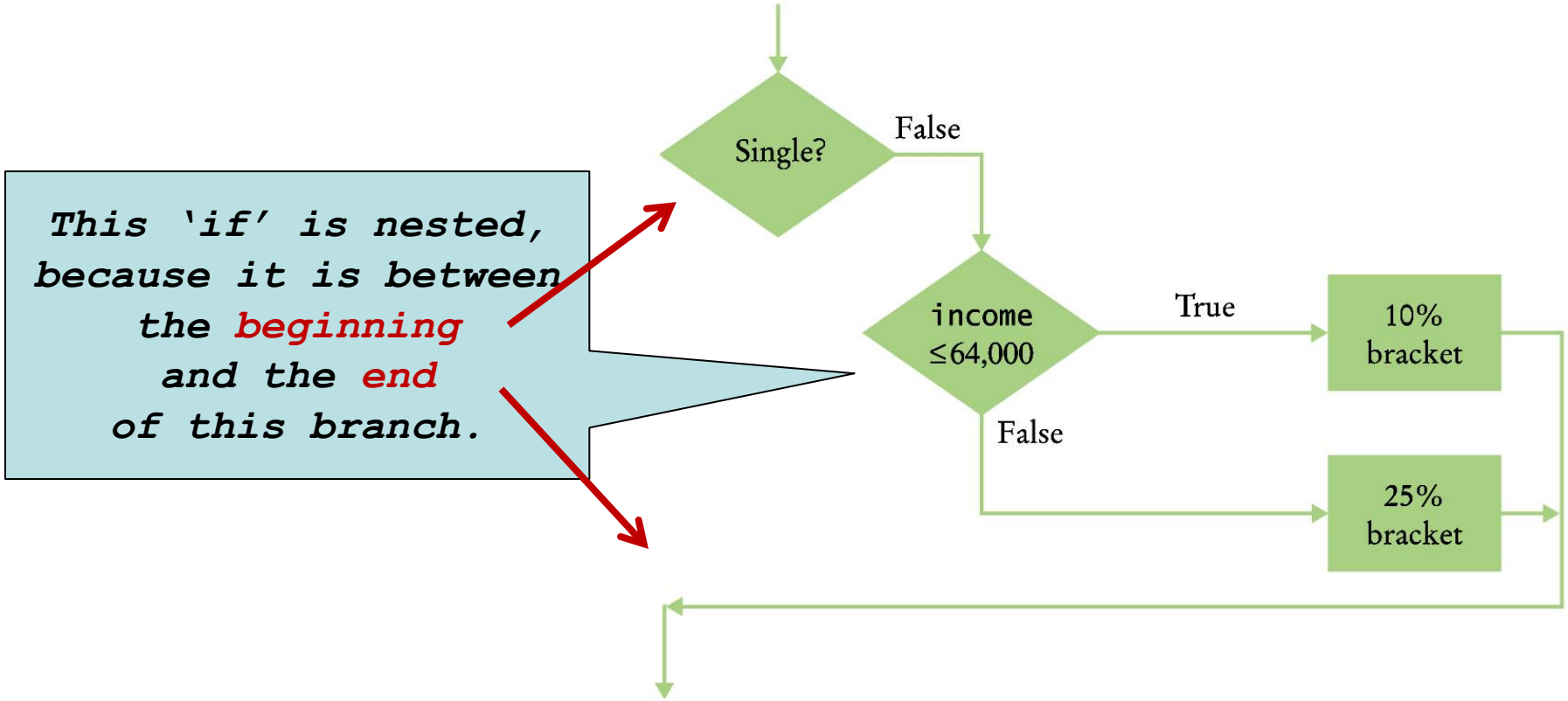
Nested Branches – Taxes

One *nested if-statement* for the married couples.
Depending on their income.



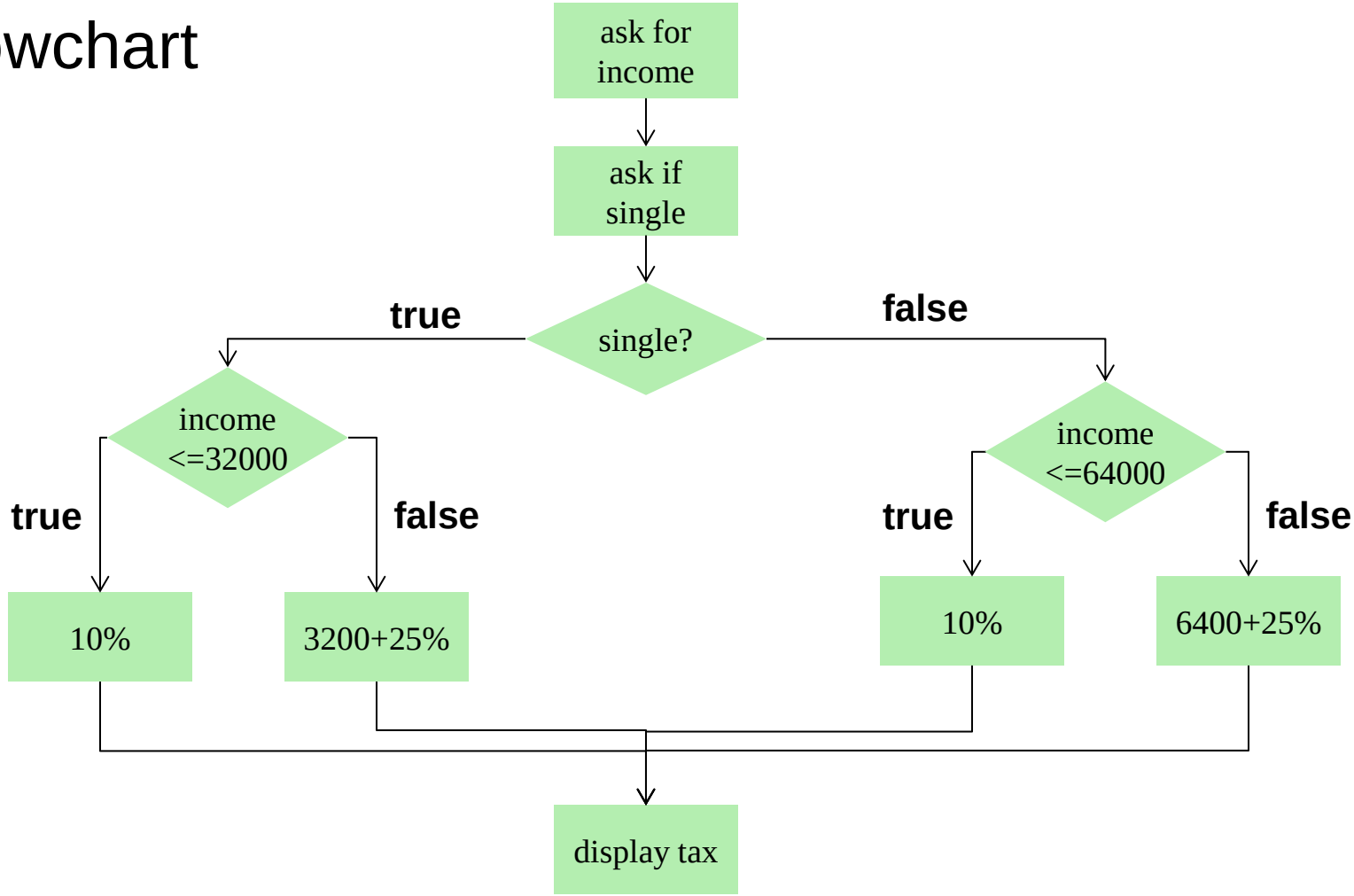
Nested Branches – Taxes

One *nested if-statement* for the married couples. Depending on their income.



Nested Branches – Taxes

Flowchart



Nested Branches – Taxes

Pseudo code

```
Ask for income
Ask for marital status

If single
        if income less than 32000
                use 10%  tax
        else
                use 3200 + 25% marginal tax
else
        if income less than 64000
                use 10% tax
        else
                use 6400 + 25% marginal tax

Display tax
```

Nested Branches – Taxes

```
int main()
{
   const double RATE1 = 0.10;
   const double RATE2 = 0.25;
   const double SINGLE_LIMIT = 32000;
   const double MARRIED_LIMIT = 64000;
   const double SINGLE_BASE = 3200;
   const double MARRIED_BASE = 6400;

   double tax = 0;
   double income;
   string marital_status;

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

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

First, we ask for income and marital status.

(to be continued)

Nested Branches – Taxes

```
if (marital_status == "s")
   {  //the true-branch (single)
      if (income <= SINGLE_LIMIT)
      {
         tax = RATE1 * income;
      }
      else
      {
         tax = SINGLE_BASE + RATE2 * (income - SINGLE_LIMIT);
      }
   }
else
```

This 'if' is nested, because it is between the beginning and the end of this branch.

(to be continued)

Nested Branches – Taxes

```
  {  //the false-branch (married - not single)
      if (income <= MARRIED_LIMIT)
      {
         tax = RATE1 * income;
      }
      else
      {
         tax = MARRIED_BASE + RATE2 * (income - MARRIED_LIMIT);
      }
   }

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

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

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

The Dangling `else` Problem

When an if statement is nested inside another if statement, the following error may occur.

Can you find the problem with the following?

```
double entrance_fee = 5.00; // Movies are usually $5

if ( part_of_week == "weekday")
   if ( format == "3D")
    entrance_fee = 7.00;  // 3D movies are $7 during the week
else
   entrance_fee = 9.00;   // All movies are $9 in the weekend
```

The Dangling `else` Problem

- Indentation level seems to suggest that the **`else`** belongs to the test **`part_of_week == "weekday"`**
- Unfortunately, that is not the case.
- The compiler ignores all indentation and matches the else with the preceding if **`if ( format == "3D")`**

```
double entrance_fee = 5.00; // Movies are usually $5

if ( part_of_week == "weekday")
   if ( format == "3D")
    entrance_fee = 7.00;  // 3D movies are $7 during the week
else
   entrance_fee = 9.00;   // All movies are $9 in the weekend
```

The Dangling `else` Problem

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

```
double entrance_fee = 5.00; // Movies are usually $5

if ( part_of_week == "weekday")
   if ( format == "3D")
    entrance_fee = 7.00;  // 3D movies are $7 during the week
   else
    entrance_fee = 9.00;   // All movies are $9 in the weekend
```

The Dangling `else` Problem

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

This problem has a name: "*the dangling else problem*"

```
double entrance_fee = 5.00; // Movies are usually $5

if ( part_of_week == "weekday")
   if ( format == "3D")
    entrance_fee = 7.00;  // 3D movies are $7 during the week
   else
    entrance_fee = 9.00;   // All movies are $9 in the weekend
```

The Dangling `else` Problem

There a solution to the dangling else problem.

Use braces.

```
double entrance_fee = 5.00; // Movies are usually $5

if ( part_of_week == "weekday"){
   if ( format == "3D"){
    entrance_fee = 7.00;  // 3D movies are $7 during the week
   }
}
else
{
    entrance_fee = 9.00;   // All movies are $9 in the weekend
}
```

Review and Testing

- Previously we told you how to write code.
- Complicated code.
- We told you how to find compile time errors.
- And how to fix them.
- We told you to look for warning messages.
- And take them serious.
- But how do you make sure your code is correct?

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 pseudo code or C++ code.

- This is a paper exercise.
- You'll need paper.
- Lot's of paper.

Hand-Tracing

You need to keep track of

- Where in the program you are
- The values of all variables

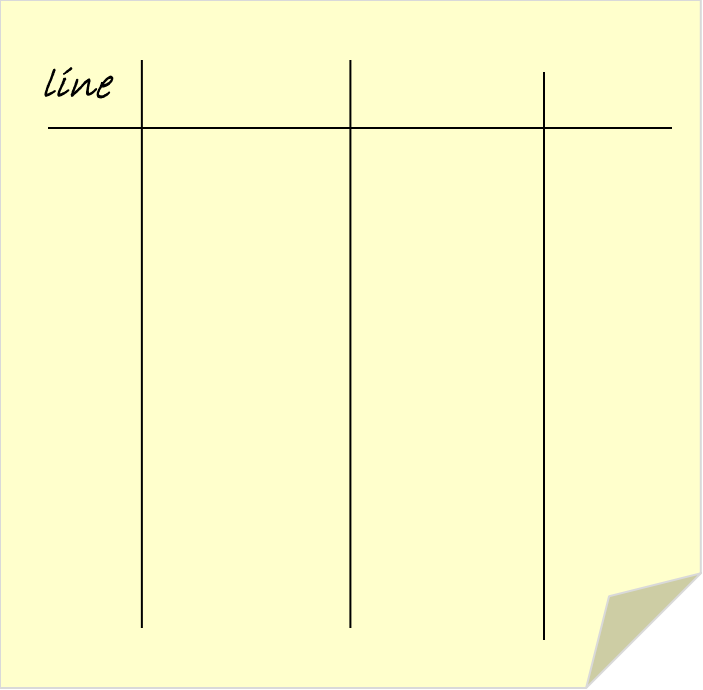
After each step you update

- Where in the program you are
- The values of all variables

Hand-Tracing

Let's do this with the tax program.

- We need some paper.
- A table with one column for the line number.



Nested Branches – Taxes

Then we add lime numbers to all non-empty lines.

```
int main()
{
      const double RATE1 = 0.10;
      const double RATE2 = 0.25;
      const double SINGLE_LIMIT = 32000;
      const double MARRIED_LIMIT = 64000;
      const double SINGLE_BASE = 3200;
      const double MARRIED_BASE = 6400;

      double tax = 0;
      double income = 0;
      string marital_status;

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

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
1   int main()
    {
2       const double RATE1 = 0.10;
3       const double RATE2 = 0.25;
4       const double SINGLE_LIMIT = 32000;
5       const double MARRIED_LIMIT = 64000;
6       const double SINGLE_BASE = 3200;
7       const double MARRIED_BASE = 6400;
…
```

line			

- Constants aren't "changes".
- They were created and initialized earlier so we don't write them in our trace.
- But we make a note.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
7       const double MARRIED_BASE = 6400;

8       double tax = 0;
9       double income = 0;
10      string marital_status;

…
```

line	tax		

- For each variable declaration we introduce a new column.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
7       const double MARRIED_BASE = 6400;

8       double tax = 0;
9       double income = 0;
10      string marital_status;

…
```

line	tax		
8	0		

- For each variable declaration we introduce a new column.
- If a variable is initialized, we write down the value.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
7       const double MARRIED_BASE = 6400;

8       double tax = 0;
9       double income = 0;
10      string marital_status;

…
```

line	tax		
8	0		
9			

- For each variable declaration we introduce a new column.
- If a variable is initialized, we write down the value.
- And we proceed with the next line.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
7       const double MARRIED_BASE = 6400;

8       double tax = 0;
9       double income = 0;
10      string marital_status;

…
```

line	tax	income	
8	0		
9	"	0	

- For each variable declaration we introduce a new column.
- If a variable is initialized, we write down the value.
- And we proceed with the next line.
- For values that don't change we mark "

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
7       const double MARRIED_BASE = 6400;

8       double tax = 0;
9       double income = 0;
10      string marital_status;

…
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	

- For each variable declaration we introduce a new column.
- If a variable is initialized, we write down the value.
- And we proceed with the next line.
- For values that don't change we mark "

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
11      cout << "Please enter your income: ";
12      cin >> income;

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

- For lines that just produce output nothing changes.

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
11      cout << "Please enter your income: ";
12      cin >> income;

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

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	

- For lines that just produce output nothing changes.
- For lines that ask for an input we select a value.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
11      cout << "Please enter your income: ";
12      cin >> income;

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

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	

- For lines that just produce output nothing changes.
- For lines that ask for an input we select a value.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
…
11      cout << "Please enter your income: ";
12      cin >> income;

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

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'

- For lines that just produce output nothing changes.
- For lines that ask for an input we select a value.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
15   if (marital_status == "s")
     {  //the true-branch (single)
17       if (income <= SINGLE_LIMIT)
         {
18           tax = RATE1 * income;
         }
19       else
         {
20           tax = SINGLE_BASE + RATE2 * (i
         }
     }
21   else
…
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"

- Every *if-condition* gets evaluated.
- If `true` continue with the next line, otherwise with the first line of the `else`.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
15   if (marital_status == "s")
     {  //the true-branch (single)
17        if (income <= SINGLE_LIMIT)
          {
18             tax = RATE1 * income;
          }
19        else
          {
20             tax = SINGLE_BASE + RATE2 * (i
          }
     }
21   else
…
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"
21	"	"	"

- Condition **`marital_status == "s"`** is false,
- Hence, continue with line 21

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
21      else
        {   //the false-branch (married - not
22          if (income <= MARRIED_LIMIT)
            {
23              tax = RATE1 * income;
            }
24              else
            {
25              tax = MARRIED_BASE + RATE2 *
            }
        }
..
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"
21	"	"	"
22	"	"	"

- Condition **`income <= MARRIED_LIMIT`** is true,
- Hence, continue with line 23

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
21      else
        {   //the false-branch (married - not
22          if (income <= MARRIED_LIMIT)
            {
23              tax = RATE1 * income;
            }
24              else
            {
25              tax = MARRIED_BASE + RATE2 *
            }
        }
..
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"
21	"	"	"
22	"	"	"
23	1000	"	"

- Condition **`income <= MARRIED_LIMIT`** is true,
- Hence, continue with line 23

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
21      else
        {   //the false-branch (married - not
22          if (income <= MARRIED_LIMIT)
            {
23              tax = RATE1 * income;
            }
24              else
            {
25              tax = MARRIED_BASE + RATE2 *
            }
        }
..
```

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"
21	"	"	"
22	"	"	"
23	1000	"	"

- At the end of a block continue with the next line after the whole `if`.

Hand-Tracing

RATE1=0.10, RATE2=0.25
SINGLE_LIMIT=32000,
MARRIED_LIMIT = 64000
SINGLE_BASE =3200
MARRIED_BASE =6400

```
26      cout << "The tax is $" << tax << endl;
27      return 0;
   }
..
```

- Output to screen does not change the values of variables.
- Continue until the end (**`return 0`**)

line	tax	income	marital status
8	0		
9	"	0	
10	"	"	
11	"	"	
12	"	10000	
13	"	"	
14	"	"	'm'
15	"	"	"
21	"	"	"
22	"	"	"
23	1000	"	"
26	"	"	"
27	"	"	"

The end.

Prepare Test Cases

Test cases

- Consider how to test the tax computation program.
- 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.

Prepare Test Cases

Test cases

- A *test case* specifies the input to the program and the expected output.
- The are usually multiple test cases, to cover different parts of the code.
- How do you choose a good test case?

Prepare Test Cases

Test cases

- A good practice is to select test cases that test boundary condition.
- Testing *boundary condition* means to select values such that condition are either just true, or just false.
- Example `if(income<=SINGLE_LIMIT)`

  SINGLE_LIMIT is 32000

  - Use a value of 32000 for `income`, for one test,
  - Use a value of 32001 for `income`, for another test

Prepare Test Cases

Test cases

- Test also very small values, and very large values.
- Test a value of 0.
- Test “invalid” input, like negative income.

- Similarly, there are two possible answers for the marital status:
  - married (“m”)
  - single (“s”)
- Use each of the values in different test cases.
- Use also ‘invalid’ input such as “x”.

Prepare Test Cases

Here are some possible test cases for the tax program:

Test Case	Expected Output	Comment
32,000 s	3,200	10% bracket
32,001 s	3,200.25	3,200 + 25% of 1
64,000 m	6,400	10% bracket
64,100 m	6,425	6,400 + 25% of 100
100,000 m	15,400	6,400 + 25% of 36000
0 m	0	boundary case
10000 x	?	invalid input
….		

Prepare Test Cases

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.

Prepare Test Cases

USS Yorktown

- Testbed for the US Navy's Smart Ship program
- In 1997 a crew member entered accidentally zero into a database field
- Result 1: division by zero
- Result 2: network failure
- Result 3: propulsion system failure
- Result 4: ship “dead in the water”
  for about 3hrs.

Lesson: Test for 0s