

Topic 3

1. The `if` statement
2. Comparing numbers and strings
3. Multiple alternatives
4. Nested branches
5. Problem solving: flowcharts
6. Problem solving: test cases
7. Boolean variables and operators
8. Application: input validation
9. Chapter summary

Multiple Alternatives Need Multiple Nested if() Statements

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

In the case of the Richter Scale for earthquake magnitude, there are five branches:

one each for the four descriptions of damage, and a "default" fifth one for no destruction (not shown).

Flowchart for the Richter Scale Code

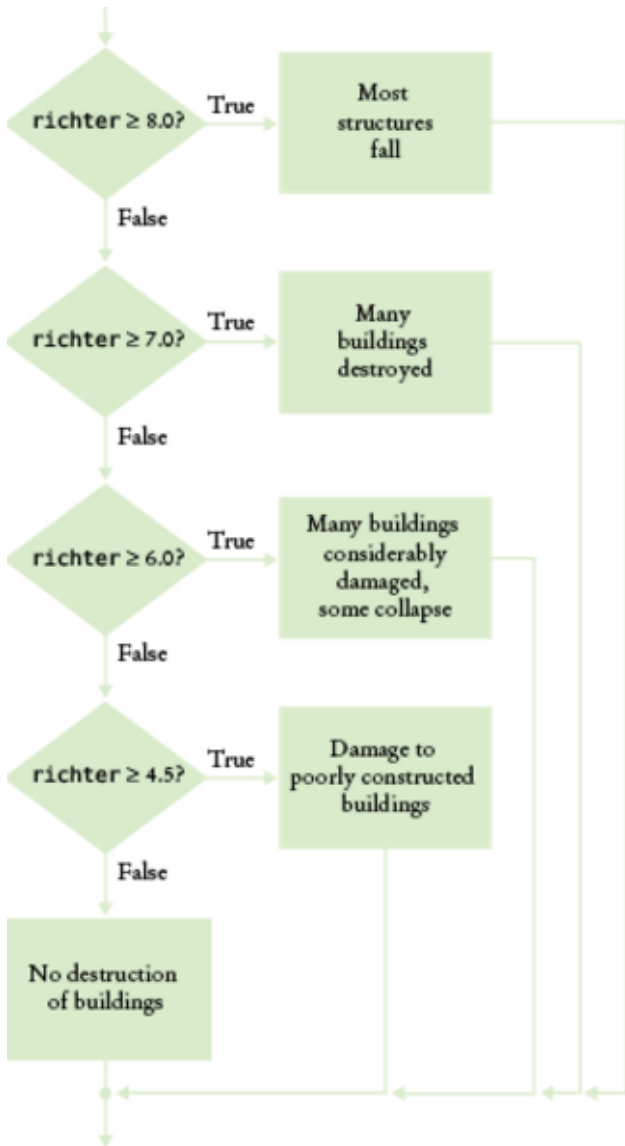


Figure 3 Multiple Alternatives

Multiple Alternatives (Richter Scale Code)

```
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";
}
```

. . .

Multiple Alternatives – 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";
}
```

Suppose the value of `richter` is 7.1. Because we tested small first with a `>=`, the first statement is (wrongly) printed.

The `switch` Statement vs. the `if` statement

- Below is a complicated `if()` statement to choose a text `string` to assign based on the value of an `int` variable:

```
int digit;
... //digit variable gets set here by some code
if (digit == 1) { digit_name = "one"; }
else if (digit == 2) { digit_name = "two"; }
else if (digit == 3) { digit_name = "three"; }
else if (digit == 4) { digit_name = "four"; }
else if (digit == 5) { digit_name = "five"; }
else if (digit == 6) { digit_name = "six"; }
else if (digit == 7) { digit_name = "seven"; }
else if (digit == 8) { digit_name = "eight"; }
else if (digit == 9) { digit_name = "nine"; }
else { digit_name = ""; }
```

The switch Statement

- The `switch` statement is an alternative to nested `if() else` statements. But `switch` is at least as awkward to code as nested `if() else`:

```
int digit; //switch can only test int and char types
... //digit variable gets set here by some code
switch(digit)
{
    case 1: digit_name = "one"; break;
    case 2: digit_name = "two"; break;
    case 3: digit_name = "three"; break;
    case 4: digit_name = "four"; break;
    case 5: digit_name = "five"; break;
    case 6: digit_name = "six"; break;
    case 7: digit_name = "seven"; break;
    case 8: digit_name = "eight"; break;
    case 9: digit_name = "nine"; break;
    default: digit_name = ""; break; //taken if none of the above
}
```


break statements in the switch statement

- Every branch of the switch must be terminated by a **break** statement. And each branch must terminate with a semicolon.
- **break** tells the machine to skip down to the end of the `switch` statement, because a match was found.
- If the **break** is missing, execution falls through to the next branch, and so on, until finally a **break** or the end of the switch is reached.
- In practice, this fall-through behavior is rarely useful, and *it is a common cause of errors.*
- If you accidentally forget the **break** statement, your program compiles but executes unwanted code. Try it and see!

Topic 4

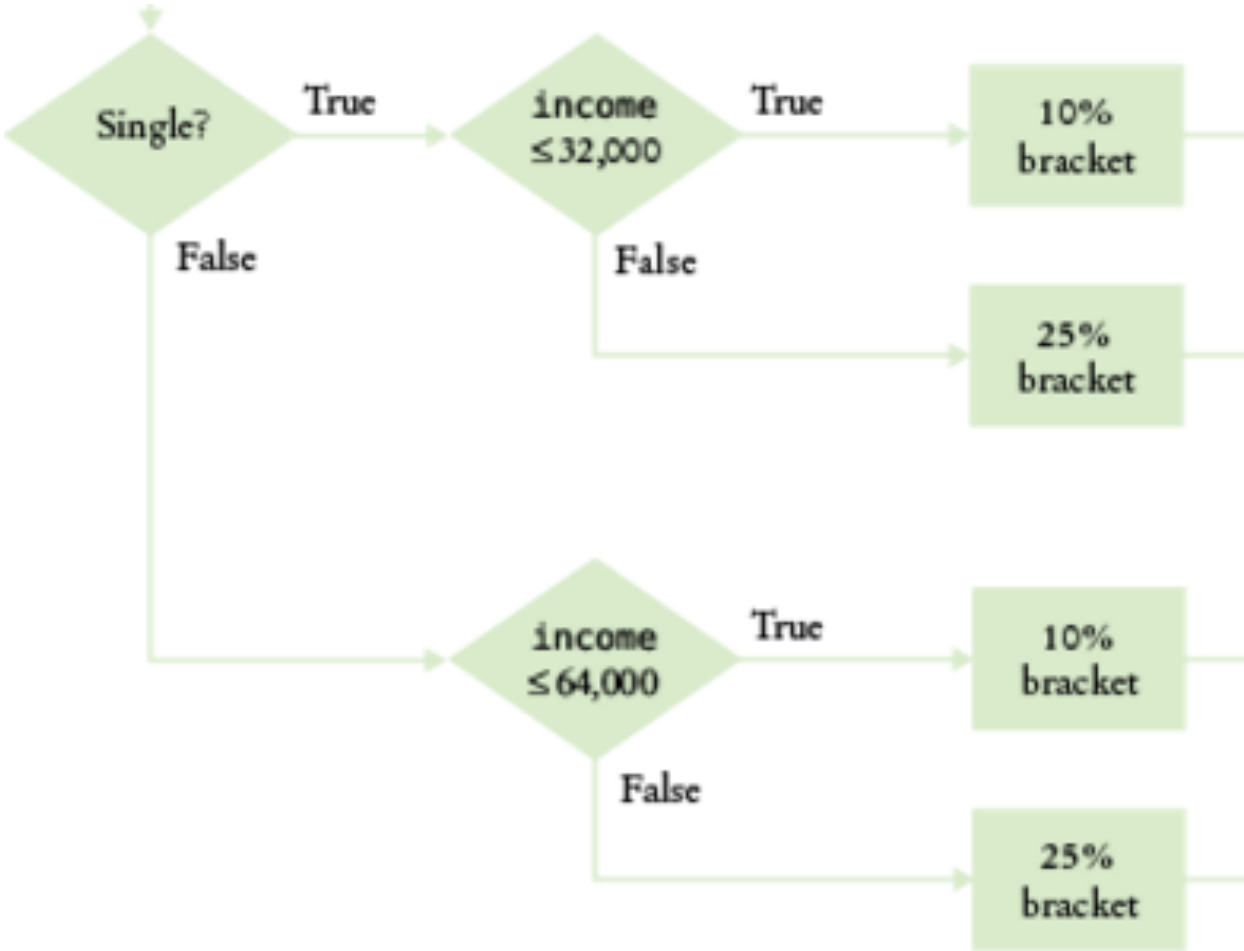
1. The `if` statement
2. Comparing numbers and strings
3. Multiple alternatives
4. Nested branches
5. Problem solving: flowcharts
6. Problem solving: test cases
7. Boolean variables and operators
8. Application: input validation
9. Chapter summary

Nested Branches – Taxes

- In the United States different tax rates are used depending on the taxpayer's marital status – single rates are higher.
- Married taxpayers add their income together and pay taxes on the total. See the IRS table below from a recent year:

Table 4 Federal Tax Rate Schedule		
If your status is Single and if the taxable income is	the tax is	of the amount over
at most \$32,000	10%	\$0
over \$32,000	\$3,200 + 25%	\$32,000
If your status is Married and if the taxable income is	the tax is	of the amount over
at most \$64,000	10%	\$0
over \$64,000	\$6,400 + 25%	\$64,000

Flowchart for Tax Table Decisions



Nested Branches – Taxes – Complete Code part 1

```
#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 (Code part 2)

```
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
```

Nested Branches – Taxes (Code part 3)

```
{
    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;
}
```

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.

Hand-Tracing, How to

- 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 the Tax Program

```
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, Filling in the Trace Table

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

Assume user typed 80000.

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

The user typed m

tax1	tax2	income	marital status
0	0	80000	m

Hand-Tracing #2

```
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 //this branch is taken because marital_status != "s"
```

Hand-Tracing #3

tax1	tax2	income	marital status
0	0	80000	m

```
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 #4

```
else
{
    if (income <= 64000) //this branch is skipped, false
    {
        tax1 = RATE1 * income;
    }
    else //this branch is taken
    {
        tax1 = RATE1 * RATE1_MARRIED_LIMIT;
        tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT);
    }
}
double total_tax = tax1 + tax2;
```

Hand-Tracing #5

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

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