```
#include <cfruit>
#include <cwarfare>
```

Selection Statements

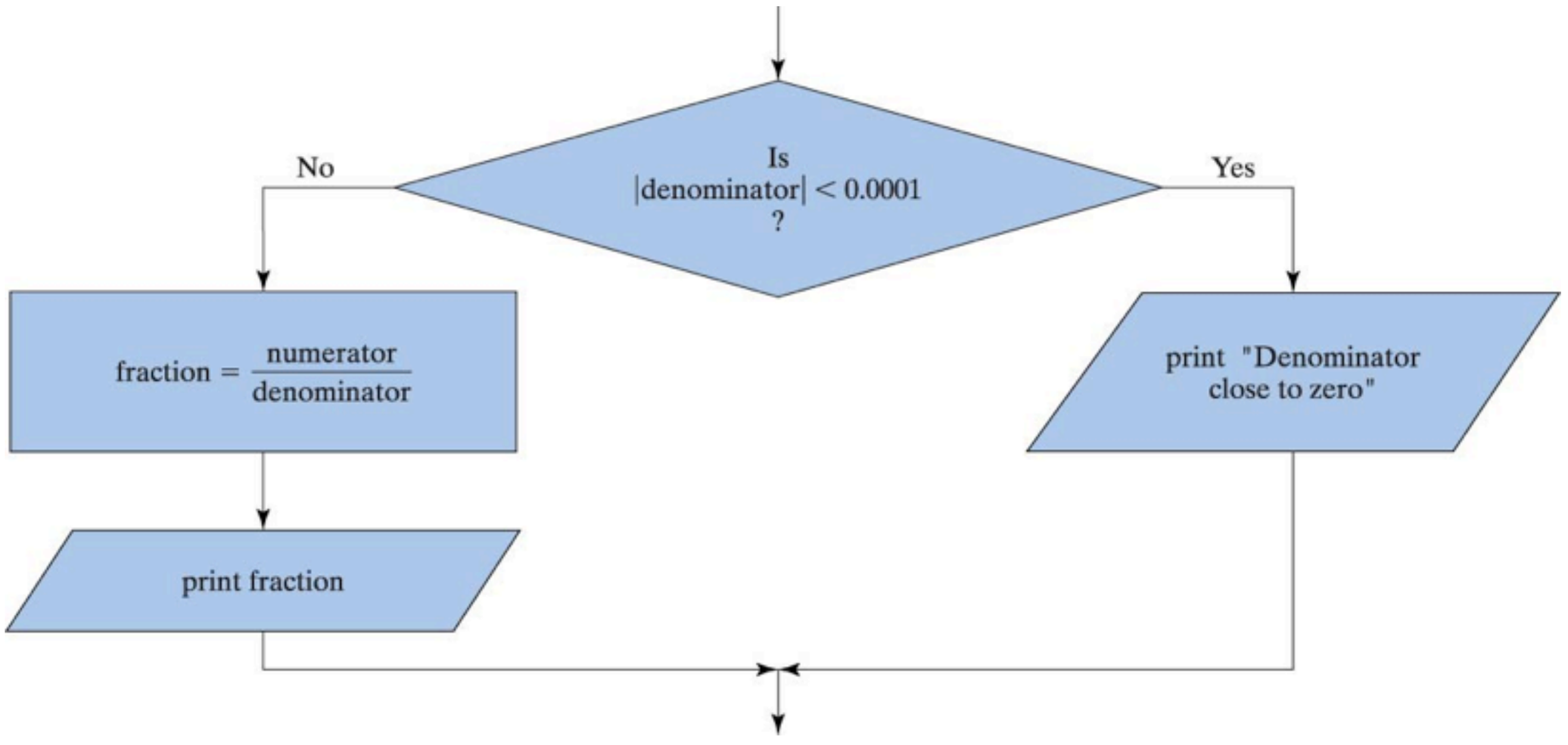
```
if (haveLemons) {
    makeLemonade();
} else {
    conquerEarth();
    buyLemons();
    makeLemonade();
}
```

Selection Statements

Selection statements *select* from one or more distinct control paths

- the decisions made are based on the result of a *condition* (Boolean expression)

A simple example:



“Blocks” of Code

Code can be grouped into distinct control paths, called “blocks”

- code is executed from top to bottom inside a code block
- code within a block should be indented one level further than code around it

Blocks of code are surrounded by curly braces:

```
{ /* a block of code */  }
```

The body of the `main()` function is a block:

```
int main() { // start of block
    cout << “Hello world!” << endl;
    return 0;
} // end of block
```

if Statements

General syntax:

```
if (condition) {
    // code to execute if condition evaluates to true
}
```

Notice the 3 components:

- the if keyword, followed by
- a condition (Boolean expression) inside of a pair of parentheses, and
- the block of code to execute if the condition is true

if Statements

Example:

```
// assume num is an int variable that gets its value from cin
if (num > 2 && num < 10) {
    cout << “num is between 2 and 10! Huzzah!” << endl;
}
```

If the condition `(num > 2 && num < 10)` evaluates as true:

- the block of code associated with the if statement will be executed, top to bottom
- `num is between 2 and 10! Huzzah!` will be printed to the console

If the condition is false:

- nothing will happen

if-else Statements

General syntax:

```
if (condition) {
    // code to execute if condition evaluates to true
} else {
    // code to execute if condition evaluates to false
}
```

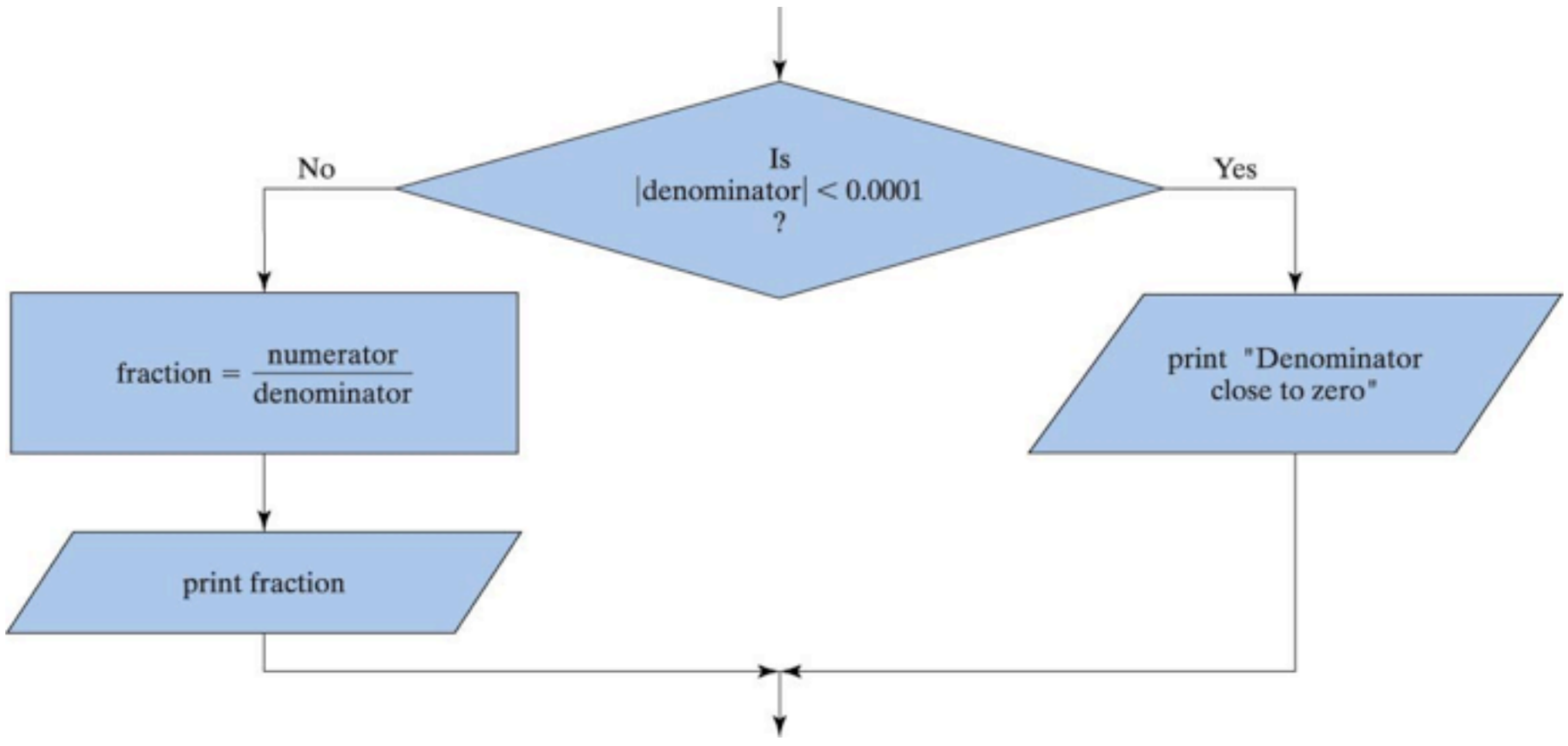
Simply an if statement, followed by:

- the else keyword, and
- the block of code to execute if the if statement's condition is false

if-else Statements

Example:

```
if (abs(denominator) < 0.0001) {
    cout << “Denominator close to zero” << endl;
} else {
    fraction = numerator / denominator;
    cout << fraction << endl;
}
```



if-else Statements

Another example:

```
// get a number from the user
cout << “Please enter an integer: ”;
int num;
cin >> num;

// check whether the number is odd or even
if (num % 2) {
    cout << “You entered an ODD number.” << endl;
} else {
    cout << “You entered an EVEN number.” << endl;
}
```

Nested Statements

Statements can be *nested* inside one another:

```
string surname = “Thomas”, prefix;
char gender = ‘M’;
bool isMarried = false; // a Boolean flag


if (gender == ‘M’) {
    prefix = “Mr.”;
} else {
    if (isMarried) {
        prefix = “Mrs.”;
    } else {
        prefix = “Ms.”;
    }
}
cout << “Hello, ” << prefix << “ ” << surname << “! You are a bum!” << endl;
```

nested if-else statement!

Nesting Gone Wild

===CENSORED===

Nesting Gone... Less Wild?

Sometimes we have a lot of different conditions...

```
if (grade >= 90) {
    cout << “A”;
} else {
    if (grade >= 80) {
        cout << “B”;
    } else {
        if (grade >= 70) {
            cout << “C”;
        } else {
            if (grade >= 60) {
                cout << “D”;
            } else {
                cout << “F”;
            }
        }
    }
}
```

That’s a lot of indenting and curly braces...

NO ES BUENO!

Nesting Gone... Less Wild?

There's a much cleaner way to do it:

```
if (grade >= 90) {
    cout << "A";
} else if (grade >= 80) {
    cout << "B";
} else if (grade >= 70) {
    cout << "C";
} else if (grade >= 60) {
    cout << "D";
} else {
    cout << "F";
}
```

Ahh... Much better! =)

When you have multiple dependent conditions, use this syntax

- it's much easier to read and to maintain

One of the many perils of C++

What is the user going to see if he runs this program?

```
// presumably the user's age...
int age = 10;
// check if the user is the same age as me
if (age = 23) {
    cout << "Hey, we're the same age!" << endl;
} else {
    cout << "Go away and be ashamed!" << endl;
}
```

The condition `(age = 23)` is always true!

- assignment operators evaluate to the value being assigned, and the number `23` is considered true
- only if age were `0` would the code associated with the `else` block be executed

Terse Selection Statements

Terse Selection Statements

If the block associated with an if or else clause has only one statement, the curly brackets { } can be omitted:

```
int x = 3, y = 0;


// terse if statement

if (x > 0)
    y++;


// terse if-else statement
if (x > 0)
    cout << “y is now greater than x” << endl;
else
    cout << “x is still larger than y” << endl;
```

Terse Selection Statements

An if-else statement is considered a single statement, so this works:

```
if (x > y)
    if (x - y > 100)
        cout << “x is much bigger than y” << endl;
    else
        cout << “x is slightly bigger than y” << endl;
else
    if (y - x > 100)
        cout << “y is much bigger than x” << endl;
    else
        cout << “y is slightly bigger than x” << endl;
```

single statement

single statement

Terse Selection Statements

An else clause always belongs to the if closest behind it:

```
if (x >= 0)
    if (y < x)
        cout << “y is less than x” << endl;
else
    cout << “x is less than zero” << endl;
```

Indentation doesn’t matter to the compiler, so you get this instead:

```
if (x >= 0) {
    if (y < x)
        cout << “y is now greater than x” << endl;
    else
        cout << “x is less than zero” << endl;
}
```

Terse Selection Statements

Terse statements are easy to screw up!

- when will x be changed?

```
if (x > 0)
    y = ++x + 1;
else if (y < x)
    x = y--;
else if (y > 0)
    y %= x * 2 + 1;
    x += 100;
```

- always! the last statement, even though it's indented, doesn't belong to the if-elseif statement
- only ONE statement can be linked to an if or else clause *if you don't use curly brackets*

Terse Selection Statements

Terse statements are easy to screw up!

- What happens if you add debugging statements?

```
if (x > 0)
    cout << “inside x > 0” << endl;
    y = ++x + 1;
else if (y < x)
    cout << “inside y < x” << endl;
    x = y--;
else if (y > 0)
    cout << “inside y > 0” << endl;
    y %= x * 2 + 1;
```

- now it won’t even compile because the else clauses are not linked to if statements!

The ternary operator

My favorite! Not that you care, but still...

The ternary operator

General syntax:

```
condition ? value_if_condition_is_true : value_if_condition_is_false;
```

Examples:

```
cout << (true ? “TRUE” : “FALSE”) << endl;  // prints “TRUE”

cout << (false ? “TRUE” : “FALSE”) << endl; // prints “FALSE”
```

Equivalent if-statement for first example:

```
if (true) {
    cout << “TRUE” << endl;
} else {
    cout << “FALSE” << endl;
}
```

Ternary operator example

Example (without using the ternary operator):

```
// assume num is an int variable holding user input
cout << “You have ” << num;
if (num == 1) {
    cout << “ penguin”;
} else {
    cout << “ penguins”;
}
cout << endl;
```

An example <u>with</u> the ternary operator:

```
cout << “You have ” << num << (num == 1 ? “ penguin” : “ penguins”) << endl;
```

```
(condition ? value_if_true : value_if_false)
```