Boolean Expressions

(based on a `true` story)

Fond memories of programs past...

Let's admit it:

- our first five programs have been stupid.
- did I say stupid? I meant "unintelligent", as in….
- REALLY, mind-blowingly (like "eating your own hand") stupid
- but it's not our fault (I say blame it on our parents)

A “Smart” Program Makes Decisions

Examples include:

- validating input (did the user enter an amino acid that has fewer than 0 atoms?)
- deciding which calculation to use (one for negative numbers, another for positive?)
- knowing to address the user as male or female… or transgender O_o
- repeating a block of code for each input item or for a set number of times

Some Disclaimers:

1. Programs are neither smart nor stupid.

2. You are a good and competent person.

3. You are likely to become less and less intelligent the more you listen to me. Sad, huh?

C++ Control Patterns

Sequence:

- a simple sequence of commands to be executed one after the other
- most of the code we've encountered is sequential

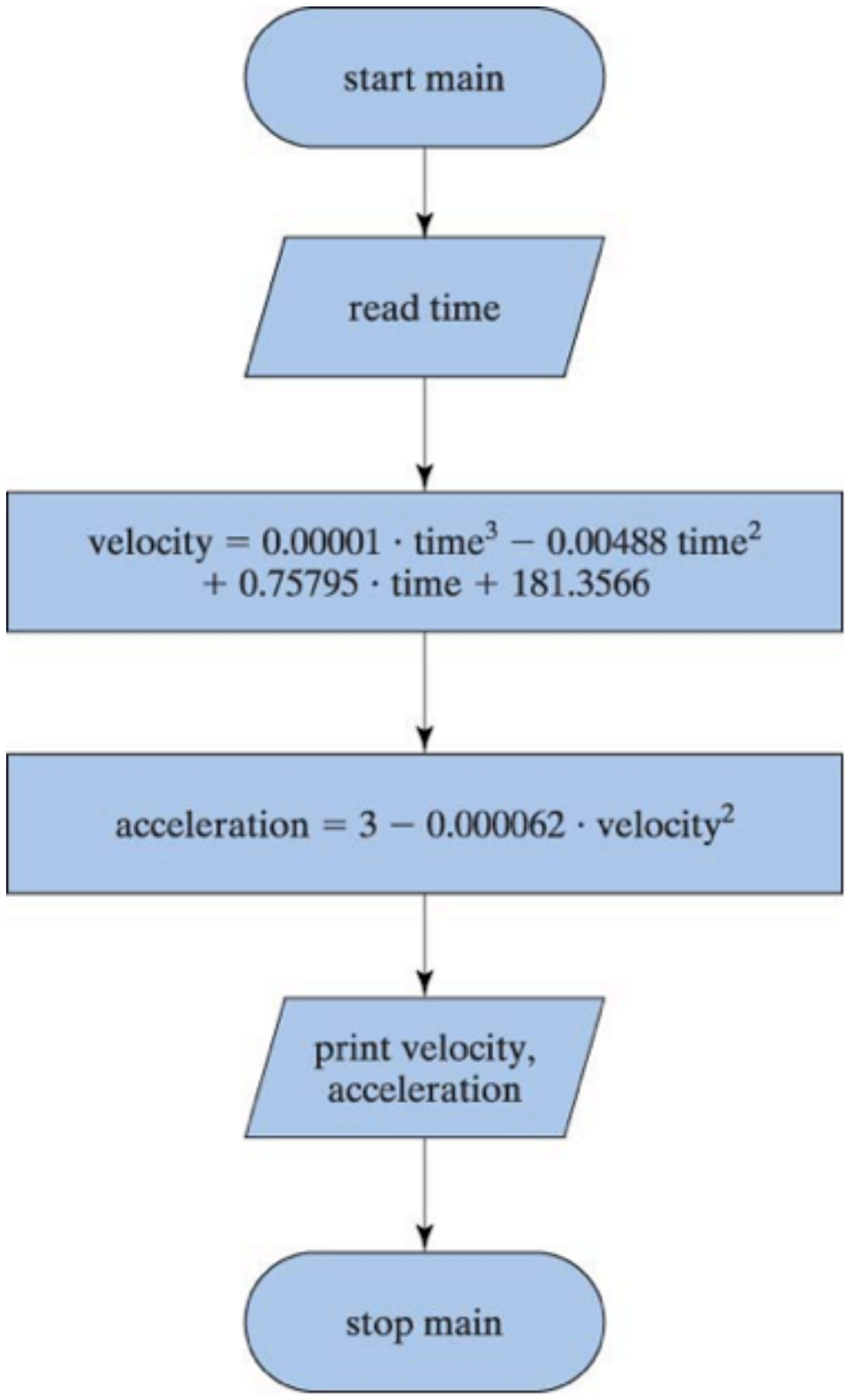
Selection:

- selectively executes a block of code depending on some condition
- can choose between multiple control paths

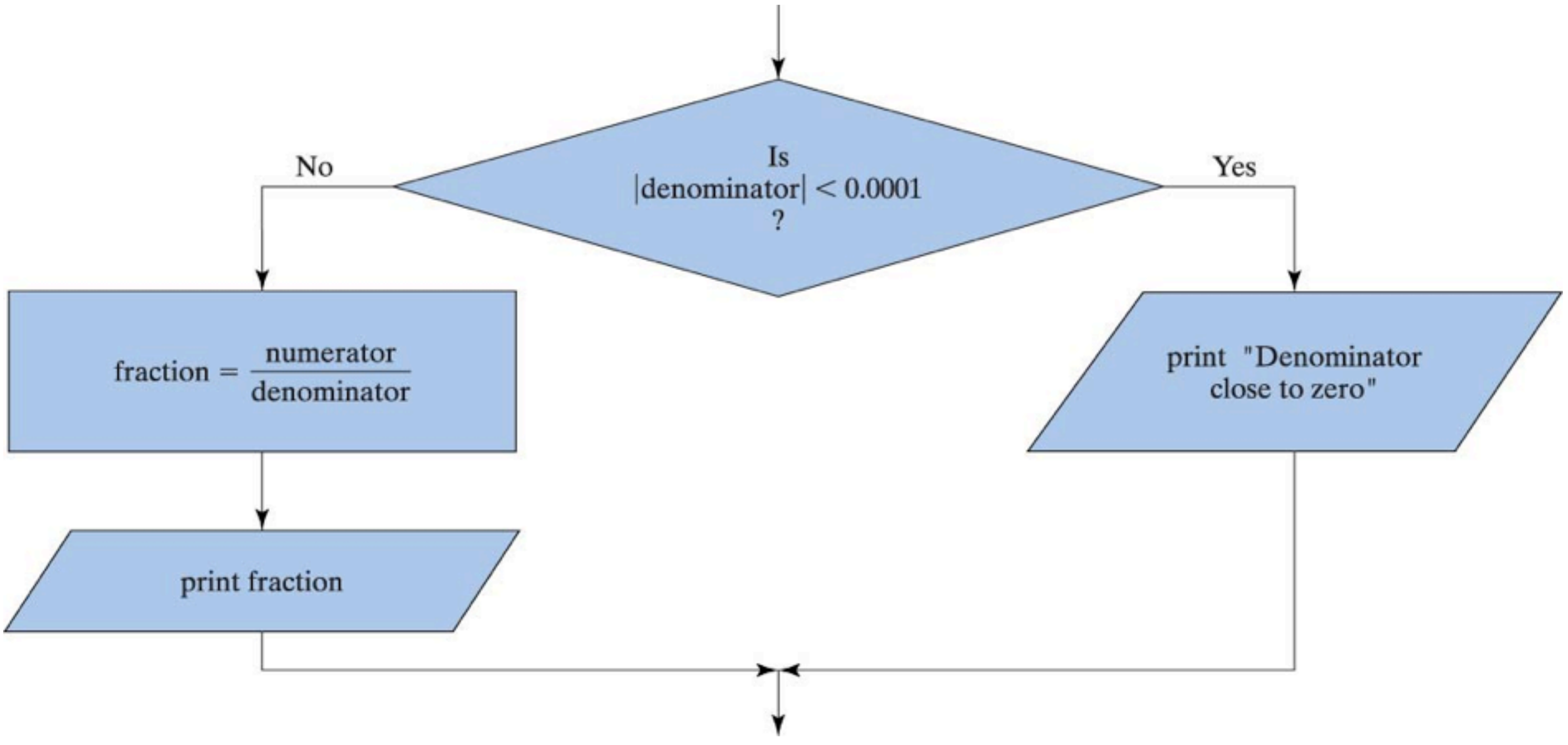
Repetition:

- repeats the same block of code until some condition is no longer true
- essentially a repeating *loop* of code

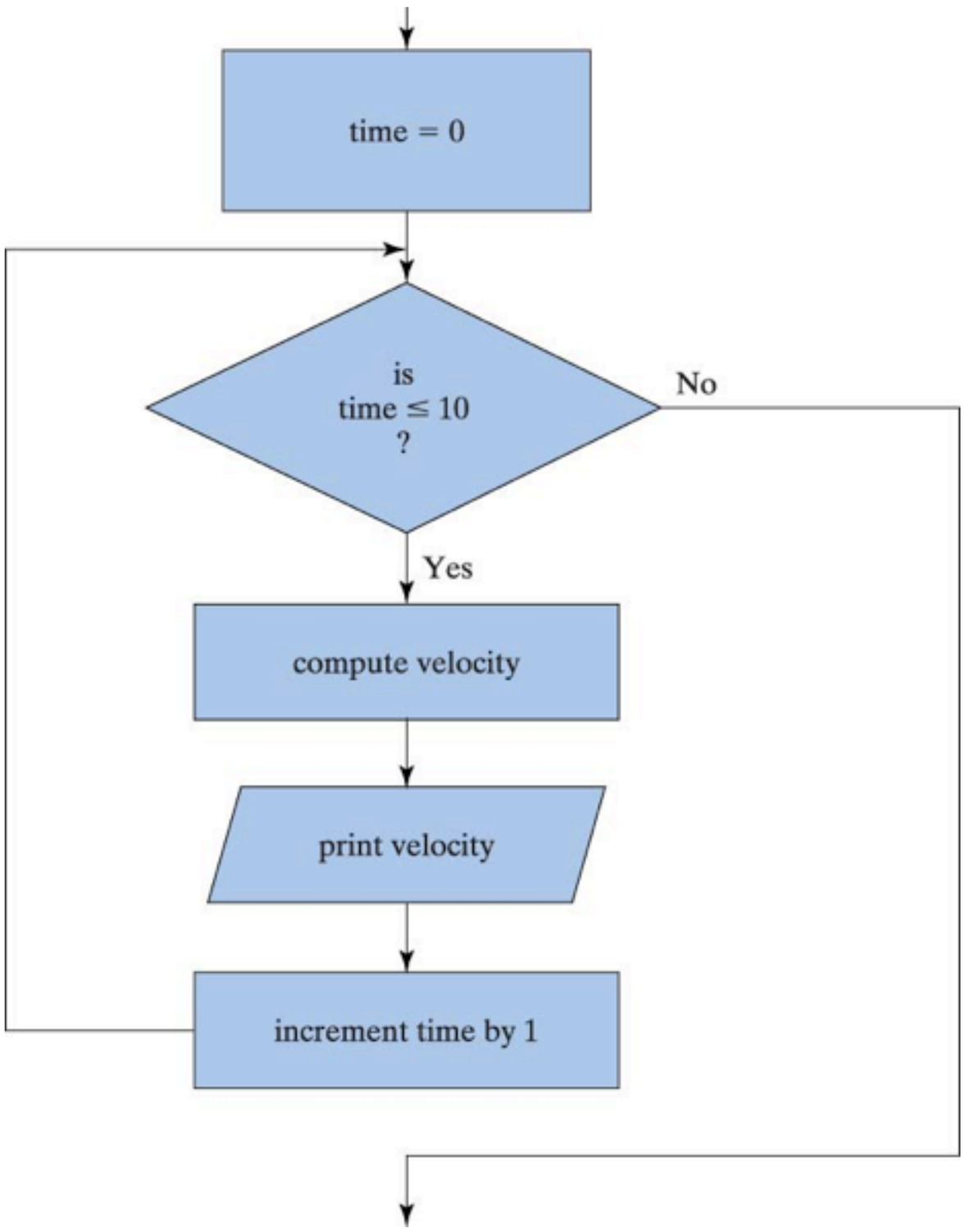
A Sequence


start main
read time
velocity = 0.00001 · time³ − 0.00488 time²
+ 0.75795 · time + 181.3566
acceleration = 3 − 0.000062 · velocity²
print velocity,
acceleration
stop main

A Selection


Is
|denominator| < 0.0001
?
No
Yes
fraction = numerator / denominator
print fraction
print "Denominator close to zero"

Some Repetition



Conditions

A condition is an expression that returns either true or false

Examples:

- is ‘x’ less than some number?
- are ‘x’ and ‘y’ equal to each other?
- are both ‘a’ and ‘b’ true?
- is ‘a’ or ‘b’ true?

Conditions form the fundamental building blocks of program logic

- I may sometimes call them “boolean expressions”, but you can still think of them as conditions

Relational Operators

Relational operators identify how two values compare to one another

==	equality
!=	non-equality
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal

Examples:

```
x < y   // is x less than y?

x != 3  // does x equal 3?
```

Logical Operators

Logical operators perform boolean operations (union, intersection, negation)

&&	logical AND
||	logical OR
!	negation (inversion)

Logical AND

	T	F
T	T	F
F	F	F

Logical OR

	T	F
T	T	T
F	T	F

Logical NEGATION

T	F
F	T

The (Growing) Precedence Table

Precedence	Operator(s)	Associativity	Notes
First	( )	innermost	
:	Unary: ++ -- .	⇒	Postfix++
:	Unary: ++ -- + - !	⇐	++Prefix
:	Binary: * / %	⇒	
:	Binary: + -	⇒	
:	Relational: < <= > >=	⇒	
:	Relational: == !=	⇒	
:	Logical: &&	⇒	
:	Logical: ||	⇒	
Last	Assignment: = += -= *= /= %=	⇐	

Numbers as Boolean Values

(and vice-versa)

Converting numbers to booleans:

- `0` is considered `false`
- everything else is `true`

Converting booleans to numbers:

- `false` becomes `0`
- `true` becomes `1`

A Common "Gotcha"

Let's say I need to check if x is between 2 and 10 (exclusive):

- WRONG way to do it:

```
bool inRange = 2 < x < 10;  // doesn't work! rabble rabble rabble!

// this is equivalent to writing:

bool step1 = 2 < x;         // step1 is either true (1) or false (0)

bool inRange = step1 < 10;  // both 0 and 1 are less than 10!
```

- Here's the right way to do it:

```
bool inRange = 2 < x && x < 10;  // hooray! it works!

// this is equivalent to writing:

bool step1 = 2 < x;              // true (1) or false (0)

bool step2 = x < 10;             // true (1) or false (0)

bool inRange = step1 && step2;   // step1 AND step2
```

Potential Test Question:

```
2 <= x <= 1
```

Answer to yourself:

- in what order will C++ evaluate the condition, and why?
- for what values of x will the condition be `true`?
- for what values of x will it be `false`?

Practice!

```
a = 5.5; b = 1.5; k = 3
```

1. `a < 10.0 + k`
2. `a + b >= 6.5`
3. `k != a - b`
4. `b - k > a`
5. `!(a == 3 * b)`
6. `-k <= k + 6`
7. `a < 10 && a > 5`
8. `fabs(k) > 3 || k < b - a`
9. `sqrt(9) != k || !b`
10. `log10(1E4) <= a - b`
11. `floor(a - 2 * k) <= -b`
12. `ceil(a) == 2 * k`
13. `pow(a, 2) > pow(k, 3)`
14. `k < a != floor(b)`

Short-Circuiting

The two binary logical operators, `&&` and `||`, can be short circuited

```
true || x   // will always be true

false && y  // will always be false
```

C++ stops evaluating as soon as it knows the result (short-circuiting):

```
int b = 2;
// what is the final value of b if a is true? if a is false?
bool c = a && b++;


b = 2;
// again, what's the final value of b if a is true? if a is false?
bool d = a || b++;
```