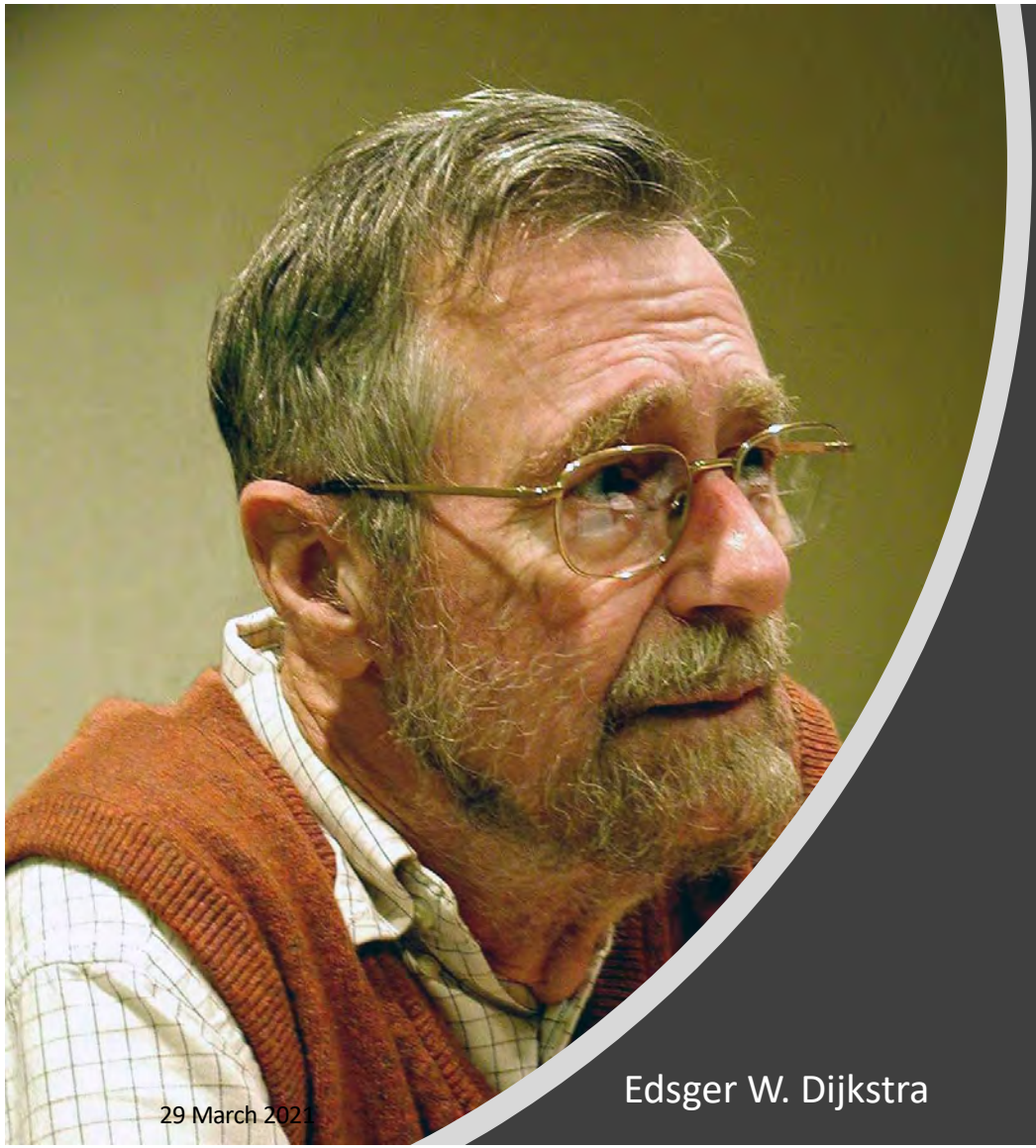


Edsger W. Dijkstra

Control Structures

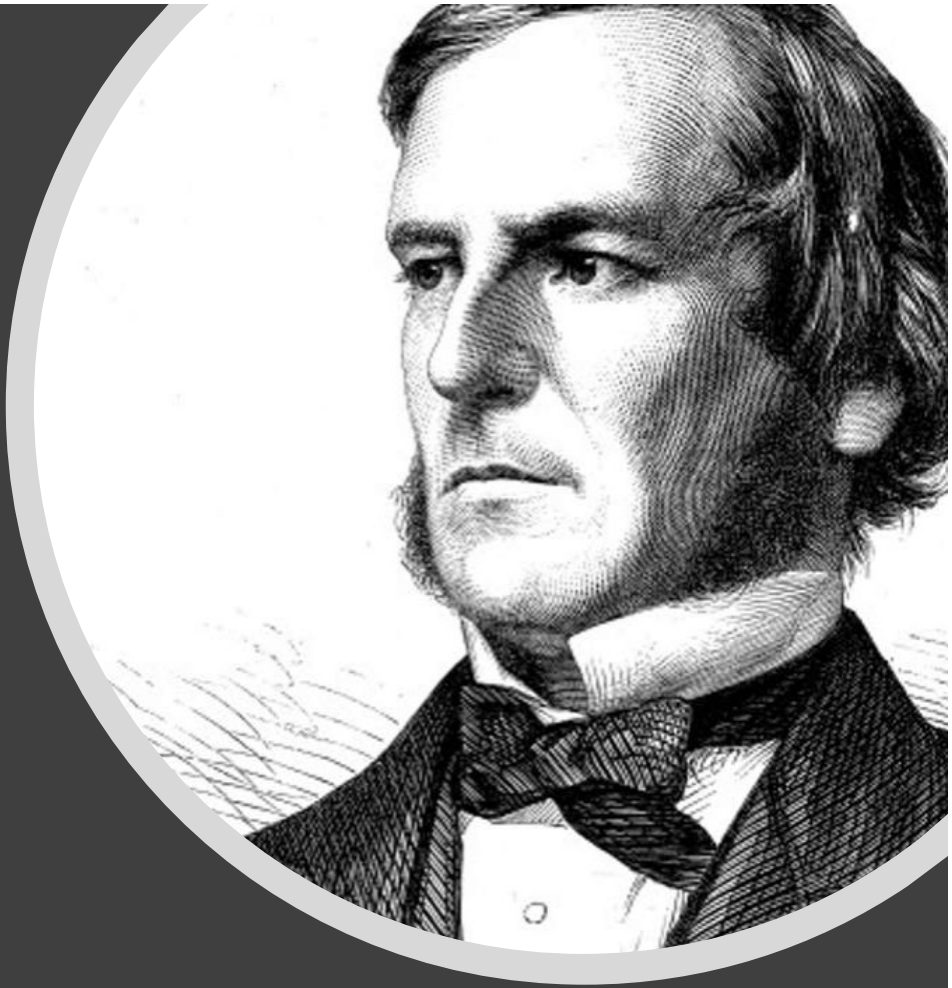
Prof. Darrell Long

CSE 13S

© 2021 Darrell Long

Boolean Algebra

- Sought to set mathematics on a logical foundation.
- Boolean logic has two values: *true* and *false*.
- There are three *basic* operators:
  - $\wedge$ called *and,* also called *conjunction* (`&&` in **C**),
  - $\vee$ called *or,* also called *disjunction* (`||` in **C**), and
  - $\neg$ called *not,* also called *negation* (`!` in C).
- We will use *true* $= 1$ and *false* $= 0$.



George Boole

© 2021 Darrell Long

And (also called *Conjunction*)

- Algebraic properties:
  - $A \wedge (B \wedge C) = (A \wedge B) \wedge C$
  - $A \wedge B = B \wedge A$
  - $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  - $A \wedge 1 = A$
  - $A \wedge 0 = 0$
  - $A \wedge A = A$

∧ (and)	**T**	**F**
T	T	F
F	F	F

© 2021 Darrell Long

Or (also called *Inclusive-or* or *Disjunction*)

- Algebraic properties:
  - $A \vee (B \vee C) = (A \vee B) \vee C$
  - $A \vee B = B \vee A$
  - $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$
  - $A \vee 0 = A$
  - $A \vee 1 = 1$
  - $A \vee A = A$

∨ (or)	**T**	**F**
T	T	T
F	T	F

© 2021 Darrell Long

Not (also called *Negation*)

- Algebraic properties:
  - $A \wedge \neg A = 0$
  - $A \vee \neg A = 1$
  - $\neg(\neg A) = A$
- DeMorgan's Law:
  - $\neg(A \vee B) = \neg\mathrm{A} \wedge \neg\mathrm{B}$
  - $\neg(A \wedge B) = \neg A \vee \neg B$

¬ (not)	
T	F
F	T

© 2021 Darrell Long

Exclusive-or

- $A \oplus B = (A \vee B) \wedge \neg(A \wedge B)$
- $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$
- Some algebraic properties:
  - $A \oplus A = 0$
  - $A \oplus 0 = A$
  - $A \oplus 1 = \neg A$
  - $A \oplus (B \oplus C) = (A \oplus B) \oplus C$

⊕ (XOR)	**T**	**F**
T	F	T
F	T	F

© 2021 Darrell Long

Logical Operators

<	Less than
<=	Less than or equal
==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal

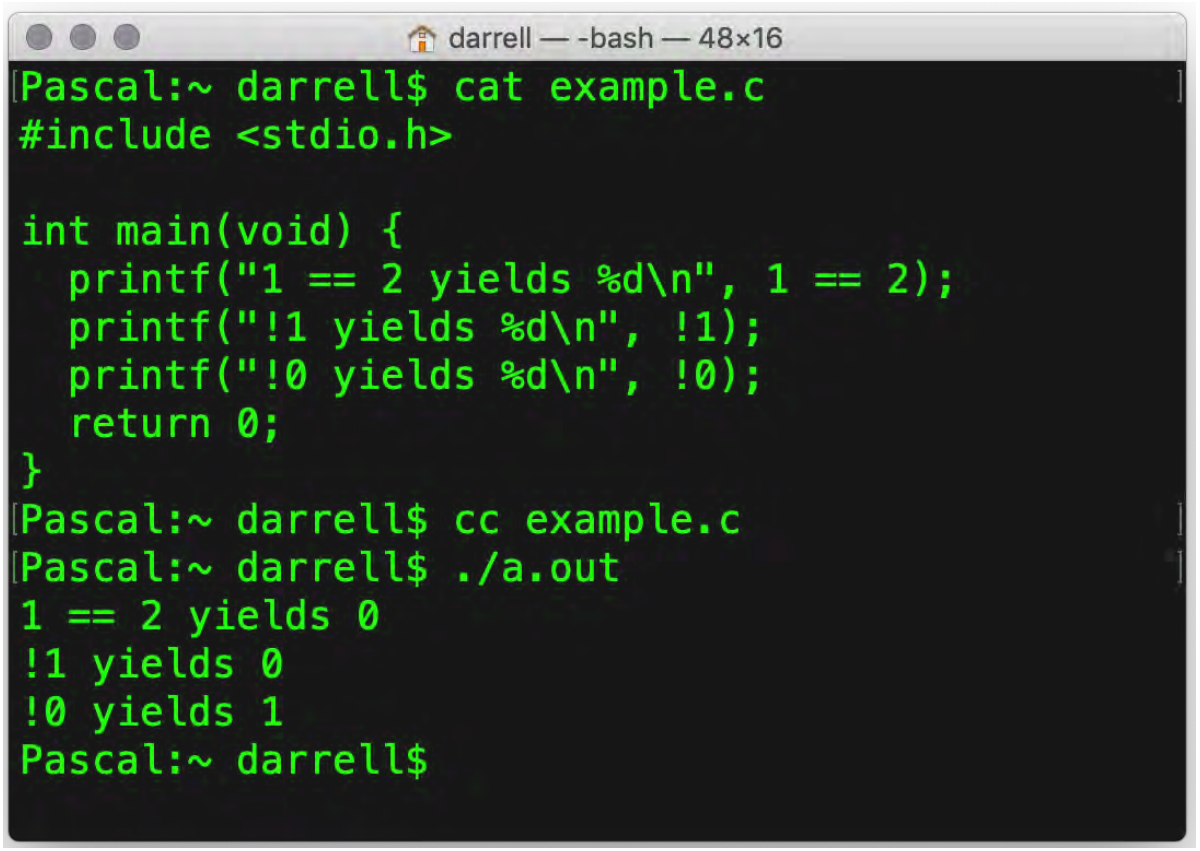
© 2021 Darrell Long

True and False in **C**

- In **C**, zero (0) is *false*.
- All that is not *false* is *true*.
- Logical expressions have type `int`.

- You can have *true* and *false* if you:

```
#include <stdbool.h>
```

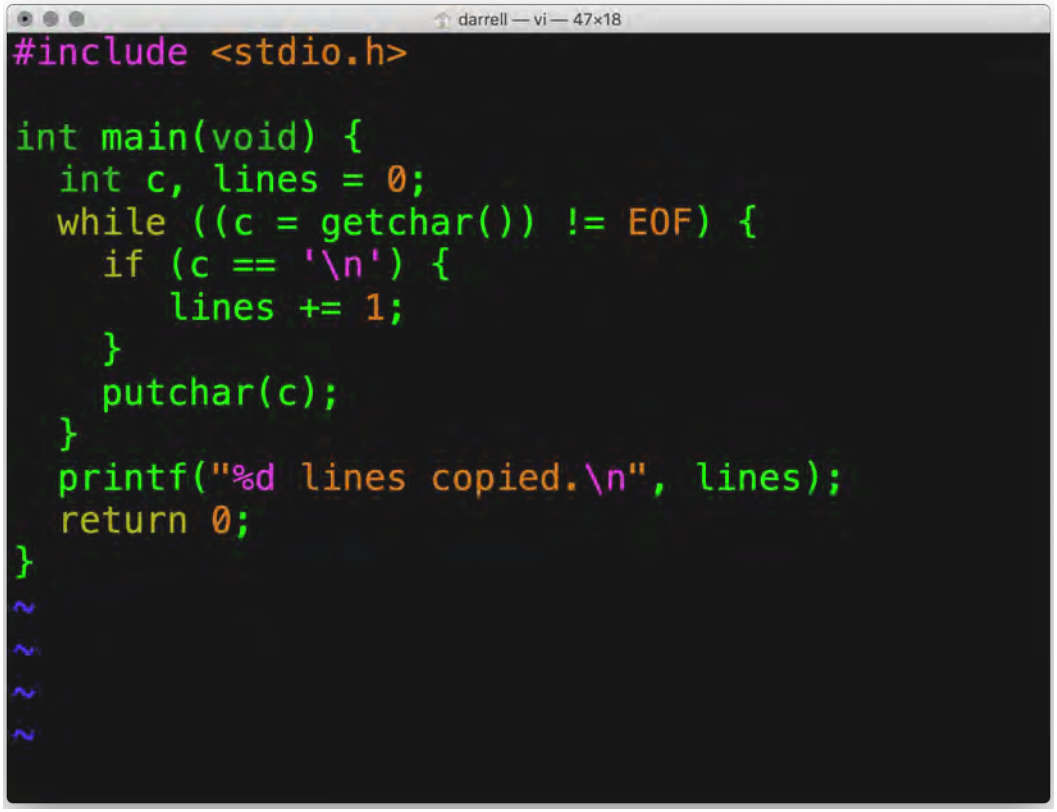


© 2021 Darrell Long

if ( )

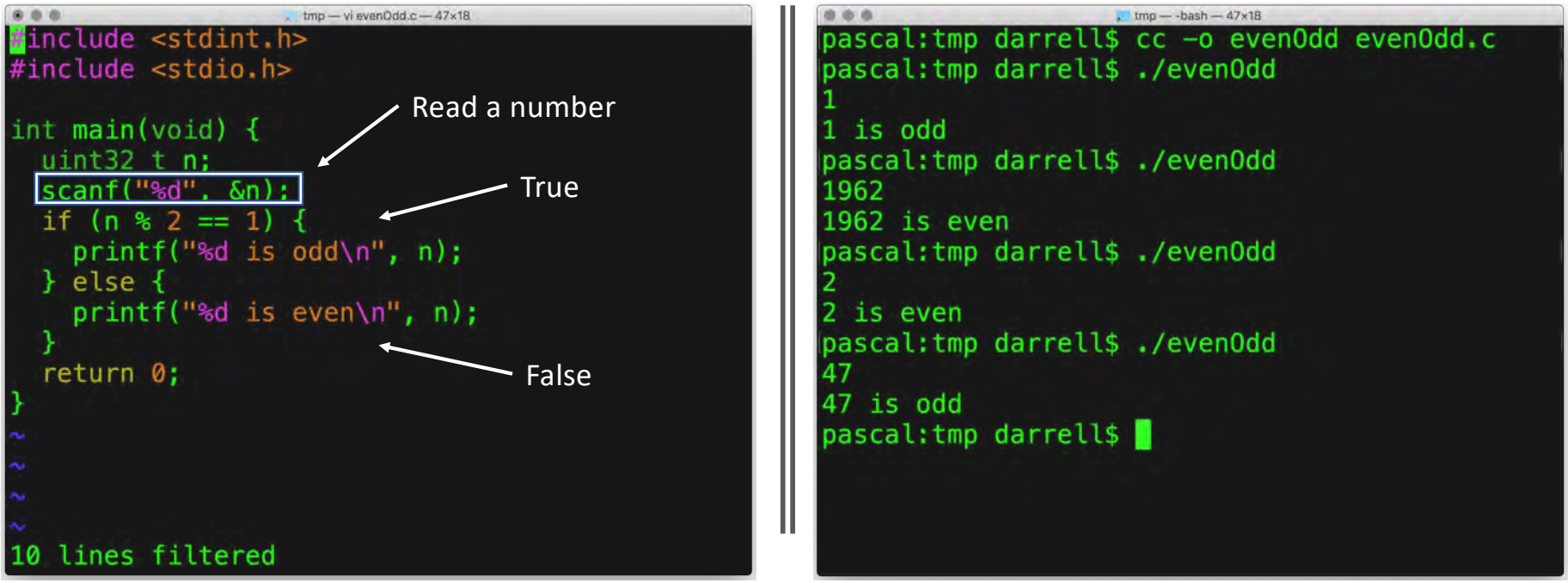
- if executes the next statement if the Boolean expression is true.
- Even through { } are not required for a single statement, *always* use them.

- Why?
  - It avoids errors when adding statements.

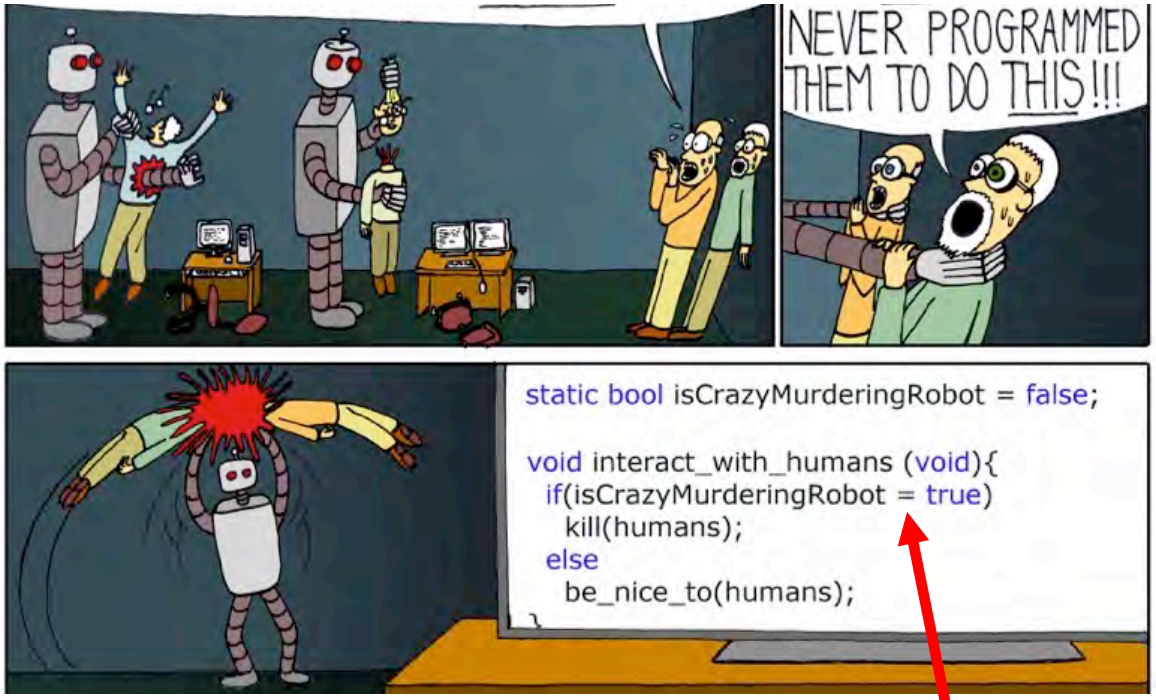


© 2021 Darrell Long

if () { … } else { … }



© 2021 Darrell Long



$= \neq ==$

Oops!

`(x = 17)` is always *true*.

What?!?

Assignment (=)

is not the same as

Equality (==)

© 2021 Darrell Long

Nested `if ( )`

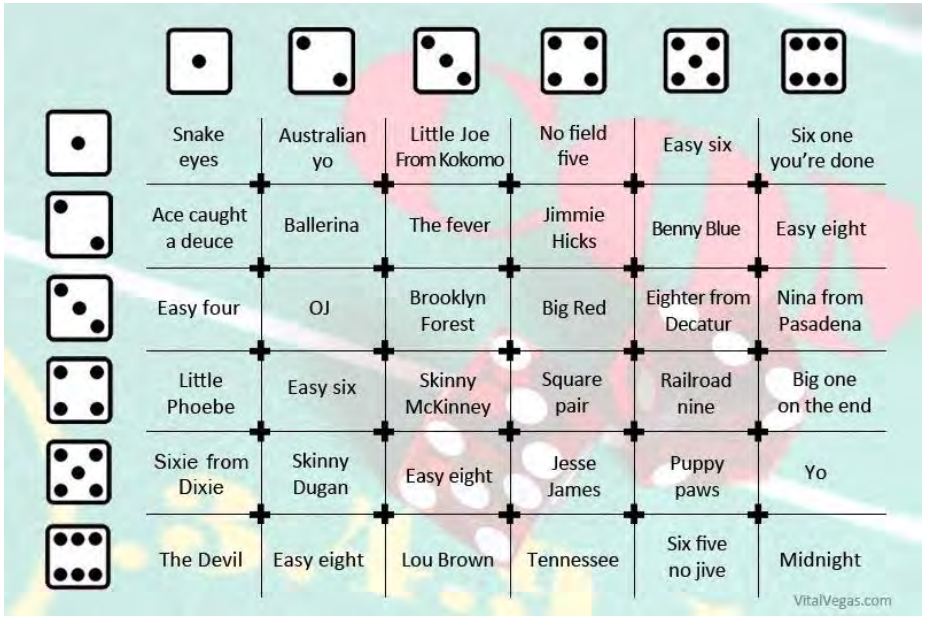
```
tmp — vi min.c — 52×21
#include <stdio.h>

int main(void) {
  int a, b, c;
  scanf("%d %d %d", &a, &b, &c);
  if (a < b) {
    if (c < a) {
      printf("c = %d is smallest\n", c);
    } else {
      printf("a = %d is smallest\n", a);
    }
  } else {
    if (c < b) {
      printf("c = %d is smallest\n", c);
    } else {
      printf("b = %d is smallest\n", b);
    }
  }
  return 0;
}
"min.c" 20L, 365C
```

```
tmp — -bash — 52×21
pascal:tmp darrell$ cc min.c -o min
pascal:tmp darrell$ ./min
1 2 3
a = 1 is smallest
pascal:tmp darrell$ ./min
17 -5 2
b = -5 is smallest
pascal:tmp darrell$
```


© 2021 Darrell Long

`if () { … } else if () { … } … else { … }`

```
darrell — vi dice.c — 51×17
if (die_1 == 1 && die_2 == 1) {
  printf("Snake Eyes\n");
} else if (die_1 == 2 && die_2 == 2) {
  printf("Ballerina\n");
} else if (die_1 == 3 && die_2 == 3) {
  printf("Brooklyn Forest\n");
} else if (die_1 == 4 && die_2 == 4) {
  printf("Square Pair\n");
} else if (die_1 == 5 && die_2 == 5) {
  printf("Puppy Paws\n");
} else if (die_1 == 5 && die_2 == 5) {
  printf("Midnight\n");
} else {
  printf("Off Diagonal %d\n", die_1 + die_2);
~
```



© 2021 Darrell Long

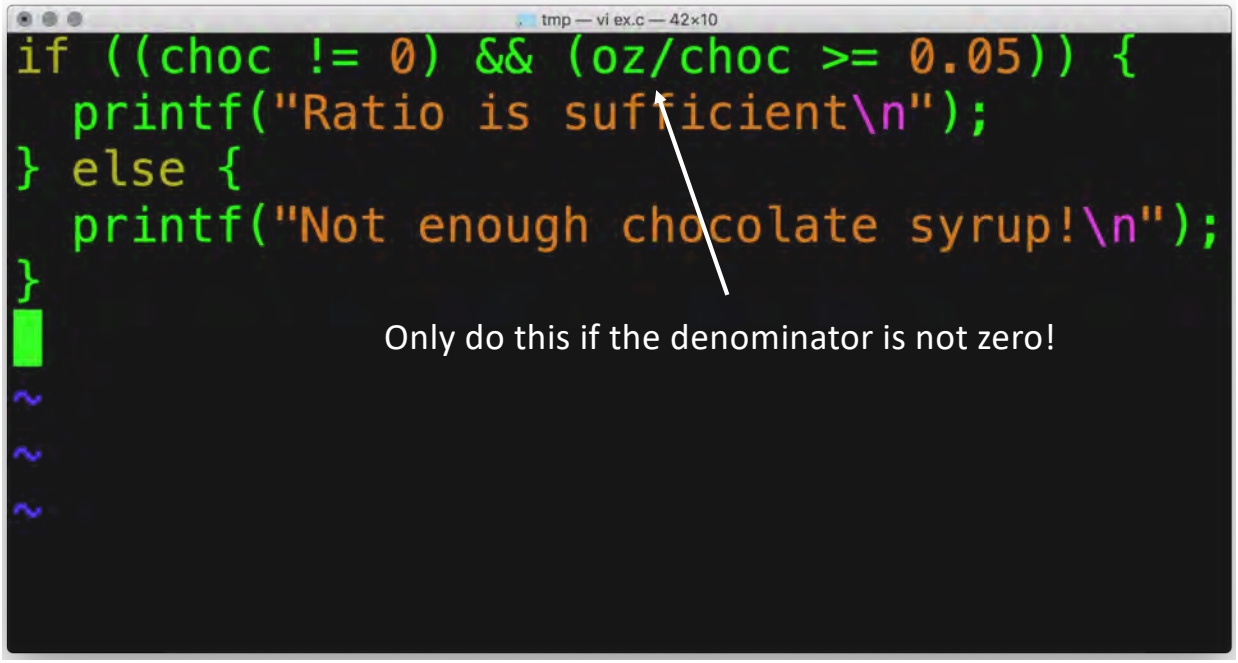
Boolean Operators

&&	Boolean “and” (∧)
||	Boolean “or” (∨)
!	Boolean “not” (¬)

© 2021 Darrell Long

Short-circuit Evaluation

- *false* && anything is *false*
- *true* || anything is *true*
- Stop evaluating as soon as we know the result.
- Suppose we evaluated the entire expression:
  - We would be dividing by zero.
- We could follow a *null* pointer.
  - More on that later.



© 2021 Darrell Long

switch ()

tmp — vi switch.c — 80×24

```
#include <stdio.h>

int main(void) {
  char *msg;
  int x;
  scanf("%d", &x);
  switch (x % 2) {
  case 0:
    msg = "even";
    break;
  case 1:
    msg = "odd";
    break;
  default:
    msg = "wait, what?";
  }

  printf("%d is %s\n", x, msg);
  return 0;
}
~
~
~
"switch.c" 20L, 261C
```

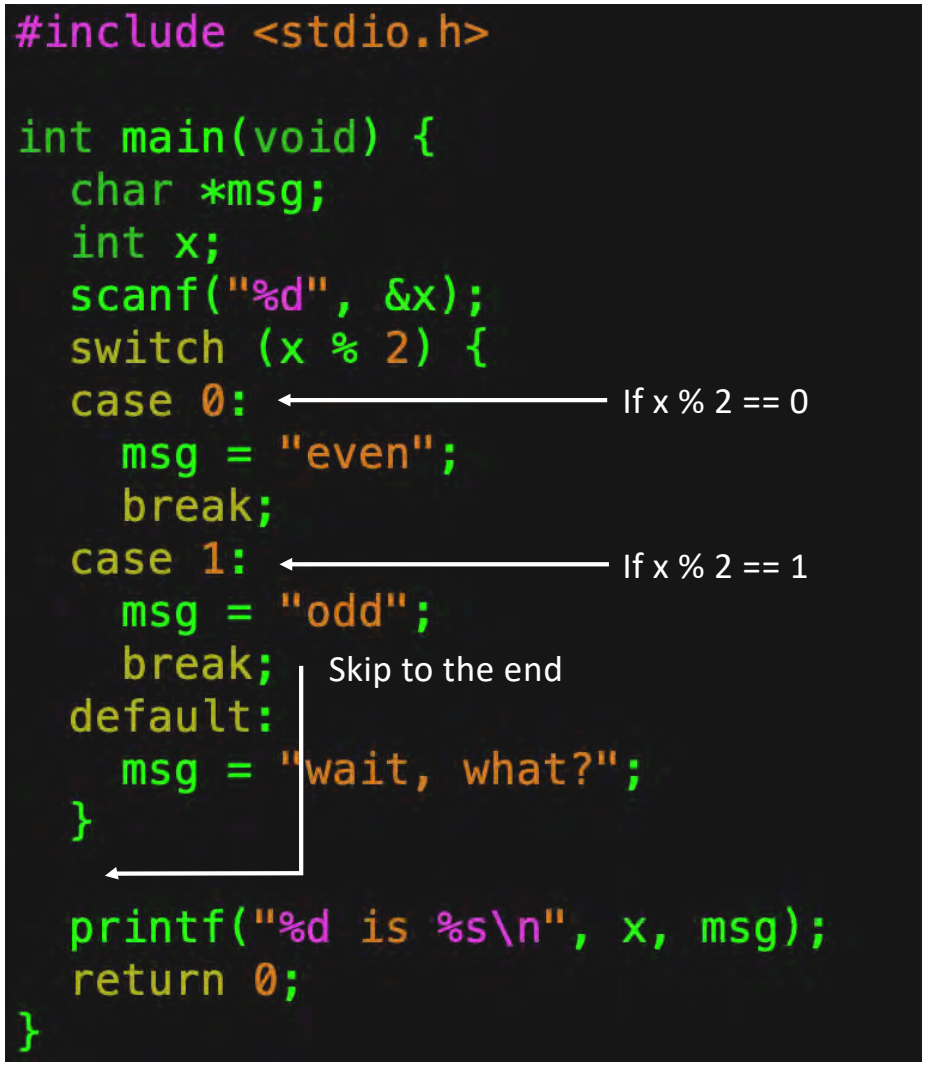
tmp — -bash — 80×24

```
pascal:tmp darrell$ cc -o switch switch.c
pascal:tmp darrell$ ./switch
12
12 is even
pascal:tmp darrell$ ./switch
13
13 is odd
pascal:tmp darrell$ ./switch
-17
-17 is wait, what?
pascal:tmp darrell$
```


© 2021 Darrell Long

`switch ()`

- Allows you to select among a fixed set of alternatives.
- If none of them are present, then `default` is chosen.
- Use `break` to skip to the end
  - If you forget, it will *fall through* to the next case.



© 2021 Darrell Long

Equivalent `goto` Code

```
if (x % 2 == 0) goto even;
if (x % 2 == 1) goto odd;
goto confused;
even: msg = "even";
goto done;
odd: msg = "odd";
goto done;
confused: msg = "wait, what?"
done:
```

← Like `break`

© 2021 Darrell Long

Dr. Emilio Lizardo

What is wrong with `goto`?

- It is not so much not knowing where you're going, but knowing how you got there.
- You could, for example, *goto* the middle of:
  - An `if` statement,
  - A `switch` statement, or
  - A loop
- It is not even clear what that would mean.

'Cause remember: no matter where you go, there you are. —*Buckaroo Banzai*

© 2021 Darrell Long

When can I use `goto`?

- *One place*: non-local error handling.
- This is when an exceptional condition—an error—that you cannot handle occurs.
- *It is not pretty.*
- More modern languages like C++ provide an exception handling mechanism.

```
{
  { // ... stuff ...
    { // ... stuff ...
      if (terrible thing) goto run_away;
    }
  }
  // ... stuff ...

run_away: // Try to fix the problem?
```


© 2021 Darrell Long

Summary

- Boolean logic is a branch of mathematics dealing with just two values: true and false.
- Since computers have just two values, 0 and 1, it is a perfect fit.
- C uses 0 for *false*, and non-zero for *true*.
- Short-circuit evaluation allows the expression to be determined as soon as possible.

- `if` (*expression*) *statement*
  - Executes the statement if the is non-zero.
- `else` *statement*
  - Executes if the *statement* was *zero*.
- `switch` (*value*) selects cases correspond to the value.
- `goto` jumps to a label.
  - Do not use it without a *very* good reason.

© 2021 Darrell Long