APS105 Lecture 5
3.1-3.2

Dan Zingaro

January 27, 2010

Feedback from Reading Quiz

The following program does not run.

```
#include <stdio.h>

int main (void) {
  int q = 4;
  if (q == 4)
    printf ("The if-statement is running.\n");
    printf ("Another printf.\n");
  else
    printf ("This is the else.\n");
  return 0;
}
```

Try to compile the program. Explain each error that your compiler gives, and suggest a solution.

- Isn't a `scanf` required?
- Can we even have an `if` without a `scanf`?

Feedback from Reading Quiz...

- Why is `(7 / 3) = 2` an illegal expression?
- If I use a `switch`, can I leave out the `return 0` at the end of `main`?
  - No. It should always be there
- Why do I have to include `stdbool.h` to use `bool`, `true`, and `false`?
  - These three words are not C keywords

Relational expressions

- Relational expressions compare two values
- The relational operators are <, <=, >, >=, == (equal to) and != (not equal to)
- If the relationship holds, the relational operators will evaluate to 1
- If the relationship does not hold, the relational operators will evaluate to 0
- e.g. `3 < 4` evaluates to 1
- e.g. `4 < 3` evaluates to 0

Relational expressions...

- Careful: = is the assignment operator, and == is the equality operator!
  - Let's make sure we understand the difference!
- Arithmetic operators have higher precedence than the relational operators
- e.g. `0.2 + 3 < 2` evaluates to `3.2 < 2`, which evaluates to `0`
- The assignment operator has lower precedence than the relational operators

ConcepTest

What is the output of the following code?

```
int x = 2 < 3;
int y = 2 != 2;
int z = 2 == 2 == 2;
printf ("%d\n", x + y + z);
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. Compilation error!

The If-statement

- If-statements let us make decisions in our code
- The syntax of an if-statement is

```
if (<expression>)
  <statement>
```

- Here, if `<expression>` is true, the `<statement>` is executed, otherwise `<statement>` is skipped
- In C, any non-zero value is considered to be true, and 0 is considered to be false

ConcepTest

Here are some if-statements, with placeholders for their bodies.
(I) `if (2 < 3) S1;`
(II) `if (2) S2;`
(III) `if (0 == 0) S3;`
(IV) `if (0) S4;`
Which statements will be executed?

- A. (I)
- B. (I) and (II)
- C. (I) and (III)
- D. (I), (II) and (III)
- E. (II) and (IV)

If and Else

- We can optionally include an `else` clause in our if-statements

```
if (<expression>)
  <statement1>
else
  <statement2>
```

- This time, if `<expression>` is true, `<statement1>` is executed, otherwise `<statement2>` is executed
- Exactly one of the two statements is guaranteed to execute
- It is impossible for both statements to be executed

Compound Statements

- What if we want to include more than one statement in the `if` or `else` portion of an if-statement?
- We can use multiple statements if we group them inside of `{` and `}` braces
- e.g. below, if `x` is 2, both the assignment and the `printf` are executed; if `x` is not 2, both statements of the compound statement are skipped

```
if (x == 2) {
  x = x - 1;
  printf ("Only one item allowed!\n");
}
```

Boolean Variables

- C treats 0 as false, and everything else as true
- If we must store values of 0 and 1 representing false and true, we can use an `int` as we have been doing
- C also has a `bool` type that can hold only the two integers 0 and 1
- If you try to store something besides 0 or 1 in a `bool` variable, 1 ("true") gets stored
- C also defines the constant `true` to mean 1, and the constant `false` to mean 0
- The benefit of `bool` over `int` is that `bool` makes it explicit that we are using the variable as a boolean truth value

Example: Boolean Variables

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

int main (void) {
  const int VOTING_AGE = 18;
  int age;
  bool eligible;
  scanf ("%d", &age);
  eligible = age >= VOTING_AGE;
  if (eligible)
    printf ("Eligible to vote.\n");
  else
    printf ("Not eligible to vote.\n");
  return 0;
}
```