Slide 1

Administrivia

- Reminder: Homework 2 due Monday.

Slide 2

Type Conversions

- Implicit conversions: When you assign a value of one type to another (e.g., `float` to `int`), or write an expression that mixes types, C will perform an implicit conversion.
- Explicit conversions: Putting a type in parentheses before an expression means you want to convert to the indicated type. Example:

```
(float) (1 / 2)
```

versus

```
(float) 1 / (float) 2
```

Slide 3

Tracing Code

- A valuable skill to have is working through what the computer will do when it executes your program — "tracing code" (also known as "playing computer").
- Idea is to write down names of variables, their values; when one changes, cross out old value and put in new one.
- Let's do an example . . .

Slide 4

Defining Named Constants with Preprocessor Directives

- Sometimes it makes sense to use numeric constants in programs — e.g., in the Fahrenheit-to-Celsius temperature conversion program (homework).
- But sometimes it's more readable, for humans, to give these constants a name. Can do this with `#define`. Examples:

```
#define DAYS_IN_YEAR 365
#define SECONDS_IN_YEAR (365*24*60*60)
```

  Then when you write `DAYS_IN_YEAR`, compiler (strictly speaking, its preprocessor) replaces it with 365.

  Notice also that if we need to calculate something, as in the second example, it's usually more readable to just write out the expression and let the compiler do the calculation.

Slide 5

Conditional Execution

- So far all our programs have executed the same statements every time, just maybe with different numbers.
- Often, though, we want to be able to do different things in different circumstances — for example, print an error message and stop if the input values don't make sense (such as a negative number for the program to make change).
- So, C (like most languages) provides some constructs for *conditional execution*. Before we talk about them, we need . . .

Slide 6

Boolean Expressions

- A *Boolean value* is either *true* or *false*; a *Boolean expression* is something that evaluates to true or false.
- We can make simple examples in C using familiar math comparison operators. Examples:
  - `x > 10`
  - `y <= 5`
  - `x == y` (*Note* the use of == and not =.)

Slide 7

Boolean Expressions, Continued

- *Boolean algebra* defines some operators on these values; the most important for us are written in C as
  - `!` — "not", true if the operand is false.
  - `&&` — "and", true if both operands are true.
  - `||` — "or", true if either operand is true (or both are).
- Can use these to build up complex expressions. As with arithmetic expressions, use parentheses when in doubt. Examples:
  - `(x >= 0) && (x <= 10)`
  - `!(x == y)` (though we could also just write `x != y`).

Slide 8

Boolean Expressions in C

- Although there are only two Boolean values, C represents them as `int`s, with 0 meaning true and anything else meaning false. (Usually you don't care about this, but it can be good to know.)
- This means that the compiler will accept both `x == y` and `x = y`, but they mean different things. Very common mistake (and not just for beginners!). Compiler will often warn you about this (though you may need to use that `-Wall` flag).

Slide 9

Conditional Execution — `if/else`

- To execute a statement if an expression evaluates to true, use `if`:

```
if (x > 0)
    printf("greater than zero\n");
```

- To execute one statement if an expression is true, another if it's false, use `if` and `else`:

```
if (x > 0)
    printf("greater than zero\n");
else
    printf("not greater than zero\n");
```

Slide 10

`if/else`, Continued

- To execute a group ("block") of statements rather than just a single statement, use curly braces for grouping:

```
if (x > 0) {
    printf("greater than zero\n");
    printf("and that is good\n");
}
else {
    printf("not greater than zero\n");
    printf("and that is bad\n");
}
```

- What happens if you forget the braces? The program may still compile and run, but it probably won't do what you meant.

Slide 11

`if/else`, Continued

- Several styles for where to put the curly braces. Which is best? Some people care; I say pick one that's readable (to humans) and stick with it.

Slide 12

Conditional Execution, Continued

- What if more than two? We could "nest" `if/else` constructs, e.g.,

```
if (x < 0) {
    printf("less than\n");
}
else {
    if (x > 0) {
        printf("greater than\n");
    }
    else {
        printf("equal\n");
    }
}
```

- But this gets ugly fairly quickly. So . . .

Slide 13

Conditional Execution, Continued

- Better:

```
if (x < 0) {
    printf("less than\n");
}
else if (x > 0) {
    printf("greater than\n");
}
else {
    printf("equal\n");
}
```

- Can have as many cases as we need; can omit `else` if not needed.

Slide 14

Conditional Execution, Continued

- Sometimes we can go further, though. If all of the conditions are of the form

  integer_expression == *value*

  then we can use the `switch` construct. Notice that characters (`char`) count as integers in this context.

- Example (similar to calculator example in book) on next slide.

Slide 15

Conditional Execution, Continued

```
char menu_pick;  /* should be one of '+', '-' */
/* .... */
switch (menu_pick) {
    case '+':
        result = input1 + input2;
        break;
    case '-':
        result = input1 + input2;
        break;
    default:
        result = 0;
        printf("operator not recognized\n");
}
```

Slide 16

Simple I/O, Revisited

- We can now do simple error-checking that `scanf` did what we asked. C-idiomatic way looks like this simple example:

```
if (scanf("%d", &x) == 1)
    /* okay */
else
    /* error */
```

- For this class it's usually best to just bail out on bad input, rather than retrying.

Slide 17

Conditional Expressions

- C also provides a short way to express things of the form

  `if (`*condition*`)`

  variable = *value1*

  `else`

  variable = *value2*

  namely the ternary (three operands) operator `?`.

- Example:

```
sign = (x >= 0) ? 1 : -1;
```

  assigns 1 to `sign` if `x` is non-negative, -1 otherwise.

- (Use with caution — compact, but can easily lead to code that's difficult for humans to understand.)

Slide 18

Minute Essay

- Have you previously used something that supports conditional execution (Matlab?), and if so how does C's version compare to it?