

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:

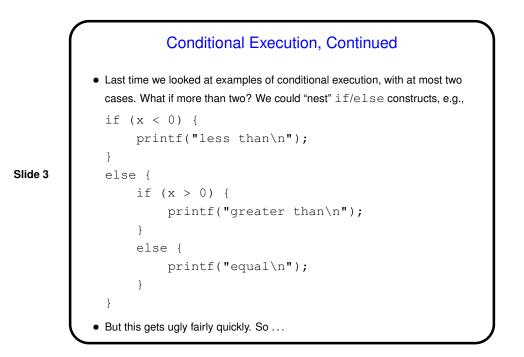
#define DAYS_IN_YEAR 365

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

Then when you write ${\tt DAYS_IN_YEAR},$ compiler (strictly speaking, its preprocessor) replaces it with 365.

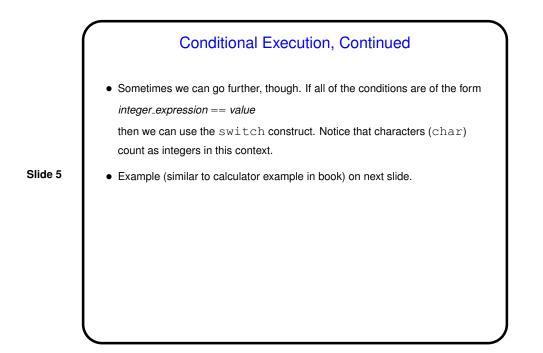
Notice also that if we need to calculate something, as in the second example, it's usually more readable to just write out the expression and let the compiler do the calculation.

 See revised program to make change, linked from the "Sample programs" page <u>here</u>.

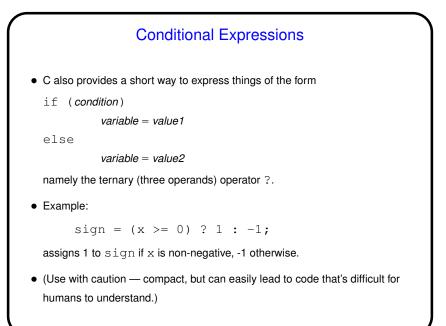


Conditional Execution, Continued

• 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.

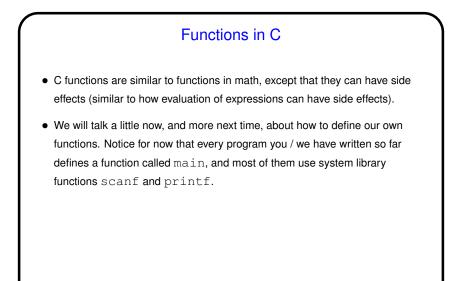


Conditional Execution, Continued
• char menu_pick; /* should be one of '+', '-' */
 /* */
 switch (menu_pick) {
 case '+':
 result = input1 + input2;
 break;
 case '-':
 result = input1 + input2;
 break;
 default:
 result = 0;
 printf("operator not recognized\n");
 }



Functions and Problem Decomposition

- So far all our programs have been one big chunk of code. This is okay for simple programs, but quickly becomes difficult to understand as problems get bigger.
- Further, some things we don't want to, or can't, really write ourselves, such as the code for input/output.
- So C, like many/most other programming languages, gives you a way of decomposing problems into subproblems. C calls them *functions*. Using this feature to good effect is something of an art, but may teach you something about problem decomposition in general, which is a useful skill.



Functions in C, Continued Every function has A name (where rules for names are the same as those for variables). Zero or more inputs (called *parameters*). A return type (void to indicate that the function doesn't return anything). Some code to be executed when the function is called. When you call (use) a function, you Supply values for inputs (pass in values for parameters). Optionally, use the value returned by the function. The function call is an expression, as discussed previously, and its value is the value returned by the function.

