APS105 Lecture 3
1.5-1.9

Dan Zingaro

January 22, 2010

Feedback from Reading Quiz

- Please, check out the Bulletin Board for more!
- What is a literal?
  - If we say `x = 5;`, `x` is a literal
  - If we say `c = 'd';`, `'d'` is a literal
  - Literals are constants (numbers or characters) that you see in the source code
- How do we use `bool` as a data type? How can we use `false` and `true` as values?

```
bool happy;
happy = true;
```

Floating-Point Numbers

- To store numeric values containing decimals or to store integers that cannot fit in a `long`, we can use floating-point numbers
- In C, these numbers are called `float` and `double`
- `double` has more accuracy than `float`
- Floating-point numbers must include a decimal point and/or an exponent
- e.g. `-23.456` or `-2.3456E1`
- The `E1` means "multiply by 10^1"; this is scientific notation
- e.g. `5.6E2` means $5.6 * 10^2 = 560$

Variables

- To store values so we can retrieve them later, we use variables
- A variable is a name (an identifier) for a location in memory
- A declaration gives the type and the identifier of a variable
- Types are things like `int` and `float`
- e.g. to declare an integer variable referred to by age, we write `int age;`
- This selects a memory location for us and calls it age
- But, declaring a variable does not put anything in that memory location!

ConcepTest

After we say `int age;`, we can refer to the selected memory location with the identifier `age`. What might already be stored in that memory location?

- A. `0`
- B. `0.0`
- C. `"abcd"`
- D. All of the above are possible

Assignment Statements

- To give a value to a variable (i.e. to put a value in the memory location identified by a variable name), we can use an assignment statement
- `<identifier> = <expression>`
- Two steps:
  1. First, evaluate the expression on the right-hand side
  2. Then, store the result in the variable on the left-hand side
- The `<identifier>` and `<expression>` do not have to be the same type, but we must be careful if they are not
- e.g. assigning a `float` or `double` value to an `int` variable loses the fraction

ConcepTest

What is the value of `y` after the execution of this code?

```
int x = 37;
int y = x + 2;
x = 20;
```

- A. 39
- B. 2
- C. 22
- D. 20

Printing Variables

- The same `printf` function we used for outputting lines of text can be used to output values of variables
- Rather than providing just one parameter, we call `printf` with more than one parameter
- The first parameter is the control string
- It indicates the text to print, and includes placeholders (also called format specifiers) for each value we want
- We provide one additional parameter for each placeholder in the control string

Printing Variables: Example

```
int cents = 5;
printf ("The total is %d cents.\n", cents);
printf ("Twice the total is %d cents.\n", cents * 2);
```

- The %d is a placeholder for an integer variable
- It means "replace the %d with the next parameter passed to printf"
- There are other placeholders: %f for floating-point, %c for character ...

Printing Variables: Example...

```
int dollars = 8;
int cents = 5;
printf ("The total is %d dollars and %d cents.\n",
        dollars, cents);
```

- This means "replace the first %d with the first parameter after the control string (`dollars`), and replace the second %d with the second parameter after the control string (`cents`)

Getting Input

- To get input from the user, we can use `scanf`
- It's similar to `printf`: it takes a control string with placeholders, and then one additional parameter for each placeholder
- Each parameter after the control string must be the memory location of a variable, telling `scanf` where to store the corresponding information
- For each placeholder, `scanf` reads input as long as it gets valid characters for the type of the placeholder

Getting Input...

This will not work

```
int age;
scanf ("%d", age);
```

- The value of age is arbitrary – we haven't set it!
- If it has value 52, then `scanf` is going to take what the user types and stuff it into memory location 52
- This is not going to put the value in the memory location for age!
  - It's going to overwrite whatever used to be at location 52!

ConcepTest

What is the result of the following code?

```
int age;
age = 0;
scanf ("%d", age);
```

- A. Ask the user for input, and *p*roperly store it in the memory location for age
- B. Ask the user for input, and store it in memory location 0
- C. Ask the user for input, and store it in an arbitrary memory location

Getting Input...

- If we want `scanf` to store a value into the memory location for age, we have to tell it what that memory location is
- If age refers to memory location 1284, we want `scanf` to put the value in location 1284
- If age refers to memory location 9822, we want `scanf` to put the value in location 9822
- We require a way to get the memory location of age
- We do this with the `&` operator
- `scanf ("%d", &age);`