Input/output with printf/scanf

An introduction to basic i/o using the cstdio library

- printf for string literals
- embedding common data types in output (%f, %d, etc)
- formatting options with printf
- basic use of scanf with variables
- impact of whitespace on scanf
- reading formatted input with scanf
- printf and scanf return values

Basic printf for text output

- if the cstdio library has been included then the printf routine is available for use
- printf displays text strings to standard output (generally the terminal window the program is running in)
  - `printf(“this is output”);`
- all printfs appear sequentially on screen
- special characters can be embedded in the string:
  - \n for a newline, \t for a tab, \g for a bell, and many others

Embedding data values in printf

- to embed a variable (or constant, or expression) value
  - we put a special marker in the string where the value is desired (a % sign followed by a special designator character)
  - we follow the text string with the value to be used
- formats: %d for integer values, %f for floating point, %c for char, %ld for long ints, %lf for doubles, %s for text strings

```
int x = 3;
float y = 1.5;
printf(“some integer %d\n”, x);
printf(“some float %f\n”, y);
```

Padding printf output

- We can specify a minimum width (characters) for the item we are displaying, as an integer after the %

```
printf("here is x: %5d\n", x);
```

- if x is less than 5 characters wide it adds extra spaces before displaying x, padding to width 5
- if x is 5 or more characters wide then it displays x normally

Formatting floats with printf

- for floats, we can specify both the overall width and the number of digits to display after the decimal point
- printf(“%6.2f”, y);
- puts 2 digits after the decimal point, pads to total width of 6 characters (including the decimal point)
- %g can be used to autoformat floats: it drops any trailing 0's, e.g. 3.650000 comes out as 3.65, 3.000000 comes out as 3, etc

Scanf to read values into variables

- scanf reads from standard input (generally the keyboard)
- it uses a format string to specify what kind of data is expected (%d for int, %f for float, etc)
- the string is followed by the & sign and the variable to store the value in

```
    scanf("%d", &x); // read an int into variable x
```

- scanfs read sequentially from the data entered, assuming all data fields are whitespace delimited
- typed text is not available to scanf until enter has been hit

Whitespace and scanf

- scanf skips any leading whitespace (spaces, tabs, newlines, etc) and starts reading when other characters are encountered
- it reads characters as long as they are valid for the input type, stopping when the next whitespace is encountered or the next character is invalid for the expected data type

```
    int i;
    scanf(“%d”, &i);
```

- skips whitespace, reads digits into variable i, stops when it sees something that isn't a digit

Formatted scanf input

- scanf strings can read multiple variables, and can include text that is expected but is not to be stored, e.g.

```
scanf(“%d-%d”, &x, &y);
```

  - reads an integer into x
  - reads and skips a '-' character
  - reads an integer into y
- scanf stops if a specific character (e.g. the '-') is expected but not found

printf and scanf return values

- printf returns a value, a count of the total number of characters printed to standard output

```
int x = 470;
int charCount;
charCount = printf(“x is %d\n”, x);
```

- charCount is 9 (counting spaces, newlines, 3 for the 470)
- scanf similarly returns a value, a count of the number of characters read from standard input