Lecture 3: Introduction to C: Input & Output, Assignments, Math functions
Generating output

• A typical program gets a set of inputs and generates a set of outputs

• How can do we generate output in C?

• One basic approach is to print some text to the screen

• There is a library function called printf in the standard input-output library stdio that lets us do that
Formatted printing with printf

• Basic form:
  printf("format", object1, object2, ...)

• Format is a string that contains specifiers (one per listed object) written as %i, %f, etc. denoting how each object is going to be printed (%i means integer value, %f means float value.
• The object could be a variable or an expression

  printf("x = %d", x);
  /* %d prints the decimal value, same as %i, x is an integer */

  –Or use %o or %x to show same value in octal or hexadecimal
Formatted printing with printf

• Use %c to print character representation of int or char
  printf("%c", 65); /* prints A */
  – See, try, and understand values.c

• %f or %e or %g for floating point, and %s for strings
Formatted printing with printf

- A minimum field width and precision can be used to specify how many characters will be printed

```c
printf("%3.2f \n", v);
```
- 3 is the field width and 2 is the precision

- We can require that a sign should be printed using "+

```c
printf("%+3.2f \n", v);
```

- If the field width specifies more positions than needed the value is printed right-justified, we can change it to left-justified using "-

```c
printf("%-9s %7.2f \n", "Value", v);
```

- More conversion specifiers in Table 2.6 (p. 44), and next
## Conversion Specifiers for Output

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Output Type</th>
<th>Specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>short, int</td>
<td>int</td>
<td>%i, %d</td>
</tr>
<tr>
<td>int</td>
<td>short</td>
<td>%hi, %hd</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>%li, %ld</td>
</tr>
<tr>
<td>int</td>
<td>unsigned int</td>
<td>%u</td>
</tr>
<tr>
<td>int</td>
<td>unsigned short</td>
<td>%hu</td>
</tr>
<tr>
<td>long</td>
<td>unsigned long</td>
<td>%lu</td>
</tr>
<tr>
<td><strong>Floating-Point Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>float, double</td>
<td>double</td>
<td>%f, %e, %E, %g, %G</td>
</tr>
<tr>
<td>long double</td>
<td>long double</td>
<td>%Lf, %Le, %Le, %Lg, %LG</td>
</tr>
<tr>
<td><strong>Character Values</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>char</td>
<td>%c</td>
</tr>
</tbody>
</table>
Getting data with `scanf`

- Form/features similar to `printf` function: `scanf("format", memory1, memory2, ...)`
  - Notice memory locations to store input

- For example, get an integer from user:
  ```c
  int x;
  printf("Enter an integer: ");
  scanf("%i", &x);
  ```
  - `&x` means "address of x" – a memory location

- Essentially same conversion specifiers as `printf`
  - Careful though: e.g., use `%lf` for double, not just `%f`

- Skips "white space" except for `%c`, and returns number of values converted as `int`
Character functions <ctype.h>

• Alternatives to `scanf`, `printf` with "%c":
  ```c
  int c = getchar();
  putchar(c); /* one character at a time */
  ```

• Ways to make a copy of a character:
  - `tolower(c)`, `toupper(c)`

• Ways to ask about a character’s kind:
  - `islower(c)`, `isupper(c)`, `isalpha(c)`, `isdigit(c)`, `isxdigit(c)`, ...
How do we change the values of the variables?

• As we said earlier, we use variables to store intermediate results

• When intermediate results change, we change the values of the variables

• How do we change the values of the variables:
  – By assigning them new values
Assignment Statement

General form of the assignment statement:
identifier = expression;

= is the assignment operator
   It does not mean “equals” (but we say it like that)

x = 5; /*! means “assign 5 to x” */
   –Now 5 is stored in the memory location called x

y = x + 2; /*! assign (x + 2) to y */
   –The value stored in x is retrieved, 2 is added to it, and
   the result is stored in y

x = x + 2; /*! assign (x + 2) to x */
   –It’s okay! It doesn’t mean “x equals x+2”. Right?
Arithmetic Expressions

- Operators:
  - +, -, *, / add, subtract, multiply, divide
  - % modulus operator – remainder

- Parentheses:
  - ( ) means whatever is inside is evaluated first

- Unary operators:
  - ( -3 ) + 5
  - -3 + 5
  - 5 - -3

- Use math.h library for difficult calculations
  - E.g., sqrt(x), cos(x), ... (more later)
Analyzing an expression

\[
\frac{-b + \sqrt{b^2 - 4ac}}{2a}
\]
Precedence and associativity so far (will expand)

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parentheses: ( )</td>
<td>Innermost first</td>
</tr>
<tr>
<td>2</td>
<td>Unary operators:</td>
<td>Right to left</td>
</tr>
<tr>
<td></td>
<td>+  − (type)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Binary operators:</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td>*  /  %</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Binary operators:</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td>+  −</td>
<td></td>
</tr>
</tbody>
</table>
Math library, get to know it

• Math library: <math.h>

• Many mathematical functions ready to use
  – sqrt(x) – to find square root of x
  – pow(v, p) – finds $v^p$ (v to power of p)
  – exp(a) – for $e^a$ (e is natural logarithm base)
  – log(y) – for ln y (natural logarithm of y)
  – log10(b) – log_{10} b (base 10 logarithm of b)

• And many more – see pp. 61-62
  – Note for trig functions – pass radians, not degrees

#define PI 3.141593 /* symbolic constant (p. 34) */
angleRadians = angleDegrees * (PI / 180);
Symbolic constants

- **A symbolic constant** is a string that is replaced by the C preprocessor using *text substitution*

- The convention is to use capital letters in defining symbolic constants
  
  ```
  #define PI 3.141593
  ```

- C preprocessor (cpp) goes over the program and replaces the string “PI” with “3.141593”
A program to compute distance

- Program chapter1_1 (distance.c)

- Note: `<math.h>` included to use `sqrt`
  - So must explicitly link to math library
  - And, so must use gcc (or cc) directly:
    
```c
gcc -o pgm -lm pgm.c
```

- “-lm” means link with the math library

Calculate the straight-line distance between two points

![Diagram of two points (1,5) and (4,7) connected by a line representing the distance]
Structure of simple C programs we have seen

preprocessing directives /* include, define */

int main(void) {
    declaration1;
    declaration2;
    ...
    statement1;
    statement2;
    ...
    returnStatement;
}

Assignment with arithmetic

• Compound assignment operators
e.g., \( a += 5; \)
  /* same as: \( a = a + 5; \) */

– Also \( -=, *=, /=, \) and \( %= \)

• Special forms for \( += \) and \( -= \), called increment and
decrement operators, respectively
  – ++ increments by 1  (same as \( += 1 \))
  – -- decrements by 1  (same as \( -= 1 \))
  – e.g. \( \text{counter}++; \) /* same as \( \text{counter} = \text{counter} + 1; \) */
Pre/post versions of ++ and --

- Post-increment is not exactly the same as pre-increment (same goes for decrement)

- Post version changes after used in expression
  e.g., say \(x = 7\), then
  ```c
  printf("%i", x++);
  ```
  would print 7

- Pre version changes before using (say \(x = 7\) again)
  ```c
  printf("%i", ++x);
  ```
  would print 8.

- In either case, \(x\) equals 8 after the print.
Operator precedence update

1. ( )

1. ++, --, unary +, -

1. *, /, %

1. +, -

1. =, +=, -=, *=, /=, %=
Operations depend on types: \( a \ op \ b \)

- If \( a \) and \( b \) are same type: (1) perform \( op \) based on rules of that type, and (2) return result of that type

- If \( a \) and \( b \) are different types: (0) make a copy of “lower” value \textit{promoted} to “higher” type
  - Then (1), (2) as above; result is higher type

- Cast operator – force type of copied value
  
  ```
  double d = 2.5;
  int n = 10 / (int)d; /* Q. What is n? */
  ```
Technique: linear interpolation

- Text pp. 52-55 describes idea of linear interpolation
- Text program applies it to seawater temperatures...

\[
\frac{f(a) - f(b)}{b - a} = \frac{f(a) - f(c)}{c - a}
\]
Errors – 2 basic types

- Syntax errors – what you find out about first
  - Improperly formed (or typed) source code
    - e.g., `Int main` ← should be `int` (*case matters*)
    - e.g., `printf("Hello\n);` ← missing " at end of string
  - Compiler won’t compile the source code
    - Important to learn to *read the error messages*

- Logic errors – a.k.a., “bugs”
  - Compiler said it’s okay, but results are wrong
  - Often have to go back and fix algorithm in this case
  - Example Mars Climate Orbiter
    - [http://en.wikipedia.org/wiki/Mars_Climate_Orbiter](http://en.wikipedia.org/wiki/Mars_Climate_Orbiter)