2. Basic Program Structure
ENEE 140

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

Today’s Lecture

• Where we’ve been
  – Comments & documentation
  – First program in C
  – Requirements
  – Using Eclipse

• Where we’re going today
  – Variables
  – Constants
  – Arithmetic operations
  – while loops
  – Program design

• Where we’re going next
  – Character input/output
We’ve Seen: Requirements

Before you start programming, you must understand the requirements
(you must know what the program is supposed to do)

Program Design

• Write a program that counts the number of words from its input
  state <- not in a word
  While characters are available on the input
    Read character c
    If c is not whitespace
      If currently not in a word
        Increment word count
        state <- in a word
      Else (i.e. c is whitespace)
        state <- not in a word
  Loop
Input/output
  Branching on a condition
  Variable
This_is_a_sentence.
Elements of Program Structure

• Variables
  – Variables and constants (L2), enumerations (L7)

• Branching
  – If statement (L3), switch statement (L9), conditional assignment (L9)

• Loops
  – while (L2), for (L3), do-while (L9)

• Arithmetic operations
  – Integer and floating point operations (L2, L5), precision limits (L5)

• Data types
  – Primitive data types (L2, L3, L6), type conversions (L2, L6)
  – Binary representation (L5), bitwise operators (L5)
  – Composite data types: struct (L6, L11), union (L11)

Elements of Program Structure – cont’d

• Vector data types
  – Arrays and strings (L7)
  – Multi-dimensional arrays (L12)
  – Sorting (L13)

• Input/output
  – Reading from standard input and writing to standard output (L1, L3, L4), file input output (L10, L11)

• Writing complex programs
  – Support for modularity: functions (L4), splitting a program into multiple files (8), variable scope (L8)
  – Coding style (L5)
  – Defensive programming (L6)
  – Testing (L6)
Designing Programs

Before you start writing C code, write down the program design (e.g. the mechanical steps your program will follow)

Variables

• Correspond to memory locations that hold data and that may be manipulated in your program

• Must be declared:
  ```c
  int a;
  float b;
  ```
  integer variable  floating-point variable (has fractional part)

• Must be assigned a value
  ```c
  a = 1;
  b = 1.5;
  ```
  assignments change the value  stored in the variable

• May be used in expressions
  ```c
  a < 10
  b = a + 1;
  ```
  comparison test  value of arithmetic operation used in assignment
Assignment vs. Equality Testing

\[a = a + 1;\]  
assignment (increment a by 1)

\[a == a + 1\]  
equality testing (result is false)

Arithmetic Operations

+ - * /

- **Integer** arithmetic
  - **Division truncates**: the fractional part is discarded
    \[\text{int } a = 1 / 2;\]  
    value of a is 0

- **Floating-point** arithmetic
  - **Division does not truncate**
    \[\text{float } b = 1.0 / 2.0;\]  
    value of b is 0.5
Relational Operators

• Used for making comparisons

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Equal</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less Than or Equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater Than or Equal</td>
</tr>
</tbody>
</table>

• Work on both integers and floats

• Good programming practice: avoid (in)equality tests with floats!
  – Example:
    
    ```
    b != 0
    ```
    
    if b is a float, try to use <= or >= instead

  – Results of floating point operations are imprecise (more on this later)

Combining ints and floats in Expressions

• If an arithmetic operator has integer operands
  – Integer arithmetic is used
    ```
    int a = 1;
    int b = a / 2;
    ```
    value of b is 0

• If an arithmetic operator has at least one floating-point operand
  – Floating-point arithmetic is used
    ```
    float a = 1;
    float b = a / 2;
    ```
    value of b is 0.5

• Expression type is evaluated before assignment
  ```
  float b = 1 / 2;
  ```
  value of b is 0
  ```
  float b = 1.0 / 2.0;
  ```
  value of b is 0.5
Symbolic Constants

• Good programming practice: if you have constants in your program, give them a symbolic name

• Declaring constants
  – Modern constant declarations
    ```
    const float pi = 3.14159;
    ```
  – Old-school constant declarations (traditionally uppercase)
    ```
    #define PI 3.14159
    ```

• Using constants
  ```
  float radius = 1;
  float circumference = 2 * PI * radius;
  ```

while loops

• Repeating program statements while a condition holds
  ```
  while (condition) {
      condition is tested first
      ...
  }
  ```

• Example: print “Hello world” 10 times
  – You need a variable to count the number of iterations. Let’s call it i
    ```
    int i = 0; /* initialize i */
    while (i < 10) { /* iterate while i is less than 10 */
        printf("Hello World\n"); /* increment i */
        i = i + 1;
    }
    ```
Review of Lecture

• What did we learn?
  – Variables and constants
  – Arithmetic operations and comparisons
  – while loops

• Next lecture
  – Character Input/Output

• Assignments for this week
  – Review K&R 1.2 and make sure you understand how while loops and arithmetic operations work
  – Read K&R Chapters 1.3, 1.5, 2.1, 2.6, 3.1, 3.2
  – Weekly challenge: word_per_line.c
  – Homework: lab02.pdf, due on Friday at 11:59 pm
  – Quiz 2, due on Monday at 11:59 pm