CSE 105, Lab 7, Summer 2006
cse.unl.edu/∼cstrope/csce105su06/lab/lab7

Instructor: Cory Strope

August 2, 2006

1. Pointers
   - A pointer is a variable whose value is a memory address (location).
   - The format for a pointer in a printf statement is “%p”.
   - Addresses are displayed in hexadecimal (base 16, a.k.a: hex)
   - A 32 bit address bus requires 8 hex digits.
   - Addresses are preceded by the letters `0x` which indicates it is being displayed in hex.
2. Declaring Pointers
   - The program needs to know the data type that the pointer “points” to. This is indicated by use of * in the declaration.

     ```
     type *variableName;
     ```
   - Example: the pointer named `ptNumber` is to be declared as pointing to an `int`.

     ```
     int *ptNumber;
     ```

     This indicates that the content of the variable named `ptNumber` contains the address of an `int`.
   - Example: the pointer named `ptDecimal` is to be declared as pointing to a `double`.

     ```
     double *ptDecimal;
     ```

     This indicates that the content of the variable named `ptDecimal` contains the address of a `double`.

3. Getting the content pointed to by a pointer

    - To indicate that the content of the memory address contained in a pointer is wanted, * is once again used. It is used as a unary operator which returns the content of the memory location pointed to by the pointer following the *.
    - Example: this statement requests the integer value pointed at by pointer named `ptNumber` which was declared above.

      ```
      int int_value = *ptNumber;
      ```

      This returns the content of the variable named `ptNumber`, which is an integer.

4. Getting the address of a variable

    - To indicate that the memory address of a specific variable is wanted, the unary operator & is used. This operator returns the memory location of the variable following it.
    - Example: these statements declare the variables and requests the address of the variable `firstVariable`.

      ```
      double firstVariable;
      double *ptVariable;
      ptVariable = &firstVariable;
      ```

      This returns the address of the variable named `firstVariable`, and assigns it to the variable `ptVariable`! Compile source code `pointerDemo1.c` and run. Note the values printed to screen. Open the source code file and compare the values printed out at different times. Compile source code `pointerDemo2.c` and run. Note the values printed to screen. Open the source code file and compare the values printed out at different times. How is this program different from `pointerDemo1.c`?

5. Scope: How long does a variable have meaning? A very simplified explanation of “life-time” and visibility of a variable is as follows

- A variable lasts only as long as the block of code in which it was declared in is being executed. For instance, all variables declared within a user defined function exist only during the function call. They cease to exist after the `return` statement if the function (or after the last statement is executed if there is no `return` value).
- Variables declared inside a loop, cease to exist when loop is exited.
- Variables in the calling function are not visable to the called function. This is why you need to pass variables to them.
- Passing pointers to a function allows the function to alter variables back in the calling function. These are referred to as output parameters.
- Compile source code `scopeDemo3.c`. OOPS! There is a bug. What is it and why? Correct the bug, recompile, and run.

6. Practice problem:

- Write and turn in the change program from homework using only dollars and quarters (leave nickels, dimes, and pennies out).
- Write a program to model a simple calculator. Each data line should consist of the next operation to be performed from the list below and the right operand. Assume the left operand is the accumulator value (initial value of 0). You need a function `scan_data` with two output parameters that returns the operator and right operand scanned from a data line. You need a function `do_next_op` that performs the required operation. `do_next_op` has two input parameters (the operator and operand) and one input/output parameter (the accumulator). The valid operators are:
  - + add
  - - subtract
  - * multiply

- / divide
- ˆ power (raise left operand to the power of the right operand
- q quit

Your calculator should display the accumulator value after each operand. A sample run:

```
+ 5.0
result so far is 5.0
ˆ 2
result so far is 25.0
/ 2.0
result so far is 12.5
q 0
final result is 12.5
```