C Pointers

CMPU 224 – Computer Organization
Jason Waterman
Comments

• /* any text until */

• // C++-style comments

• Convention for longer comments:
  /*
   * AverageGrade()
   * Given an array of grades, compute the average.
   */
Data Definition

• A variable must be defined before use
  • It must be defined exactly once
  • Specify variable name and type

• Example:
  • `int x; // holds an integer`
  • `char str[6]; // holds an array of 6 characters`

• You can also initialize a variable when you define it
  • `int x = 42; // x holds the value of 42`
  • `char str[6] = {'h', 'e', 'l', 'l', 'o', '\0'};`
Data objects

• Variable is a named container that can hold a value
• Default value is (mostly) undefined – treat as random
  • compiler may warn you about uninitialized variables
• Every data object in C has
  • a name and data type (specified in definition)
  • an address (its location in memory)
  • a size (number of bytes of memory it occupies)
  • visibility (which parts of program can refer to it)
  • lifetime (period during which it exists)

• Assignment:
  • int x; // variable definition
  • x = 42; // assignment
  • The left hand side of the equal sign is always a variable name
  • The right hand side of the equal sign is always a value
Data objects and pointers

• What is a pointer?
  • A variable that holds the memory address of another variable
  • Since memory addresses are 8 bytes, all pointers are 8 bytes long, regardless of what type of data they point to
  • Also known as a reference
Why do we have pointers?

• It closely models how the hardware uses memory (as we will see)

• Efficiency
  • It allows data to be passed into a function without copying the data
  • You can pass in a reference (pointer) to the data instead of copying the data

• Example:
  • You have a very large (100k) array of integers (400kB)
  • To give a function access to that data, you can copy 400kB of data into the function or a single 8 byte pointer
Pointer operators

• & (address of operator)
  • Gives the address (where it lives in memory) of a variable
  • Putting an & in front of a variable gives you the address where that variable lives in memory
    • That value is a pointer (and can be assigned to a pointer)

• * (dereference operator)
  • Used when defining a pointer
    • int *ip; // type: int *
    • Defined just like any other variable
  • Also used when dereferencing a pointer
    • Gives the value at the memory location stored in the pointer variable
    • We’ll come back to this
#include <stdio.h>

int main(int argc, char *argv[]) {
    // declare int ival and int pointer
    // iptr. Assign address of ival to iptr.
    int ival = 1;
    int *iptr = &ival;

    // dereference iptr to get value pointed
    // to, ival, which is 1
    int get = *iptr;
    printf("*iptr = %d\n", get);

    // dereference iptr to set value pointed
    // to, changes ival to 2
    *iptr = 2;
    int set = *iptr;
    printf("*iptr = %d\n", set); /* 2 */
    printf("ival = %d\n", ival); /* 2 */
}