CS 24000 - Programming In C

Week 16: Review
(cont’d)

Zhiyuan Li
Department of Computer Science
Purdue University, USA
Scope of the Final Exam

• We have covered issues related to the address space and the relationship between memory addresses and pointers (including function pointers)

• The next issue in C language that is not present in Java-like languages is the direct manipulation of data bits
The Concept of Data Representation

• The address space is addressable in bytes
• C, unlike many other languages, allows the program to directly manipulate the individual bytes of a piece of data (e.g. integer, float)
• This makes C well suited for implementing low-level system operations
  – Data conversion between different systems (different endianess)
  – Control status (interrupt registers)
  – Extracting specific bits, e.g. part of a data packet in the network
• This also make C suitable for applications that must manipulate data at bit level, e.g. data compression, encryption
Number representation

• In order to extract bit-level information, we need to know how a number (int, float) is represented by bits
• Decimal to binary (or its compact representation by hexadecimal)
• Binary to decimal
• Unsigned int
• Signed int under 2’s complement representation
The Endianess

- It has two aspects
  - Byte organization of a number in the address space (how do we extract a certain byte)
  - How we “linearize” the data, i.e. when we print a number, which byte comes first

- In computer organization and networking courses, we will see endianess is an issue at the bit level also
  - When transmit a byte through the network, which bit goes first?
  - But in the address space or at the modern processor hardware level, bit-level endianess has no practical meaning
    - Because bits are accessed in parallel, unlike in some networks
• We often need to either truncate or to expand a piece of data (e.g. from byte to int and vice versa)
• Understanding data representation will enable us to do such conversion correctly
• Sign extension of signed numbers
  – Char to int, int to long long, e.g.
• Floating point number representation
  – 32-bit to 64-bit
Review of Quiz 12 #2

- What does the following program print? (assume 32 bit machine)
  - #include <stdio.h>
  - int main() {
    int x;
    char a = 0xaa, b = 0x11;
    a = b+a;
    x = a;
    printf ("%x", x);
  }

- The answer is ffffffbb , why? We go through the intermediate steps.
Declarations and Definitions

• This is related to the issue of memory allocation
• Definition binds a variable to a memory location and defines how it will be accessed
• Static, local, heap-allocated
• Function declarations versus definitions
• Global variables, external variables
Data representation versus interpretation

• There is a conceptual difference between how a piece of data is represented internally on computer hardware versus the data is interpreted
• Data interpretation is application dependent
• Do we view a word of four bytes as
  – Four ASCII characters?
  – Four char-sized integers?
  – A pointer
  – An integer
    • Signed?
    • Unsigned
  – A float number?
• The printing function (e.g. printf) simply displays it the way we want the data to be interpreted
Floating point number representation

- The IEEE standard
- Sign, exponent, mantissa
- Normalized form of the mantissa
- Bias of the exponent
- We don’t require to remember the special (non-normalized) forms
Parameter passing and function return value

• C function calls pass parameters by value
  – In order for a called function to modify variables that are in the scope of the calling function, the addresses of those variables must be passed as parameters
  – Passing a big structure by value is expensive, because the entire structure will be copied to the callee’s stack frame
  – Returning a big structure is also expensive, for the same reason
  – Passing the base address of the structure is more efficient
  – In Java, the reference to the structure (object) is passed as the parameter
Strings

• In C, strings are perhaps one of most error prone data structure
• The type of a string is an array of chars.
• Since the length of the string is often undetermined at compile time, the proper composition of a string must be null-terminated
• A quoted string is always automatically null terminated
• But if we explicitly pack a string to an array, we must remember to terminate it.
Syntax Rules

• We assume students are familiar with the basic syntax rules
• We will not pose questions involving obscure syntax rules, e.g. obscure expressions and obscure library function interfaces
• We assume students are familiar with the basic precedence ordering among operations and associativities
  – E.g. left associative arithmetic operations
  – E.g. postfix expressions having higher precedence than prefix expressions
Basic I/O functions

- We assume students are familiar with the basic concepts associated with file I/O
- Read, write, fread, fwrite, getchar, putchar, printf, scanf
- And their impact on seek positions
- Open, fopen, close, fclose, seek, fseek
Processes and IPC

• The impact of fork() calls on the semantics of the calling program
  – Impact on file I/O
  – Impact on variable values
  – etc
• Pipe calls and the proper use of pipes
• Shared memory and proper operations to allocate, attach, and remove shared memory blocks
• Semaphores and proper use of semaphore operations (sem_init, sem_wait, sem_post) and destroy operations
Summary

- The scope of final exam will basically be midterm 1 + midterm 2 + scope of the four projects