```
Handout for CS 372H
Lecture 3
26 January 2010

1. Fun with function pointers

    /* countas.c */

    #include <stdio.h>
    /*
     * define new type fptr: "pointer to a function whose input is a char* and
     * whose return is an int"
     */
    typedef int (*fptr)(char*);

    int return_0(char* p)
    {
        return 0;
    }

    int funfun(char* p)
    {
        /*
         * on the survey, it was:
         * int (*func_ptrs[])(char*) = {return_0, funfun};
        */

        fptr func_ptrs[] = {return_0, funfun};

        return (*p == 'a') + (*func_ptrs[*p != 0])(p+1);
    }

    int
    main(int argc, char** argv)
    {
        printf("%d\n", funfun(argv[1]));
        return 0;
    }

    NOTES

    --to compile this:

        % gcc −g −Wall −o countas countas.c

    --to get an assembly dump:

        % gcc −S countas.c

    --no conditional branches! (so no looping, if statements, etc.)

    --(*p != 0) gets compiled into:
            testb   %al, %al
            setne   %al
            movzbl  %al, %eax

    --then %eax is used to help index into the table of function
    pointers

    --common design pattern: use a value to index into a table of
    function pointers. (the table is a list of addresses.)

    --even happens at hardware/software interface. for example, when an
    interrupt happens, the x86 chooses which interrupt handler to
    execute by using the value of the interrupt as an index into a table
    of interrupt handlers.
```



```
2. Example for gcc calling conventions

    Here is the C code:
        int main(void) { return f(8)+1; }
        int f(int x) { return g(x); }
        int g(int x) { return x+3; }

    The assembly code:

        _main:
                    prologue

            pushl %ebp
            movl %esp, %ebp

                    body
            pushl $8
            call _f
            addl $1, %eax

                    epilogue
            movl %ebp, %esp
            popl %ebp
            ret

        _f:
                    prologue

            pushl %ebp
            movl %esp, %ebp

                    body
            pushl 8(%esp)
            call _g

                    epilogue
            movl %ebp, %esp
            popl %ebp
            ret

    <small version of _g>:
            movl 4(%esp), %eax
            addl $3, %eax
            ret

    <longer version of _g>:
                    prologue
            pushl %ebp
            movl %esp, %ebp

                    save %ebx
            pushl %ebx

                    body
            movl 8(%ebp), %ebx
            addl $3, %ebx
            movl %ebx, %eax

                    restore %ebx
            popl %ebx

                    epilogue
            movl %ebp, %esp
            popl %ebp
            ret
```

```
3. pseudocode for Bochs CPU simulation

    for (;;) {
        read_instruction();
        switch (decode_instruction_opcode()) {
        case OPCODE_ADD:
            int src = decode_src_reg();
            int dst = decode_dst_reg();
            regs[dst] = regs[dst] + regs[src];
            break;
        case OPCODE_SUB:
            int src = decode_src_reg();
            int dst = decode_dst_reg();
            regs[dst] = regs[dst] − regs[src];
            break;
        ...
        }
        eip += instruction_length;
    }

4. simulate PC's physical memory map

    #define KB      1024
    #define MB      1024*1024

    #define LOW_MEMORY  640*KB
    #define EXT_MEMORY  10*MB

    uint8_t low_mem[LOW_MEMORY];
    uint8_t ext_mem[EXT_MEMORY];
    uint8_t bios_rom[64*KB];

    uint8_t read_byte(uint32_t phys_addr) {
        if (phys_addr < LOW_MEMORY)
            return low_mem[phys_addr];
        else if (phys_addr >= 960*KB && phys_addr < 1*MB)
            return bios_rom[phys_addr − 960*KB];
        else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
            return ext_mem[phys_addr−1*MB];
        else ...
    }

    void write_byte(uint32_t phys_addr, uint8_t val) {
        if (phys_addr < LOW_MEMORY)
            low_mem[phys_addr] = val;
        else if (phys_addr >= 960*KB && phys_addr < 1*MB)
            ; /* ignore attempted write to ROM! */
        else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
            ext_mem[phys_addr−1*MB] = val;
        else ...
    }

[Credit to Frans Kaashoek, Robert Morris, and Nickolai Zeldovich for 2−4
above]
```