NAME:

Login name:

Computer Science 217
Final Exam
May 15, 2009
1:30pm-4:30pm

This test has eight (8) questions and thirteen (13) pages. Put your name (or login-id) on ***every page***, and write out and sign the Honor Code pledge before turning in the test.

``I pledge my honor that I have not violated the Honor Code during this examination."

Question	Score
1 (8 pts)	
2 (10 pts)	
3 (12 pts)	
4 (20 pts)	
5 (10 pts)	
6 (10 pts)	
7 (10 pts)	
8 (20 pts)	
Total	

QUESTION 1 Building a Program *(8 POINTS, 1 point each)*

This question concerns the steps involved in creating a binary executable **`a.out`** for the following program:

```
#include <stdio.h>

/* This is my cool program */
int main(void) {
  int i;

  scanf("%d", &i);
  while (i--)
    printf("i=%d\n", i);
  return 0;
}
```

The four main stages of creating **`a.out`** are pre-processing, compiling, assembling, and linking. For each operation, circle which component performs the task, where **P** stands for pre-processor, **C** for compiler, **A** for assembler, and **L** for linker.

P C A L Inserts the contents of **`/usr/include/stdio.h`**

P C A L Includes the code that implements **`scanf()`**

P C A L Ensures that the first argument of **`printf()`** is of type "**`char *`**"

P C A L Removes the comment "**`This is my cool program`**"

P C A L Checks that "**`return 0`**" returns an integer

P C A L Translates '**`i--`**' into the "**`decl`**" instruction

P C A L Determines the argument for the "**`jmp`**" instruction to determine how far to jump to get the start of the while loop

P C A L Determines the address to call to invoke the **`scanf()`** function

QUESTION 2: Memory Management *(10 POINTS)*

2a) Consider a computer system that has virtual memory with ***32-bit*** addresses and a ***16 KB*** page size. How many bits are used to identify the ***byte offset*** in a page? How many ***virtual pages*** can a process have? *(2 points)*

2b) Which component is responsible for the following tasks, the underlying hardware (H) or the operating system (O)? Please circle either "H" or "O" for each item. *(4 points)*

H O Mapping virtual address into a physical address for pages already in physical memory

H O Deciding which virtual page to "swap out" of physical memory on a "miss"

H O Updating the page tables with new virtual-to-physical page mappings

H O Preventing one user process from accessing the pages of another user process

2c) For caching in a memory hierarchy, what is the motivation for a *larger* cache block size? Please check one answer. *(1 point)*

___ Temporal locality

___ Spatial locality

2d) For caching disk pages in main memory, what overhead is amortized by using *large pages*? Please check one answer. *(1 point)*

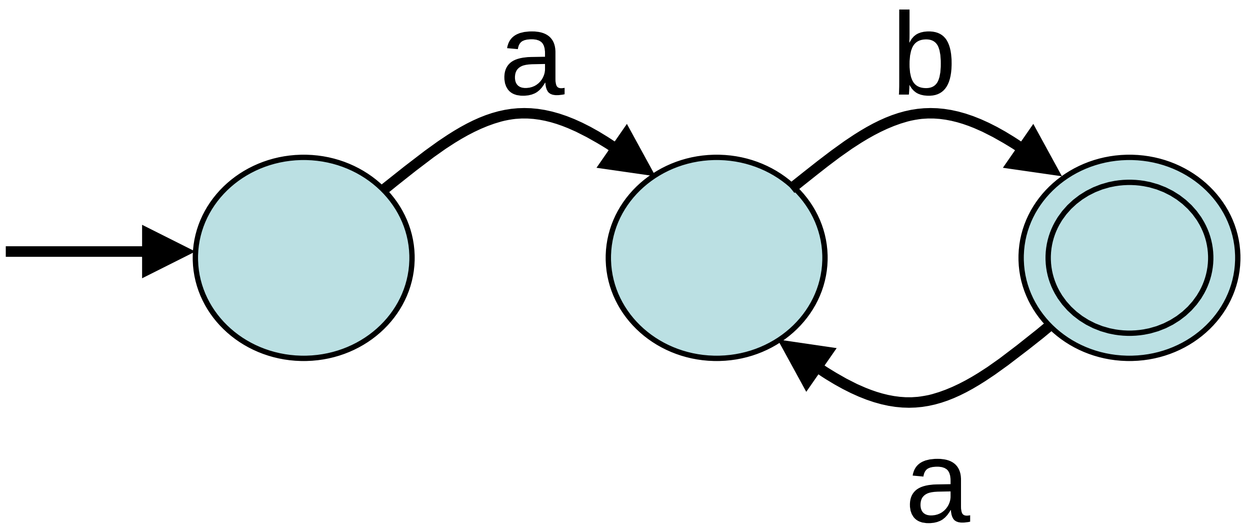
___ Slow disk seek time

___ Low disk throughput

2e) Why does implementing **`malloc()`** and **`free()`** with a single free list (with free blocks of different sizes) lead to a lot of virtual-memory *page faults*? *(2 points)*

QUESTION 3: Deterministic Finite Automata *(12 POINTS)*

In this question, you will draw a deterministic finite automata (DFA) that accepts particular strings. Please draw your DFA diagrams with a start state (marked with an incoming arrow) and success states (circled twice). As an example, consider a DFA that reports "success" for an input that repeats the characters "ab" one or more times and reports "failure" otherwise. The inputs "ab", "abab", and "ababab" would lead to success, whereas "a", "aba", "abc", "789", or "cabab" would not. That DFA would be drawn as:



Draw a DFA that accepts only strings of a's and b's that have an even number of each character. For example, the DFA would accept the empty string, "aa", "bb", "aabb", "abab", and "ababbb", but not "aab" or "aaaba". Ensure that your DFA has the *minimum* possible number of states.

QUESTION 4: Reduced Instructions for Silly C *(20 POINTS)*

The instruction set of a computer defines the basic operations it can perform, and all other operations must be performed by combining these operations together. This question explores the challenges of "making do" with an instruction set with less functionality. Rather than writing assembly-language programs, we envision a reduced version of C with limited functionality, and build functions that implement the missing operations.

4a) Suppose C does not support the multiplication operation (e.g., "4 * 5"). Write a function **`multiply()`** that performs multiplication of two unsigned integers, and returns the unsigned integer result. Do not use multiplication, division, bit-wise operations, or any functions from libraries. Make your code as *short* as possible. *(6 points)*

4b) Suppose C does not have a left shift operator (e.g., "k << 3"). Write a function **`lefty()`** that performs left shift on an unsigned integer, shifting an unsigned integer number of times. Do not use multiplication, division, or bit-wise operations, or any functions from libraries. Do *not* use the **`multiply()`** function from question **4a**. Make your code as *short* as possible. *(6 points)*

4c) Suppose C only supports one size of unsigned integers (**`unsigned`**) but you want to perform arithmetic on larger integers. Define a **`big_unsigned`** data structure that consists of two unsigned integers corresponding to the upper and lower bits of a **`big_unsigned`** number. Implement a function that performs addition on two **`big_unsigned`** numbers and returns the **`big_unsigned`** result. Use only comparison operations and unsigned arithmetic. (*8 points*)

QUESTION 5: Process Control ***(10 POINTS)***

5a) Consider the similarities and differences between *calling a function* and *performing a context switch*. Circle whether the following statements are true for a function call (F), a context switch (C), or both (B). (*3 points*)

F C B Values stored in registers are saved so they can be restored later

F C B Control of the computer transfers to the operating system

F C B The instruction pointer (EIP) changes to execute instructions in a new location

5b) A call to **`fork()`** creates a child process that inherits a copy of the parent's virtual address space. The virtual address space is quite large, so copying every byte would be quite time consuming. How is this overhead avoided? (*3 points*)

5c) Give *two* examples of why a process might leave the "running" state. (*2 points*)

5d) Suppose a user types

```
echo foobar | wc –l
```

at the UNIX prompt. What are ***stdin*** and ***stdout*** for the **`echo`** and **`wc`** processes? (*2 points*)

QUESTION 6: Review ***(10 POINTS)***

6a) If the character '/0' can be used to signify the end of a *string*, why can't it be used as a way to signify the end of a *file* (instead of using the integer EOF)? *(2 points)*

6b) Consider the following code, where **`i`** and **`j`** are unsigned integers:

```
i = 7;
printf("%d\n", (i=j) ? 0 : j);
```

where **`j`** has already been assigned a value. Give a concise description of what the code prints to standard output. (*2 points*)

6c) Consider the following operations in C where **`i`** and **`k`** are unsigned integers:

```
i = ((((k/4) * 4) >> 2) << 2);
```

Rewrite to compute the same result as succinctly as possible. *(2 points)*

6d) Give an example of a programming error that is caught by the *preprocessor*, and another that is caught by the *compiler*. (*2 points*)

6e) What does

```
printf("%x %x\n", 0xfad – 0xcab, 0xfad & 0xdff);
```

print to standard output**?** *(2 points)*

QUESTION 7: Reading Fork Code ***(10 POINTS)***

This question concerns the following program:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

void cos217(void) {
  pid_t pid;

  if (!(pid = fork())) {
    fork();
    fork();
    fork();
    printf("cos217\n");
  }
}

int main(void) {
  cos217();
  printf("cos217\n");
  return 0;
}
```

7a) How many times does this program print “cos217”? (*4 points*)

7b) When running the program, sometimes parts of the output look like this:

cos217cos217

cos217

Why does this happen, and how can it be prevented? (*6 points*)

QUESTION 8: Code Reading, and Writing *(20 POINTS)*

8a) Consider the following code:

```
…
double* x;

x = (double *) malloc(sizeof(double*));
&x = 3.145;
…
```

Identify *two* bugs in the code, and rewrite the code to be correct. (*2 points*)

8b) Consider the following code:

```
char a[10][20][30];
int i, j, k, sum = 0;

…
for (j=0; j<20; j++)
  for (i=0; i<10; i++)
    for (k=0; k < strlen(a[i][j]); k++)
      sum += (a[i][j][k] == 'J');
```

Give a high-level description of what this code computes (i.e., what does "`sum`" store at the end)? What are *two* reasons why this code runs slowly? Rewrite the code to run faster. (*4 points*)

8c) Consider the following code for swapping two numbers

```
void swap(int i, int j) {
  int t;

  t = i;
  i = j;
  j = t;
}

int main(void) {
  int a = 5, b = 10;
  printf(”a=%d, b=%d\n”, a, b);
  swap(a,b);
  printf(“a=%d, b=%d\n”, a, b);
}
```

What does the program produce as output? What is the bug? Rewrite the code with the bug fixed. Show the modification of **`main()`** to call your new version of **`swap()`**. (*3 points*)

8d) This question follows up on question **8c**. Rewrite the swap code to be "generic" (i.e., to work on any type of input data), and show the necessary code to call your new version of **`swap()`** to swap two integers. (*6 points*)

8e) Consider the following function **`foo()`**:

```
int foo(unsigned num){
  int i;

  for (i = 0; num; i++)
    num &= (num-1);

  return i;
}
```

State concisely what function **`foo()`** returns. Do *not* describe how the function works, just what it computes – your answer should be no more than 10 words long. (*5 points*)