Name:

COS 217 Midterm
Spring 2006
Mar 15, 2006

Pledge:

Directions:

- Please answer each question in the space provided. The amount of space should be sufficient for a correct answer. If you need more space, please use the backs of pages, and make a note to that effect. If you run out of space, exam books are provided at the front of the room.
- This exam is open-book, open-notes, and is covered by the Honor Code. Please write and sign the pledge after you finish your exam.
- There are a total of five sections, with the number of points for each shown by the question. While it is not the intent for the exam to be a race, spending too much time on a single question may preclude finishing the exam. Budget your time wisely.
- To be fair, I will try to avoid answering content-related questions during the exam, unless it's to correct a mistake on my part.
- If you feel that a question **requires** additional assumptions or information to answer, please state them. Your guiding principle should be *Occam's razor*, which loosely translated states that you should allow as few assumptions as necessary to explain the situation.
- Answers should assume C/Unix unless otherwise stated or implied
- Please first read over the entire exam and then begin to answer questions. I will wait outside the exam room for the first five minutes, and then will be available in my office (room 322).
- Please write legibly

#	Name	Points Available	Score
1	True or False	10	
2	Short answer	15	
3	Code reading	20	
4	Code writing	30	
5	Bug hunt	25	
	Total	100	

1. True or False (10pts) For each statement, write "true" if the statement evaluates to true, or "false" if the statement evaluates false. If you believe the statement does not have a clear answer, give whichever choice is more appropriate and explain why.

- 1 & 2 & 3 & 4
- 0x2B | | (!0x2B)
- 0100 – 64
- 3 >> 2
- 3 < 2

2. Short Answer (15pts) Answer each item *well* in no more than 3 sentences.

   - What is a memory leak?

   - What does this mean: int (*blah)(const void *, const void *);

   - What does this mean: float *blah[25][25];

- What does the following code segment do:
  ```
  char *p = NULL;
  *p = 1;
  ```

- Why might a compiler legitimately place data in the text segment?

3. Code reading (20pts)

Consider the following piece of code:

```
#include <stdio.h>

static void f(unsigned int a)
{
  if (a & (~7))
    f(a>>3);
  putchar('0' + (a & 7));
}

int main(int argc, char *argv[])
{
  unsigned int a = 225;
  f(a);
  putchar('\n');
  return(0);
}
```

- What is the output of the above program?

- Give a one-sentence explanation of what the program does for different values to the function f.

- Write a simpler implementation of function f that achieves the same result

- If you inserted `printf("%d\n", a);` before the return in main, what would it print?

4. Code Writing (30pts)

You are given the task of writing a program that reads lowercase words from the input, and which outputs a list of the different words and their counts. So, if the input was
`happy happy joy joy`
your program would output

```
2 happy
2 joy
```

or

```
2 joy
2 happy
```

and `Paris in the the spring` would produce

```
1 Paris
1 in
2 the
1 spring
```

or any permutation of the 4 lines.

Devise a program that is correct, concise, reasonably designed, and reasonably efficient in both memory consumption and running time. You may assume an input of no more than 2 billion total words, and at most roughly 1 million distinct words. You are given two functions which you can use freely:

- `unsigned int Hash(char *str)` – given an input string, produces a hash value in the range of 0 to 2^{31}-1, which is roughly 2 billion.
- `char *ReadWord(void)` – returns a pointer to the next word, or NULL if no more input exists. The pointer points to a NUL-terminated string, but no other guarantees are provided.

Hint: the next power of 2 above 1 million is 1048576, which is 1024*1024 this page left intentionally blank to answer the previous question

5. Bug hunt! (25pts)

The following code is supposed to read as much data as possible from the input, store it into a giant buffer, and then print it all out. However, it's full of bugs. Identify and briefly explain as many bugs, possible bugs, and design flaws as you can. Suggest fixes where appropriate. Five right gets full credit.

```
int main(int argc, char *argv[])
{
  int used;
  int alloc = 2;
  char *buf;     /* the giant buffer */
  char *str;     /* the new line */

  /* read until we run out of input */
  while (scanf("%s\n", str) == 1) {
    int i;

    /* grow the buffer if needed */
    if (used + strlen(str) > alloc) {
      alloc *= 2;
      buf = realloc(buf, alloc)
    }

    /* add the new line to end of buffer */
    for (i = 0; i < strlen(str); i++, used++)
      buf[used] = str[i];

  /* print out the buffer, and we're done */
  printf(buf);
  return(0);
}
```

this page left intentionally blank to answer the previous question