Princeton University
COS 217: Introduction to Programming Systems
Spring 2006 Midterm Exam Answers

The exam was a 50 minute open-book open-notes exam.

Question 1

(a) FALSE - this is bitwise AND, so 1 & 2 yields zero. The rest of the statement does not make any difference.

(b) TRUE - this is logical OR, so as soon as you see that the first expression is nonzero, you can short-circuit it. This also answers the eternal question "to be or not to be" - the answer is TRUE.

(c) FALSE - the first number is 64 in octal, as per C. For those of you who assumed binary and said FALSE because you assumed negative numbers yield FALSE, I guess sometimes two wrongs do make a right.

(d) FALSE - shifting the value 3 right by two bits always yields zero. If the value were -3, then the result is not specified by the spec, since the high-order bits can be zero or one.

(e) FALSE - this is just the standard arithmetic comparison

Question 2

(a) Your program allocates memory and then loses the reference to it or forgets to free it after it's no longer in use. In general, this is a problem when it's in a loop or something similar, so that lots of memory gets leaked.

(b) blah is a pointer to a function that takes two const void pointers as arguments, and returns an integer.

(c) blah is a two-dimensional array of pointers to floats.

(d) It's an attempt to deref a NULL pointer. It seg faults.

(e) Because the data is constant, such as a string used to initialize a character pointer or character array.

Question 3

(a) 341 - lots of people read 225 as 255 and said 377.

(b) Prints the given value in octal.

(c) printf("%o", a);
Alternatively, you could replace the bitwise operations with div and mod, and replace the first test with just (a > 7).
Alternatively, you could do some combination of for loops that prints three bits at a time, starting with the leftmost bits.

(d) 225 - this is all call-by-value. It remains unchanged.

Question 4

I saw two right answers to this question. The more common one used a hash table where each bin pointed to a linked list of entries. The less common one simply used an array of entries, and collisions were handled by searching for the next available spot. In either case, this problem was made simpler than the programming project by noting that the hash table size could be fixed, and that no freeing of resources is required.

```
typedef struct Entry {
      char *e_word;
      int e_count;
```

```
      struct Entry *e_next;
} Entry;

#define NUMBINS (1024*1024)
#define MASK (NUMBINS-1)

Entry *bins[NUMBINS];

int main(int argc, char *argv[])
{
   char *word;
   Entry *walk;
   int i;
   while ((word = ReadWord()) != NULL) {
      i = Hash(word) & MASK;
      for (walk = bins[i]; walk != NULL; walk = walk->e_next) {
         if (strcmp(walk->e_word, word) == 0)
            break;
      }
      if (walk == NULL) {
         if ((walk = calloc(1, sizeof(Entry))) == NULL ||
             (walk->e_word = strdup(word)) == NULL)
            assert(FALSE);
         walk->e_next = bins[i];
         bins[i] = walk;
      }
      walk->e_count++;
   }
   for (i = 0; i < NUMBINS; i++) {
      for (walk = bins[i]; walk != NULL; walk = walk->e_next) {
         printf("%d %s\n", walk->e_count, walk->e_word);
      }
   }
   return(0);
}
```

Question 5

1. Missing semicolon after buf = realloc(buf, alloc).
2. Missing closing brace for while.
3. Should set used = 0.
4. Should set buf = NULL.
5. Allocate some space for str, like char str[1024].
6. This scanf is dangerous - fgets is a better choice. Even doing it letter-by-letter is a better choice. Changing the end condition to == EOF is not a great idea unless you know this will never generate a return value of zero.
7. The if test for growing the buffer is unsafe and should be while. Otherwise, doubling the buffer size just once can be a problem if the first few strings are really long.
8. Growing the buffer should account for the NUL-termination.
9. Calling strlen for every character is unnecessary - call it once before doing the copying. Alternatively, just use the memcpy function or, if you know that the string is properly terminated, even strcpy is fine.
10. Add NULL-termination after exiting while loop
11. The printf call is unsafe, especially if there's a % in the input. Replace with printf ''%s'',buf;
12. Sizes are declared as int, instead of size_t.