Princeton University
COS 217 Spring 2015 Midterm Exam

Name (please print clearly):

Login Id (please print clearly):

Precept (circle one): 1 (MW 1:30) 2 (MW 3:30) 3 (TTh 12:30) 4 (TTh 1:30)
5 (TTh 3:30) 6 (TTh 7:30)

Pledge and Signature:

The exam is closed-book and closed-notes. Use of electronic devices is prohibited.

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 extra pages provided at the end, and make notes to tell your grader that you have done so.

The exam is covered by the Honor Code. Write and sign the pledge after you finish your exam. The pledge is "I pledge my honor that I have not violated the honor code during this examination."

This is a 50 minutes exam. Each minute is worth roughly 2 points. Budget your time wisely.

If you feel that a question requires additional assumptions or information to answer, then state them. Your guiding principle should be Occam's razor, which loosely translated states that you should make as few assumptions as necessary to explain the situation.

All questions are in the context of the nobel/Linux/C/gcc217 computing environment unless otherwise stated or implied.

For instructor use only:

Question	Max Points	Points
1 Expressions in C	12	
2 Matching Terms	10	
3 Short Answer	30	
4 Bug Hunt	20	
5 Abstract Data Type	28	
TOTAL	100	

This page is intentionally left blank.

Question 1 (12 points) Expressions in C

Indicate whether each of these expressions evaluates to true or false. No explanation is needed.

(1a) `~1 && 1`

(1b) `512 - 01000`

(1c) `0x2B | (! 0x2B)`

(1d) `16 >> 4`

(1e) `sizeof(5) > sizeof(2L)`

(1f) `-10 < i < -1`

Question 2 (10 points) Matching Terms

Match the terms on the left with the phrases on the right. Each phrase can be used more than once. Pick the best match, or put "N/A" if none of the phrases is applicable.

Term	Phrase
_____ Pass by value	(a) Failing to free some chunk of allocated memory
_____ Pass by reference	(b) Validating parameters using assert macro
_____ Modulo arithmetic	(c) Pointer to a struct
_____ Dangling pointer	(d) When memory allocation fails
_____ Memory leak	(e) Class of abstract objects and their operations
_____ External testing	(f) Arithmetic that never results in overflow
_____ Regression testing	(g) Displaying outputs by writing to a log file
_____ Debugging heuristic	(h) Pointer to memory that has been free'ed before
_____ Modularity heuristic	(i) Used to send data out of a function
_____ Abstract data type	(j) Arithmetic with fixed number of bits
	(k) Used to send data into a function
	(l) Re-running all test cases after fixing a bug
	(m) Data encapsulation
	(n) Creating test inputs to execute all statements

Question 3 (30 points) Short Answer

(3a) (5 points) What is the binary representation of -30 on a system that uses two's complement representation and has an 8-bit word size.

(3b) (5 points) What does the following statement print to stdout?

```
printf("%d", (12 ^ 34) ^ 34);
```

(3c) (5 points) What value (in decimal) does the following code segment assign to *p?

```
char *p = malloc(1);
*p = 0x010A;
```

(3d) (5 points) The following function computes a hash function for strings (similar to what we discussed in lectures and precepts). Assuming that the input argument is valid, and without changing what the function computes, show how you can change the code to improve its performance. (*Hint*: The arithmetic part is fine.)

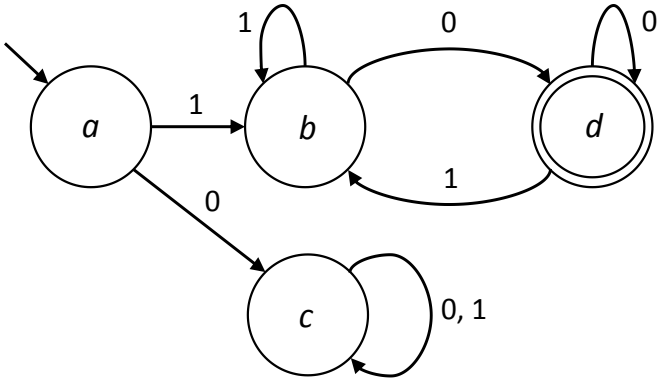
You can change the code in place or rewrite the function.

```
unsigned int hash(const char *s)
{
   int i;
   unsigned int h = 0U;

   for (i=0; i < strlen(s); i++)
      h = h * 65599U + (unsigned int) s[i];

   return h;
}
```

(3e) (5 points) Consider the DFA shown below on 0/1 inputs, i.e., it accepts or rejects binary words. The start state is the state labeled *a*, and the only accepting state is the state labeled *d* (shown as a double circle). State in one sentence the set of words accepted by the DFA. (*Hint:* Try out some examples of words that are accepted and words that are rejected.)



(3f) (5 points) Suppose all input words to the DFA above (in question 3e) are 8-bits long. If you interpret them as signed numbers in two's complement representation (on an 8-bit word system), with the first input being the most significant bit, state in one sentence the set of numbers accepted by the DFA.

Question 4 (20 points) Bug Hunt

The code shown below checks whether two words read from stdin are anagrams, i.e., whether one is formed by re-arrangement of letters of the other. For example, “secure” and “rescue” are anagrams. However, the code has many serious bugs. Identify and fix the program so that it works. Do not mention style errors, lack of comments, missing asserts, or lack of necessary “include” files. No explanations are required. Finding and fixing 5 bugs will get you full credit. (If two bugs appear on the same line, they will count as only one.) We will examine only the first 6 bugs listed.

You can assume that each word has a maximum of 50 letters and contains only lower-case letters. There is no need to check this. The line numbers on the left are for reference in your answers (not part of the code).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum {MAX_WORD_LENGTH = 50}
5  int check_anagram(char *s1, char *s2);
6
7  int main(void)
8  {
9    char str1[MAX_WORD_LENGTH], str2[MAX_WORD_LENGTH];
10    int result;
11   if (scanf(“%c %c”, str1, str2)!= 2)
12   {
13      fprintf(stderr, “Bad data\n”);
14      return EXIT_FAILURE;
15   }
16   result = check_anagram(str1, str2);
17   if (result)
18      printf(“strings are anagrams\n”);
19   else
20      printf(“strings are not anagrams\n”);
21   return 0;
22 }
23
24 /* returns 1 if s1 and s2 are anagrams, 0 otherwise */
25 int check_anagram(char *s1, char *s2)
26 {
27    int freq1[26], freq2[26], i, result;
28    for (i=0; i<26; i++)
29    {
30       freq1[i] = 0;
31       freq2[i] = 0;
32    }
33    for (i=0; s1[i] != '\0'; i++)
34       freq1[s1[i] - 'a']++;
35    for (i=0; s2[i] != '\0'; i++)
36       freq2[s1[i] - 'a']++;
37
```

```
38      result = 1;
39      for (i = 0; i < MAX_WORD_LENGTH; i++)
40         if (freq1[i] == freq2[i])
41            result = 0;
42      return result;
43   }
```

Bug 1 line number:

Fixed statement:

Bug 2 line number:

Fixed statement:

Bug 3 line number:

Fixed statement:

Bug 4 line number:

Fixed statement:

Bug 5 line number:

Fixed statement:

Bug 6 line number:

Fixed statement:

Question 5 (28 points) Abstract Data Type

A bag is an abstract data type used to represent a collection of items. A bag is an unordered collection, with no restrictions on the types of the items it may contain. The Bag ADT shown below provides a simple interface to clients. It includes operations to create a new bag, free a bag, check if a bag is empty, count the total number of all items in a bag, add an item to a bag, get an item from a bag, remove an item from a bag, and count the number of times a given item appears in a bag.

The bag.h file specifies the interface and the bag.c file has the code for implementing a bag using a linked list, where "psFirst" points to the first element in the list. Parts of bag.h and bag.c are shown below. (Implementations of most of these operations are not needed to answer the questions.)

bag.h

```
#ifndef BAG_INCLUDED
#define BAG_INCLUDED

typedef struct Bag *Bag_T;

Bag_T Bag_new(void);
void Bag_free(Bag_T oBag);
int Bag_empty(Bag_T oBag);
int Bag_count_all_items(Bag_T oBag);
int Bag_add_item(Bag_T oBag, const void *pvItem);
void *Bag_remove_item(Bag_T oBag, const void *pvItem,
      int (*pfCompare)(const void *pvItem1, const void *pvItem2));
int Bag_count_item(Bag_T oBag, const void *pvItem,
      int (*pfCompare)(const void *pvItem1, const void *pvItem2));
...

#endif
```

bag.c

```
#include <stdlib.h>
#include <assert.h>
#include "bag.h"

struct BagNode
{
  const void *pvItem;
  struct BagNode *psNext;
};

struct Bag
{
  struct BagNode *psFirst;
};
```

```
Bag_T Bag_new(void) {
  Bag_T oBag = (Bag_T) malloc(sizeof(*oBag));
  if (oBag == NULL)
     return NULL;
  oBag->psFirst = NULL;
  return oBag;
}
...
```

(5a) (4 points) Why is the type `struct Bag` declared in bag.c rather than in bag.h? List two benefits of declaring it in bag.c.

(5b) (4 points) Why is `pvItem` defined as `const void *` instead of a specific type like `const int *` ? Discuss one benefit each of using `const` and `void *` here.

(5c) (4 points) Write the code for the function `int Bag_empty(Bag_T oBag)`. The function returns 1 if the bag is empty and 0 otherwise. Make sure to check that the input parameter is valid. Your code should be correct and well-styled.

(5d) (4 points) The `Bag_add_item` operation does not check for duplicate items, i.e., an item is allowed to be added multiple times to a bag. Can you tell this from the given function declaration without knowing its code? Explain. (You do not have to provide its code.)

(5e) (12 points) Write the code for the function

```
int Bag_count_item(Bag_T oBag, const void *pvItem,
  int(*pfCompare)(const void *pvItem1, const void *pvItem2)).
```

The function `Bag_count_item` returns the number of times the item pvItem appears in the bag oBag by using the pointer pfCompare to a comparison function. Specifically, it returns 0 if the item is not in the bag at all. You can assume that *pfCompare function returns 0 to indicate equality.

Again, make sure to check that the input parameters to `Bag_count_item` are valid. Your code should be correct and well-styled.

Extra Page

Extra Page