

NAME:

Login name:

Computer Science 217
Midterm Exam
March 12, 2008
10am-10:50am

This test has six (6) questions. You should spend no more than 8-10 minutes per question for the 50-minute exam. Put your name 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."`

<u>Question</u>	<u>Score</u>
1 (20 pts)	
2 (20 pts)	
3 (15 pts)	
4 (15 pts)	
5 (25 pts)	
6 (5 pts)	
Total	

QUESTION 1: Integer Arithmetic and Character Output (20 POINTS)

1a) How is the decimal number 38 represented as an eight-bit binary number? What is the one's complement? The two's complement? (3 points)

1b) What does

```
printf("%d,%d,%d,%d,%d\n", 74/4, 74%4, 74&&4, 74&4, 74&3);
```

print to standard output? (5 points)

1c) Consider the following code, where k is an unsigned int:

```
printf("%u\n", k - ((k >> 2) << 2));
```

What does the code do? Rewrite the line of code in a more efficient way. (5 points)

1d) What string does the function $f()$ below return? Explain your answer. (7 points)

```
char* f(unsigned int n){
    int i, numbits = sizeof(unsigned int) * 8;
    char* ret = (char *) malloc(numbits + 1);

    for (i=numbits-1; i>=0; i--, n>>=1)
        ret[i] = '0' + (n & 1);
    ret[numbits] = '\0';
    return ret;
}
```

QUESTION 2: Arrays, Pointers, and Strings (20 POINTS)

Consider the following function that converts an integer to a string, where the `printf()` function “prints” to a formatted string, e.g., `printf(retbuf, “d”, 72)` places the string “72” starting at the location in memory indicated by the address `retbuf`:

```
char *ittoa(int n) {  
    char retbuf[5];  
    printf(retbuf, “%d”, n);  
    return retbuf;  
}
```

2a) Identify two serious bugs in this function. (6 points)

2b) Rewrite the function to fix these bugs. In your solution, allocate memory *dynamically* to store the string, using only the necessary space. E.g., you should allocate less memory for the integer 27 and more for 239592895. (Feel free to use the back of the page.) (14 points)

QUESTION 3: Short Answer (15 POINTS, 3 points each)

3a) When using an Abstract Data Type (ADT), the client program does not know how the underlying data structure. Provide two reasons for hiding this information from the client. In C, how does the programmer implementing the ADT hide this information from the client?

3b) A function main can include arguments `argc` and `argv[]`. If you run the program as “`a.out cs217 rules`”, what is stored in `argv[0][2]`? What is stored in `argv[2][0]`?

3c) Why is it inappropriate to perform pointer arithmetic (e.g., “`p++`”) on a void pointer?

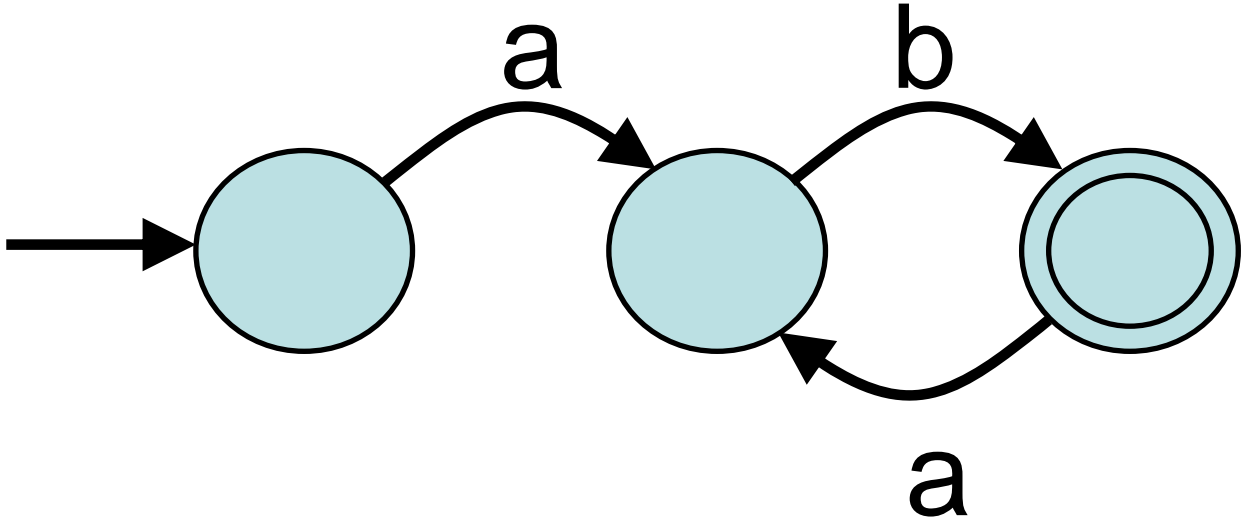
3c) Suppose a global variable is defined like: “`char* myname = "jrex";`” In what region of memory are the characters in “`jrex`” stored? How many bytes are allocated?

3d) In the example in question 3c, in what region of memory is the pointer variable `myname` stored? Can the program assign `myname` to a new value later? (For example, suppose there is another global variable defined as “`char* yourname= "dondero";`”. Can the `main()` function contain the line “`myname = yourname;`”?)

3e) What output does this code produce: “`while (c = getchar() != EOF) putchar(c);`”

QUESTION 4: Deterministic Finite Automata (15 POINTS)

In this question, you will draw two deterministic finite automata (DFA) that accept 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:



4a) Draw a DFA that accepts only strings of a’s and b’s that *begin* and *end* with the same character. Assume only a’s and b’s appear in the input string. (8 points)

QUESTION 4 (continued)

4b) The computer science department has two 200-level courses – 217 and 226. In this question, you will create a deterministic finite automaton (DFA) that recognizes whether a string includes at least one instance of either “217” or “226”, as well as the code that implements it. You can use the symbol “#” as a wildcard that matches “other” character not associated with an existing arc out of a state. Please ensure your DFA has the minimum number of states. (7 points)

QUESTION 5: Abstract Data Types (25 POINTS)

A queue is a first-in-first-out data structure. The Queue ADT offers a simple interface to clients that creates a new queue, checks if a queue is empty, adds an item to the end of a queue, and removes an item from the beginning of a queue. The `queue.h` file specifies the interface, and the `queue.c` file has the code for implementing a Queue using a linked list, where the “head” points to the first element in the list and the “tail” points to the last element. First, `queue.h` has

```
#ifndef QUEUE_INCLUDED
#define QUEUE_INCLUDED

typedef struct Queue_t *Queue_T;

extern Queue_T Queue_new(void);
extern int Queue_empty(Queue_T queue);
extern void Queue_add(Queue_T queue, void* item);
extern void* Queue_remove(Queue_T queue);

#endif
```

Then, `queue.c` has

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

struct list {
    void* item;
    struct list *next;
};

struct Queue_t {
    struct list *head;
    struct list *tail;
};

Queue_T Queue_new(void) {
    Queue_T queue = malloc(sizeof(*queue));
    assert(queue != NULL);
    queue->head = NULL;
    queue->tail = NULL;

    return queue;
}
```

along with the code for `Queue_empty()`, `Queue_add()`, and `Queue_remove()`.

QUESTION 5 (continued)

Here is the code for `Queue_add`:

```
1 void Queue_add(Queue_T queue, void *item) {
2     struct list *newnode ;
3
4     assert(queue != NULL);
5     newnode = (struct list*) malloc(sizeof(*newnode)) ;
6     assert(newnode != NULL);
7     newnode->item = item;
8     newnode->next = NULL;
9     if (queue->tail == NULL)
10        queue->head = newnode;
11    else
12        queue->tail->next = newnode;
13    queue->tail = newnode;
14 }
```

5a) If we were (somehow) absolutely certain the ADT implementation had no bugs, could we safely remove the `assert()` checks after the `malloc()` calls in `Queue_new()` and `Queue_add()`? Why or why not? (5 points)

5b) Why does the `malloc()` call in `Queue_add()` have `sizeof(*newnode)` as an argument instead of `sizeof(newnode)`? (5 points)

QUESTION 5 (continued)

5c) Suppose you wanted to change the implementation of the `queue_t` structure to contain *only* a “head” pointer and *no* “tail” pointer. This would require changes to the last five lines of `queue_add()`. Provide the new implementation below. (Please exclude lines 1-8, since these parts need not change.) (7 points)

QUESTION 5 (continued)

5d) Rewrite your `queue_add()` function from problem 5c to store the items in sorted order, based on the value that “`item`” points to. Do not make any assumptions about the size or type of the items. Please include a new function definition for `queue_add()` but exclude lines 2-8 of the implementation, since these lines of code do not change. (8 points)

QUESTION 6: C Puzzle (5 POINTS)

What output does this code produce? Please explain your answer. (The correct answer without any explanation is worth only 2 points.)

```
char*s="char*s=%c%s%c;main(){printf(s,34,s,34);}";main(){printf(s,34,s,34);}
```

This question may take some time and is only worth five points, so you should complete the rest of the exam to your satisfaction before working on this question.