CSE 333 Midterm Exam 7/22/12

Name _____

There are 6 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, no signal flares, etc.

If you don't remember the exact syntax of something, make the best attempt you can. We will make allowances when grading.

Don't be alarmed if there seems to be more space than is needed for your answers – we tried to include more than enough blank space.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 20
2. _____ / 17
3. _____ / 20
4. _____ / 20
5. _____ / 22
6. _____ / 1

Question 1. (20 points) A little C programming. A *palindrome* is a string that reads the same forwards or backwards. For instance, "madam", "abba", and "x" are palindromes, while "ab", and "foo" are not. You are to complete a function to determine if a string is a palindrome. For this question, a string must be exactly the same forward and backward to be a palindrome, including whitespace (so the string "nurses run" is not a palindrome here). We will also consider an empty string (length 0) to be a palindrome.

Complete the definition of function `IsPalindrome` below so it returns 1 (true) if its string argument is a palindrome and returns 0 (false) if it is not. You may assume that the function argument is a properly \0-terminated C string. You may use any of the C string library functions in <string.h>. You may not copy or modify the string – only examine it.

```
#include <string.h>

// Return 1 if s is a palindrome, otherwise return 0.
// If the string has length 0, return 1 (true).
int IsPalindrome(char *s) {









}
```

Question 2. (17 points) Preprocessor madness. Suppose we have the following code in file `hdr.h`:

```
#ifndef _HDR_H_
#define _HDR_H_
#define BAZ 17
int func(int n);
#endif  // _HDR_H_
```

Then suppose we have the following code in file `hdr.c`:

```
#include "hdr.h"
#define BAR 42
#include "hdr.h"

int main(int argc, char **argv) {
  int n = BAR;
  int f = func(BAZ+BAR);
  return n+1;
}
```

Below, write the exact output produced by the C preprocessor when it reads and processes the file `hdr.c`. In other words, what is the output that is generated by the preprocessor and sent as input to the compiler when we use `gcc` to compile file `hdr.c`?

Question 3. (20 points) A touch of class. The following simple C++ class `Int` has an integer instance variable and various constructors, methods (functions) and a destructor. The various functions write output that includes the value of the instance variable `ival` when they are executed.

```
#include <iostream>
using namespace std;

class Int {
 public:
  Int()             { ival = 17;     cout<<"aaa("<<ival<<")"<<endl; }
  Int(int n)        { ival = n;      cout<<"bbb("<<ival<<")"<<endl; }
  Int(const Int &n) { ival = n.ival; cout<<"ccc("<<ival<<")"<<endl; }
  ~Int()            {                cout<<"ddd("<<ival<<")"<<endl; }
  int get() const   { cout<<"eee("<<ival<<")"<<endl;   return ival; }
  void set(int n)   { ival = n;      cout<<"fff("<<ival<<")"<<endl; }
 private:
  int ival;
};
```

What output is produced when we run the following program that uses this class? If there is more than one possible sequence in which the output appears, write down any one of the possible output sequences.

```
int main(int argc, char **argv) {
  Int p;
  Int q(p);
  Int r(5);
  q.set(p.get()+1);
}
```

Output:

Question 4. (20 points) Bugs, bugs, bugs…. The following C program compiles and links without errors, but it produces segfaults when we run it.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// linked list node for a list of c-strings
typedef struct node {
  char * data;          // string data in this node
  struct node * next;   // next node or NULL if none
} Node;

// print strings in list that starts at head
void prlist(Node * head) {
  Node * p = head;
  while (p != NULL) {
    printf("%s\n", p->data);
    p = p->next;
  }
}

// add x to front of strlist and return pointer to new list head
Node * push_node(Node x, Node * strlist) {
  x.next = strlist;
  return &x;
}

// link two nodes together as a list and then print the list
int main(int argc, char ** argv) {
  Node n1;
  Node n2;
  Node * list = NULL;    // head of linked list or NULL if empty
  // copy "world" to first node and push onto front of list
  strcpy(n1.data, "world");
  list = push_node(n1, list);
  // copy "hello" to second node and push onto front of list
  strcpy (n2.data, "hello");
  list = push_node(n2,list);
  // print list
  prlist(list);
  return EXIT_SUCCESS;
}
```

(continued on next page)

Question 4. (cont.) Describe the error(s) in the code on the previous page, and describe minimal changes that would allow the program to execute successfully. Your changes should not alter the general structure of the program, i.e., it should still link the two nodes in the main program into a linked list and print it. Just describe the changes needed to allow the program to execute successfully.

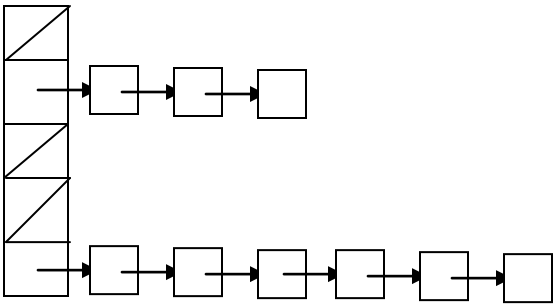
You can either annotate the code on the previous page to show the problems and corrections, or describe them below.

Question 5. (22 points) The question with an alarming amount of reading.

This question concerns that hash table and linked list data structures from HW1 and HW2. You should have a separate set of pages with listings of the header files `ll.h`, `hashtable.h`, and `hashtable_priv.h`, which you should reference as needed when writing the code for this question.

We would like to add a function `AvgBucketLength` to `hashtable.c` that computes the average number of elements in the *non-empty* buckets in the hash table. This is not the same as the load factor (recall that the load factor is a simple average of the total number of elements in the table divided by the number of buckets – that's not what we're after here).

For example, suppose we have a table with 5 buckets where three buckets are empty, one of the others contains 3 elements, and the last one contains 6 elements.



Here we have 2 buckets that are not empty and 9 elements in the non-empty buckets, so the average number of elements in the non-empty buckets is 4.5 (9/2).

Complete the definition of function `AvgBucketLength` on the next page. Besides being correct, your code should be written in reasonably good style (although we will make allowances for the fact that this is an exam). As much as possible, use good variable names and write neatly to make it easier to understand your code. You should use appropriate interfaces defined in `ll.h` to examine the linked lists in the hash table buckets, and be sure to declare appropriate variables, use casts as needed, and so forth.

Question 5. (cont.) Complete the definition of function `AvgBucketLength` below. Assume that this function is being added to `hashtable.c`, and that all of the necessary `#includes` are already present. You probably won't need all of the available space for your answer.

```
// Return the average number of items stored in non-empty buckets
// in the HashTable table. Return 0.0 if the hash table is empty.

double AvgBucketLength(HashTable table) {




}
```

Question 6. (512>>9 points) The **ONE TRUE EDITOR** for programming is

a) ed

b) vi/vim

c) emacs

d) pico

e) nano

f) genie

g) teco

h) notepad

i) notepad++

j) Word

k) cat

l) butterfly

m) None of the above. The correct answer is _____ .

n) I really don't care. Just give me my free point, please.

o) I really do care, but just give me my free point, please.