Name _____

Do **not** write your id number or any other confidential information on this page.

There are 8 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.

You may have a sheet of hand-written notes plus the notes from the midterm if you brought them. Other than that, the exam is closed book, closed notes, closed laptops, closed twitter, closed telepathy, etc.

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

Score _____ / 100

1. _____ / 12
2. _____ / 8
3. _____ / 22
4. _____ / 14
5. _____ / 12
6. _____ / 12
7. _____ / 12
8. _____ / 8

Question 1. (12 points) (testing) In Homework 5, many solutions included a function to add or insert a new word into the trie, creating appropriate strings and nodes as needed (something like `insert(char *word, struct tnode *trie)`). For this problem, describe two appropriate **black-box** tests and two appropriate **white-box** tests for this function. For each test, briefly describe the test setup or inputs and the expected output or results to be checked or verified. You do not need to write any actual code – just describe the tests.

The question continues on the next page to give you plenty of room to write, but *don't* feel obligated to use all the space. Brief and to the point is good.

(a) Describe two different **black-box** tests that you could use to test that the add/insert operation works properly. The tests must check different things – you can't just describe the same test twice with different data.

Black-box test (i) Setup or input:

Expected result(s):

Black-box test (ii) Setup or input:

Expected result(s):

(continued on next page)

Question 1. (cont)

(b) Describe two different **white-box** tests you could use to test the add/insert operation. As before, the different tests must check different things.

White-box test (i) Setup or input:

Expected result(s):

White-box test (ii) Setup or input:

Expected result(s):

Question 2. (8 points) (debugging) Your partner has been working on the memory manager project for a long, long time and seems to be stuck with some sort of bug. After looking at it for a while, the problem seems to be somewhere in file `mumble.c` in function `chop_and_return`. This function begins at line 100 in the file and parts of it look something like this:

```
struct free_node * chop_and_return(
                        struct free_node *big_block, int size) {
   void * leftover_block;
   printf("entered chop_and_return\n");
   leftover_block = (void *)(big_block + size);
   ...
   printf("why don't we ever get to here?\n");
   ...
}
```

(Don't worry about syntactic details in the above code – that's not the point of the question.)

When the program is run, the following output appears:

```
entered chop_and_return
Segmentation fault
```

How would you use `gdb` to find out what was causing the error and/or test your guesses about what might be wrong? Please be specific – what would you do and in what order? What commands would you enter? Note that the question does **not** ask you to figure out what the bug is. We want to know how you would use `gdb` and related tools to find it.

Question 3. (22 points) (C programming) A classic data structure is a *binary search tree* (BST). This is a binary tree where each node in the tree holds a value. For any particular node, all of the values in its left subtree are less than the value in that node and all of the values in its right subtree are greater.

For this problem, we will work with a binary search tree whose nodes contain strings or, more precisely, the data in each node is a pointer to a null-terminated C string. The tree nodes are defined as follows:

```
struct bstnode {          /* node for a BST of strings:     */
   char * str;            /* string value in this node      */
   struct bstnode *left;  /* left subtree or NULL if empty  */
   struct bstnode *right; /* right subtree or NULL if empty */
};
```

You are to write code to insert a new string into the proper place in a binary search tree. Do this in two steps in parts (a) and (b) below.

(a) (10 points) Complete the definition of the following function `new_node` so that it returns a pointer to a newly allocated `bstnode` that references a newly allocated copy of the given string. Some useful `#include`s are provided for you.

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

/* Return a pointer to a newly allocated bstnode holding   */
/* string s. The new node should reference a newly         */
/* allocated copy of the string s, and its left and right  */
/* subtree pointers should be initialized to NULL.         */
struct bstnode * new_node(char * s) {









}
```

(continued on next page)

Question 3. (cont.) (b) (12 points) Now, implement the following function to insert a copy of a string `s` into the proper place in a binary search tree `t`. If a new node is needed to hold `s`, use the `new_node` function from part (a) to create it. If string `s` already occurs in the binary search tree, the function should not change the tree. The function should return a pointer to the root of the (possibly modified) tree `t`.

You may define additional functions in your solution if they are helpful. Clarity and understandability of your solution are much more important than cleverness or coding tricks (particularly for the grader.) Assume that any `#include`s you need to access standard C library functions are provided for you – you do not need to write them.

```
/* Insert a copy of string s into the binary search tree t */
/* and return a pointer to t.  If s already appears in t,  */
/* the tree is not changed and a pointer to t is returned. */
struct bstnode * insert(char *s, struct bstnode *t) {




}
```

Question 4. (14 points) (`make`) We're working on a program that has the following collection of header and implementation files.

```
---------------
mumble.h
---------------
#ifndef MUMBLE_H
#define MUMBLE_H
...
#endif

---------------
generic.h
---------------
#ifndef GENERIC_H
#define GENERIC_H
...
#endif
```

```
---------------
mumble.c
---------------
#include "custom.h"
#include "mumble.h"
...

---------------
test.c
---------------
#include "mumble.h"
...
int main(...) { ... }
```

These source files are used to build an executable program named `test`, whose `main` function is in `test.c` and which uses all of the functions defined in the above files.

The header file `generic.h` is not `#included` directly in any of the other source files. Instead, it contains information that is used to produce the file `custom.h` at the time the program is built (compiled), and that header file is `#included` in `mumble.c`, as shown above.

To produce `custom.h` from `generic.h`, the following `sed` command needs to be executed:

```
sed 's/CITY/Seattle/g' generic.h > custom.h
```

Furthermore, this command should be re-executed to rebuild `custom.h` whenever changes are made to `generic.h`.

(Note: You do not need to worry about changing the `GENERIC_H` symbols that are `#defined` in `generic.h`. It's fine if these are copied unchanged to `custom.h`.)

On the next page, write a `Makefile` that will build program `test` from the various source files, recompiling or regenerating only the necessary files each time the program is built.

You can remove this page for reference while working on the question if you wish.

(continued on next page)

Question 4. (cont.) Write the contents of your `Makefile` here.

Question 5. (12 points) (memory management) In this problem we'd like to write a function to help us analyze the memory manager free list. You should assume that the following `struct` gives the layout of the header part of each free list node.

```
struct free_node {
   int size;              /* number of bytes in this node, */
                          /*   including this header       */
   struct free_node *next; /* next node on the free list,   */
                          /*   or NULL if no more nodes.   */
};
```

Complete the following function so it scans the entire free list and prints the maximum address occupied by *any* part of any block on the free list. (Note that the maximum address occupied by a block is *not* the address of the header node or the beginning of the block.) For this problem, assume that the free list blocks might not be sorted properly, so the maximum address might be found in a block anywhere on the list. If the free list is empty, the function should print `0x00000000` for the maximum address.

Hint: `"0x%08x"` is a suitable `printf` format string to print an `int` value using 8 hex digits with `"0x"` in front.

```
void print_max_address(struct free_node * free_list) {









}
```

Question 6. (12 points) (The ~~dreaded~~ ~~traditional~~ ~~annoying~~ inevitable C++ "what does this print" question) What output is produced when this program is executed? It does compile and execute with no warnings or errors using g++.

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

class Bird {
public:
   virtual void noise() {          cout << "mumble" << endl; }
           void move()  { noise(); cout << "fly" << endl; }
};

class Canary: public Bird {
public:
   void noise() {          cout << "chirp" << endl; }
   void move()  { noise(); cout << "flap" << endl; }
};

class Tweety: public Canary {
public:
   void noise() { cout << "tweet" << endl; }
   void move()  { cout << "run" << endl; }
};

int main() {

   Canary *yellow = new Tweety();
   yellow->noise();
   yellow->move();

   Bird *big = new Tweety();
   big->noise();
   big->move();

   return 0;
}
```

Output:

Question 7. (12 points) (A not-so-Complex C++ problem) A complex number *a*+*b*i can be represented in a program as an object containing a pair of `double`s (floating-point numbers). Here is a header file `complex.h` for a class `Complex` containing one constructor and one operation (multiplication). (`#ifndef`, etc., omitted to save space.)

```
class Complex {
 public:
  // Construct Complex x+yi
  Complex(double x, double y);

  // Return a new Complex object that represents this*other.
  // If this = a+bi and other = c+di the result is a new Complex
  // object with real part = ac-bd and imaginary part = ad+bc.
  Complex times(Complex other);

 private:
  // Representation of a Complex number: re+im*i
  double re;  // real part
  double im;  // imaginary part
};
```

Give implementations below of the constructor and the `times` function as they would appear in a file `complex.cpp` that implements class `Complex`.

```
#include "complex.h"

// write your code below.
```

Question 8. (8 points) (concurrency) Short-answer question spread over two pages to give you room to write – but please don't feel obligated to use all of the space. We thank you for your brevity.

In lecture we used the following example to discuss issues with concurrency. This code declares a simple data structure to represent a bank account and a function to process a withdrawal transaction.

```
struct Acct { int balance; /* ... other fields ... */ };
int withdraw(struct Acct * a, int amt) {
   if(a->balance < amt) return 1; // 1==failure
   a->balance -= amt;
   return 0;                      // 0==success
}
```

(a) The claim is that although this code is correct in a sequential program, it may produce incorrect results in a concurrent program, allowing a negative balance. How could this happen? A short concrete example to support your argument is much better than a long essay.

Question 8. (cont.) (b) One of our summer interns claims that the problem is that the balance modification is not part of the if-statement, and that the concurrency bug can be fixed by rewriting the code this way:

```
struct Acct { int balance; /* ... other fields ... */ };

int withdraw(struct Acct * a, int amt) {
   if(a->balance >= amt) {
      a->balance -= amt;
      return 0;
   } else
      return 1;
}
```

Will this fix the problem? Give a brief technical reason to support your answer.