Name _____

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

There are 10 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. _____ / 8

2. _____ / 8

3. _____ / 10

4. _____ / 14

5. _____ / 12

6. _____ / 12

7. _____ / 8

8. _____ / 8

9. _____ / 12

10. _____ / 8

Question 1. (8 points) (debugging) One of your colleagues is trying to use `gdb` to diagnose the reason for a segfault. Unfortunately, whenever he tries to look at his program, `gdb` won't display the source code, and it doesn't display any variable names – just hexadecimal memory addresses and values.

What is likely to be the problem here? What needs to be done to get `gdb` to display source code and symbolic variable names properly?

Question 2. (8 points) (svn) Another colleague is a bit confused about the difference between the subversion `add` and `commit` commands, since both of them seem to have something to do with putting information in the repository. What is the difference between `svn add` and `svn commit`?

Question 3. (10 points) (testing) When we talked about testing, we distinguished between two kinds of tests: "white box" and "black box".

(a) What is the main difference between these two kinds of tests?

(b) Applying these ideas to the memory manager (`getmem`/`freemem`) assignment, describe two "black box" tests that you could perform to test `getmem`, and two "white box" tests you could perform to test `freemem`.

Two **black box** tests for `getmem`:

Two **white box** tests for `freemem`:

Question 4. (14 points) (trees & strings) As you may recall, a *binary search tree* (BST) is a binary tree where the values in the tree nodes are organized so that all the values in the left subtree of a node are less than the value in the node, and all the values in the right subtree are greater. For this problem, assume that we have a binary search tree whose values are C strings (or more precisely, the values in the nodes are pointers to null-terminated C strings). The tree nodes are defined by the following struct:

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

Complete the definition of the following function so it returns true (1) if string `s` is contained in the binary search tree `t` and returns false (0) if `s` is not contained in tree `t`. For full credit your solution should only search subtrees that might contain the string value. You should assume that any necessary standard library headers are already `#included` and you do not need to write any `#includes`.

```
/* return true (1) if s is contained in the bst */
/* with root t, otherwise return false (0).     */

int contains(struct tnode * t, char * s) {









}
```

Question 5. (12 points) (make) Suppose we have the following collection of C header and implementation files.

```
--------------------
thing.h
--------------------
#ifndef THING_H
#define THING_H

#include "gadget.h"
...
#endif
```

```
--------------------
gadget.h
--------------------
#ifndef GADGET_H
#define GADGET _H
...
#endif
```

```
--------------------
thing.c
--------------------
#include "thing.h"
...
```

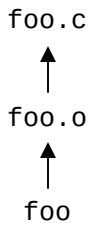
```
--------------------
gadget.c
--------------------
#include "gadget.h"
...
```

```
--------------------
widget.c
--------------------
#include "gadget.h"
#include "thing.h"

int main() { ... }
```

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

Answer the questions on the next page using the above information. You may remove this page from the exam if it makes it easier to use. (Suggestion: sketch your answer to part (a) below before you make a clean copy of it on the next page.)



Question 5. (cont.) (a) Recall that we can specify the dependencies between files in a program using a graph, where there is an arrow drawn from each file name to the file(s) it depends on. For example, the drawing to the left shows how we would diagram an executable program named `foo` that depends on (is built from) `foo.o`, which in turn is depends on `foo.c`.

In the space below, draw a graph (diagram) showing the dependencies between the executable program `widget` and all of the source (`.c`), header (`.h`), and compiled (`.o`) files involved in building it from the files on the previous page.

(b) Write the contents of a `Makefile` whose default target builds the program `widget`, and which only recompiles individual files as needed. Your `Makefile` should reflect the dependency graph you drew in part (a).

Question 6. (12 points) (memory management) While working on the memory manager assignment, several groups ran into problems with free lists that somehow became circular when they shouldn't have been. We'd like to implement a function that could be used to help detect these problems.

For this problem, assume that the following `struct` defines the layout of 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.   */
};
```

In a properly formed free list, the successive nodes should occupy increasing memory addresses. If some node on the list has a `next` pointer that that is not `NULL` and contains a lower memory address, then something is wrong with the list. At a minimum the nodes are out of order, but it also may be that the list is circular.

Complete the definition of function `looks_ok` on the next page so that it returns true (1) if the successive nodes on list `p` are stored at increasing addresses, and returns false (0) if some node has a successor with an address that is less than the address of the node itself.

You should assume that any necessary standard library headers are already `#include`d and you do not need to write any `#include`s.

(write your code on the next page)

Question 6. (cont.) Free list node definition repeated for reference.

```
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.   */
};
```

Write your code for this function below.

```
/* return true (1) if the node addresses on list p are */
/* strictly increasing, otherwise return false (0)     */

int looks_ok(struct free_node *p) {









}
```

Question 7. (8 points) One of the ways we could use the `looks_ok` function from the previous question to test our memory manager code would be to use it in `assert` macros to check the condition of the free list at strategic points. Describe where you would place these `assert` checks in the memory manager code to try to detect a free-list bug as soon as possible, without adding too many extraneous checks that would simply consume time without being likely to find anything useful.

Question 8. (8 points) C++ allows member functions of a class to be declared `virtual`, meaning that dynamic dispatch is used to select the actual function to be executed depending on the runtime type of an object. But it also provides non-virtual functions where the code to execute for a function call is selected statically at compile time. By contrast, in Java and some other languages all function calls are virtual.

Give one technical reason why it can be useful to have non-virtual functions in an object-oriented language and why this option is available in languages like C++.

Question 9. (12 points) The ~~dreaded~~ traditional 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 A {
public:
  virtual void y()   { z(); cout << "A::y" << endl; }
          void z()   {      cout << "A::z" << endl; }
};

class B : public A {
public:
  void y() { z(); cout << "B::y" << endl; }
  void z() {      cout << "B::z" << endl; }
};

class C: public B {

};

int main() {
  B* b = new B();
  b->y();
  b->z();

  cout << "----" << endl;

  A* a = new B();
  a->y();
  a->z();
  return 0;
}
```

Output:

Question 10. (8 points) (concurrency) Problems with electronic voting machines have been much in the news. Because of your expertise in C code and concurrency, you have been asked to look at the secret source code for the Democracy 'R Us ®©™ voting machines to see if you can figure out why they don't always work.

The voting machines are set up on a network, and the individual machines communicate with a central server that adds up the votes. There is a separate process (or thread) on the server for each of the voting machines, and when a vote is cast, the appropriate server process adds one vote to the selected candidate's total by calling this function:

```
/* record a vote for the specified candidate.     */
/* vote_b and vote_g are shared, global variables */
void tally_vote(int candidate) {
   switch (candidate) {
      case BUSH:
              vote_b = vote_b + 1;
              break;
      case GORE:
              vote_g = vote_g + 1;
              break;
   }
}
```

(a) What could cause the vote totals to be incorrect if this function is executed by more than one process at the same time? How might the results be incorrect? (Hint: to compute `x=x+1`, we have to read the old value of `x`, add 1, and store the new value back in `x`, and each of these operations requires executing a separate machine instruction.)

(b) How could you fix the code so that the problem(s) you described in part (a) would not occur in the future?