CSE 303 Final Exam

June 10, 2008

Name _____Sample Solution_____

The exam is closed book, except that you may have a single page of hand-written notes for reference, plus the single page of notes from the midterm.

If you have questions during the exam, raise your hand and someone will come to help. Stay seated.

Please wait to turn the page until everyone has their exam and you have been told to begin.

1	/ 10
2	/ 12
3	/ 9
4	/ 8
5	/ 10
6	/ 8
7	/ 11
8	/ 12
9	/ 8
10	/ 12
Total	/ 100

Question 1. (10 points) Suppose that you are in charge of testing a new implementation of a List class. The List class stores lists of strings and supports the following operations on List objects:

List() – construct a new, empty list
add(s) – add string s to the end of the list
get(k) – get the string at position k of the list (where the first string is at position 0)
size() – return the number of strings in the list
delete(k) – delete the string at position k of the list

For this question, the exact implementation language doesn't matter; you should assume that the list is implemented in Java or a similar mainstream object-oriented language.

Describe 5 different "black-box" tests that you could use to test this list implementation. Your tests should be different enough from each other that they verify or test for different things, and don't just test the same thing several times.

There were (of course) many possible answers. Good ones typically described some operation(s) to perform on the list and what to check for afterwards. For example,

- **Create an empty list and verify that its size is 0**
- **Create a new list, add something to it, verify the list size and that get(0) returned the correct item**
- **Create a new list, add several things, verify the list size and contents**
- **Delete something from a list; check the list (variations: delete the first/last/middle item in the list; delete the last thing in a list; delete the last thing then add something to the empty list that results, …)**

Question 2. (12 points) Suppose we have a collection of C header and implementation files that contain the following #include and other preprocessor statements:

```
--------------------
fee.h
--------------------
#ifndef FEE_H
#define FEE_H
...
#endif

--------------------
fum.h
--------------------
#ifndef FUM_H
#define FUM_H
...
#endif

--------------------
foo.h
--------------------
#ifndef FOO_H
#define FOO_H

#include "fee.h"
...
#endif
```

```
--------------------
fee.c
--------------------
#include "fee.h"
#include "foo.h"
...

--------------------
fum.c
--------------------
#include "fum.h"
...

--------------------
foo.c
--------------------
#include "foo.h"
...

--------------------
grump.c
--------------------
#include "fum.h"
#include "foo.h"

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

Now, before learning about make, we'd been using the following command to compile and link the program:

```
gcc –Wall –g –o grump *.c
```

On the next page, give the contents of a Makefile whose default target builds the program `grump`, but only recompiles files when needed instead of always recompiling everything.

In addition, your Makefile should have a target "clean" so that "make clean" removes the executable grump program, all .o files, and all emacs backup files, whose names end with a ~. A "make clean" should execute without producing any error messages, even if there is nothing to remove.

(You may remove this page from the exam if you like as you write your solution on the next page.)

Question 2. (cont) Write your Makefile here.

There are a lot of possible solutions, particularly if you take advantage of makefile variables or special symbols. Here is a simple one:

```
grump: grump.o, fee.o, fum.o, foo.o
	gcc –Wall –g –o grump  grump.o  fee.o  foo.o  fum.o

grump.o: grump.c, fum.h, foo.h, fee.h
	gcc –Wall –g –c  grump.c

fee.o: fee.c, foo.h, fee.h
	gcc –Wall –g –c  fee.c

fum.o: fum.c, fum.h
	gcc –Wall –g –c  fum.c

foo.o: foo.c, foo.h, fee.h
	gcc –Wall –g –c  foo.c

clean:
	rm –f grump *.o *~
```

A subtle point is that since foo.h includes fee.h, any file that depends on foo.h also depends on fee.h, and that dependency needs to appear in the makefile rule.

Question 3. (9 points) The dreaded C++ "what does this print?" question. Suppose we have the following C++ class declarations, implementations, and main program:

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

class Base {
public:
  virtual void x() { cout << "Base x" << endl; }
  virtual void y() { cout << "Base y" << endl; }
           void z() { cout << "Base z" << endl; }
};

class Extended: public Base {
public:
           void x() { cout << "Extended x" << endl; }
  virtual void y() { cout << "Extended y" << endl; }
           void z() { cout << "Extended z" << endl; }
};

int main() {
  Base *p = new Extended();
  p->x();
  p->y();
  p->z();
  delete p;
  return 0;
}
```

(In the above code, the function implementations are included as part of the classes, instead of being written as separate functions elsewhere. This is legal C++ and compiles and executes without errors.)

What output is produced when this program is executed?

Extended x

Extended y

Base z

Question 4. (8 points) One of your colleagues has heard about the C assert macro and is trying to use it in his code. He's got a program that creates a struct of type Thing, calls an initialization function to initialize it, then calls another function to use it. Here's the relevant part of the code, including the assert check to detect when the initialization fails during testing:

```
#include <assert.h>

/* Data structure definition – fields omitted and not relevant */
struct Thing {
   ...
}

/* Initialize Thing *p.  Return true (1) if successful and false (0) if not */
int init(struct Thing *p) { ... }

/* Create a Thing struct, initialize it, then process and delete it */
void do_something() {
  struct Thing * t;
  t = (struct Thing *)malloc(sizeof(struct Thing));
  assert(init(t));
  process_thing(t);
  free(t);
}
```

Unfortunately something is flakey about this code. Depending on the options used to compile it, the code sometimes gets the right results and sometimes fails to work properly.

(a) What is probably wrong here?

If debugging is disabled, no code is generated for an assert macro, so any expressions inside an assert are not executed. In this case, init(t) is not executed if debugging is disabled.

(b) How could the code be fixed so it works correctly and still does everything the author apparently intends? (You can explain your answer here, or mark up the code above and explain your changes.)

Separate out the init call from the assert so it is guaranteed to execute regardless of debug settings:

```
int init_result = init(t);
assert(init_result);
```

Question 5. (10 points) Another buggy program, alas. Another of your friends needs some help with a program that creates a computerized address book. Here is the C code:

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

struct Address {    // address book entry
  char *name;       //   person's name (string)
  char *address;    //   person's address (string)
  int zip;          //   person's zip code
};

/* Allocate a new address struct, copy the name, address, and zip code */
/* into it, and return a pointer to the newly allocated struct.        */
struct Address * new_address(char *n, char *a, int code) {
  struct Address *p = (struct Address *) malloc(sizeof(struct Address));
  strcpy(p->name, n);
  strcpy(p->address, a);
  p->zip = code;
  return p;
}
```

(a) What is the problem?

The fields p->name and p->address are uninitialized pointers, so attempts to copy strings there will cause some sort of trouble (probably a segfault).

(b) How would you fix it? Show what needs to be added, deleted, or changed in the above code so it works properly.

Put appropriate mallocs after the first malloc that creates the Address struct, but before the strcpy operations:

```
p->name    = (char*) malloc((strlen(n)+1) * sizeof(char));
p->address = (char*) malloc((strlen(a)+1) * sizeof(char));
```

Question 6. (8 points) Concurrency. Suppose we have multiple threads rendering the frames of a computer animation film . Each thread is responsible for rendering some number of frames. There is a global int variable that keeps track of the total number of frames rendered so far, and each time one of the threads finishes rendering some frames, it calls update_frame_total to add to this total.

```
/* Global variable, initially 0.  Total number of frames rendered so far */
int total_frames_rendered = 0;

/* Update total_frames_rendered to record that n more frames are done */
void update_frame_total(int n) {
  int new_total = total_frames_rendered + n;
  total_frames_rendered = new_total;
}
```

Unfortunately, something isn't right with the code. Sometimes total_frames_rendered has the right value; sometimes it doesn't, and the values can be different even when the program is run again with the same input data.

(a) What is the likely problem here?

If two threads execute update_frame_total simultaneously, it is possible for one thread to be interrupted between the time it reads the old value of total_frames_rendered and the time it writes back the new value. The other thread could then either read an incorrect value for total_frames_rendered, or write a new value that is ignored by the first thread.

(b) Explain how the code could be modified to avoid the problem. You don't have to give precisely correct C code to do this – just give an accurate idea of what needs to be done and where the changes are needed.

The two lines of code in the body of the function need to be executed as an uninterruptable critical section. Either surround those statements with `atomic { ... }` (if something like that is available); or associate a lock variable with total_frames_rendered and add code to the function to acquire the lock before reading total_frames_rendered and release the lock after writing the updated value.

Question 7. (11 points) In assignment 4 we implemented a trie to store the words in a dictionary using their numeric equivalents. The data structure for the trie nodes in most programs (and for this question) was declared like this:

```
struct tnode {              // A node in the trie:
  char * word;              //   Word in this node if there is one, otherwise NULL.
  struct tnode* child[10]; //   Children of this trie node.  child[2]-child[9] are
};                          //   the children for digits 2-9.  child[0] points to a
                            //   node whose word has the same digit spelling as this
                            //   one, if any (# key).  child[1] is unused.  Each child
                            //   entry is NULL if it doesn't point to another tnode.
```

Recall that a node in the trie points to a word string when the path to that node corresponds to a digit sequence for that word. Nodes contain a NULL word pointer if they don't appear at the end of a complete sequence of nodes that make up the digits for a word.

Complete the definition of the following recursive function so that it returns a count of the number of words stored in the trie whose root is t (i.e., the number of tnodes whose word pointer is not NULL). You may not define any global variables or additional functions, and you may not alter the parameter list in any way.

```
/* return the number of word strings in the trie with root t */
int nstrings(struct tnode * t) {

  int k;
  int result;

  /* result is 0 if this trie is empty */
  if (t == NULL) {
    return 0;
  }

  /* set result to 0 or 1 depending on whether this node has a string */
  if (t->word != NULL) {
    result = 1;
  } else {
    result = 0;
  }

  /* recursively count words in subtries and add those to the total */
  for (k = 0; k < 10; k++) {
    result += nstrings(t->child[k]);
  }

  /* return number of strings in this trie */
  return result;
}
```

Question 8. (12 points) A typical way to represent the free list in the getmem/freemem storage allocator is as a linked list of blocks that begin with the following C struct:

```
struct free_block {
  int size;                  // number of bytes in this block,
                             //      including this header
  struct free_block * next;  // next block on the free list
};
```

Although different people used different conventions in the actual project, for the purposes of this problem (if it matters), assume that the size in a free_block header includes both the free_block struct itself plus the rest of the block, and that it is the total number of bytes.

One of the operations needed in the project was to insert a new block on the free list in the correct position. For this problem, you should assume that there is a single global pointer to the free list:

```
struct free_block * free_list;   // free list blocks; NULL if none
```

and that the blocks on the free list are stored in ascending order by block address.

Complete the definition of function insert_free_block on the following page so it inserts block b in the correct place on the free list. You should assume that pointer b points to a free_block header and not to somewhere else in the block, You do **not** need to merge block b with any adjacent blocks to make a larger block, even if it is immediately adjacent to some block already on the free list.

(write your code on the next page)

Question 8. (cont.) Reminders: A free list node begins as follows:

```
struct free_block {
  int size;                  // number of bytes in this block,
                             //      including this header
  struct free_block * next;  // next block on the free list
};
```

The head of the free list is this global variable:

```
struct free_block * free_list;   // free list blocks; NULL if none
```

Write your code below.

```
/* Insert block b in the correct place on list free_list */
void insert_free_block(struct free_block * b) {

  /* initialize pointer p to beginning of free list */
  struct free_block *p = free_list;

  /* if new node belongs at the beginning of the list, insert it there and exit */
  if (p == NULL || b < p) {
    b->next = p;
    free_list = b;
    return;
  }

  /* Advance p to last node on the free list whose address is less than b */
  while (p->next != NULL && p->next < b) {
    p = p->next;
  }

  /* here, either p->next == NULL or p->next >= b; */
  /* insert block b between p and p->next.         */
  b->next = p->next;
  p->next = b;

}
```

Notes: The boolean expressions in the code above depend critically on the short-circuit behavior of && and ||, which do not evaluate their right operands if those are not needed to determine the result.

Technically, it is not legal in standard C to compare two pointers unless they are known to point into the same array. To get this absolutely right we would need to cast the pointers to unsigned ints before doing the comparisons. But for the purpose of this question that was a technicality we didn't worry about.

Question 9. (8 points) Your partner doesn't quite have the hang of using svn for version control. He's managed to check out a local (working) copy of the project, and he's updated a couple of files. Now he wants to transfer his changes back into the repository to update the master copy.

(a) Your partner used the "svn update" command to try to update the repository with his modified files. But this didn't work. Why not? What happened instead?

svn update merges any changed files in the repository with the local copies, bringing the local copies up to date. It does not copy files from the local working directory to the repository.

(b) What should your partner have done instead of, or in addition to, the "svn update" command to get his changes to the project properly stored in the repository?

Use svn commit to copy the local changes to the repository.

(It's not a bad idea to use svn update first to check for any conflicting changes that are already in the repository, but it's not required.)

Question 10. (12 points) A little C++. Suppose we want to implement a C++ class to represent points in a 2-D plane (i.e., points with x and y coordinates). Here is the definition for an appropriate class, which you should assume is stored in file point.h

```
class Point {
public:
  // construct a new point with coordinates 0.0, 0.0
  Point();

  // construct a new point with coordinates xloc, yloc
  Point(double xloc, double yloc);

  // return a new Point that is the midpoint of a line segment between
  // this Point and Point other.
  Point midpoint(Point other);

private:
  // representation: x and y coordinates of this point
  double x, y;
};
```

Give the contents of a C++ file point.cpp that contains everything necessary to implement the two constructors and the midpoint function for this class. To save time, you do not need to copy the function heading comments.

```
#include "point.h"

Point::Point() { x = 0.0; y = 0.0; }

Point::Point(double xloc, double yloc) { x = xloc; y = yloc; }

Point Point::midpoint(Point other) {
  return Point((x+other.x)/2.0, (y+other.y)/2.0);
}
```

The body of function midpoint could have been written in various other ways, for example:

```
  Point result;
  result.x = (x + other.x) / 2.0;
  result.y = (y + other.y) / 2.0;
  return result;
```