Name:_____

CSE 303 - Practice Final Exam

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for two 8.5x11in pieces of paper (both sides).
- You can rip apart the pages, but please write your name on each page.
- There are **100 points** total, distributed **unevenly** among 11 questions (many of which have multiple parts).
- When writing code, style matters, but don't worry about indentation.

Question	Max	Grade
1	15	
2	12	
3	12	
4	10	
5	7	
6	7	
7	10	
8	10	
9	7	
10	4	
11	6	
Total	100	

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit.**
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.

Name:_____

1. (**15** points) Consider the following declarations in a C program:

```
typedef struct node {
  char *name;
} Node;


Node* create(char* message) {
  Node *new_element = (Node*)malloc(sizeof(Node));
  if ( !new_element ) {
      printf(``Out of memory\n'');
      return NULL;
  }
  new_element->name = message;
  return new_element;
}


void destroy(Node* garbage_element) {
    free(garbage_element->name);
    free(garbage_element);
}
```

(a) (**4** points) Why will the following sequence of instructions cause a segmentation fault?

```
char message[] = "Hello World";
Node *element = create(message);
if ( element ) {
    printf("Element is %s\n",element->name);
    destroy(element);
}
```

Name:_____

(b) (**4** points) What is the problem with the following sequence of instructions?

```
char message[] = "Hello World";
char *copy = (char*)malloc(strlen(message)+1);
strcpy(copy,message);

Node *element1 = create(copy);
Node *element2 = create(copy);
if ( element1 ) {
    destroy(element1);
}
if ( element2 ) {
    destroy(element2);
}
```

(c) (**7** points) Write a `create` function that avoids the problems above.

Name:_____

Name:_____

2. (**12** points) Consider a list of integers composed of self-referential structures of the form:

```
typedef struct node {
    int value;
    struct node *next;
} Node;
```

(a) (**7** points) Write a C function that meets the specifications below. You do not need to check the first precondition.

```
/**
 * Clears a list: removes all elements from the list and
 * frees the space that was allocated for each element
 *
 * Precondition 1: all elements in the list were allocated on the heap
 * Precondition 2: head is not NULL
 *
 * @param head, address of pointer to the first element in the list
 * @return nothing
 */
void clear(Node** head);
```

Name:_____

(b) (**5** points) Complete the implementation of the following C function

```
/**
 * Makes a copy of a list of integers
 *
 * Precondition: the src list is not empty
 *
 * @param src, pointer to the first element in the list to copy
 * @return a pointer to the first elment in the copy of the list
 */
Node* copy(Node* src) {

    // Checking precondition
    assert(src);

    // Step 1: make a copy of the first element in the list
    Node *dst = (Node*)malloc(sizeof(Node));
    if ( ! dst ) {
        printf("Out of memory");
        return NULL;
    }

    dst->value = src->value;
    dst->next = NULL;

    // Copy the other elements
    // ...
```

Name:_____

Name:_____

3. (**12** points) Suppose a program is composed of the following files: `A.h`, `A.cc`, `B.h`, `B.cc`, `X.h`, and `main.cc`. You are given the Makefile below:

```
CXX = g++
CXXFLAGS = -Wall -g
LDFLAGS =

PROGRAMS = main

OBJ  = A.o B.o main.o
HEADERS = A.h B.h X.h

all: $(PROGRAMS)

%.o: %.cc %.h X.h
        $(CXX) $(CXXFLAGS) -c $<

main.o: main.cc $(HEADERS)
        $(CXX) $(CXXFLAGS) -c $<

main:   $(OBJ)
        $(CXX) $(LDFLAGS) -o $@ $^

clean:
        rm -f *.o $(PROGRAMS)
```

Reminder:

- `$@` designates the current target
- `$^` designates all prerequisites
- `$<` designates the left-most prerequisite

Suppose we just executed `make`.

(a) (**7** points) Write the sequence of commands that will get executed if we delete `main.o` and type `make` again. List the commands in the exact order in which they will get executed.

Name:_____

(b) (**5** points) Explain the difference between the sequence of commands that will get executed if we either modify `B.h` and type `make` or modify `X.h` and type `make`.

Name:_____

4. (**10** points) Alice, Bob, Chuck, and Donna are working on the project from the previous question.

   (a) (**3** points) They wish to manage the project with the version control system, CVS. Circle the files they should add to CVS:

`A.h`	`A.cc`	`A.o`
`B.h`	`B.cc`	`B.o`
`X.h`	`main.cc`	`main.o`
`main`	`Makefile`	

   (b) (**5** points) Each team member checks-out a local copy of the project. They perform the following changes:

   - Alice moves part of the content from files `A.h` and `A.cc` into two new files `C.h` and `C.cc`. Alice modifies the `Makefile` and `main.cc` to reflect these changes.
   - Bob completes the implementation of `B.cc`.
   - Chuck and Donna do nothing.

   List the CVS operations (add, commit, or update) that each team member must perform in order for everyone to end-up with the most recent version of the project. List the commands, and the users who need to execute them, in an order that achieves the desired result. Pick any of the possible orders.

   (c) (**2** points) Donna and Chuck now make different changes to the same lines inside file `main.cc`. Donna commits her changes first. What happens when Chuck tries to update `main.cc` before committing his own changes. Circle the appropriate answer:

   i. CVS merges the changes made by Donna and Chuck. Because the changes affect the same lines, Donna's changes automatically overwrite Chuck's changes.

   ii. CVS merges the changes made by Donna and Chuck. Because the changes affect the same lines, Chuck's changes automatically overwrite Donna's changes.

   iii. CVS tries to merge the changes made by Donna and Chuck. Because the changes affect the same lines, CVS labels the changes as conflicting. Chuck must resolve the conflicts manually.

Name:_____

5. (**7** points) Given the following specification for function `count_occurrences`, and the two sample test-cases, write 5 additional test-cases for this function:

```
/**
 * Computes the number of occurrences of a word in a file.
 * Precondition: file_name and word are null terminated strings
 * Postcondition:
 * If file_name is null, returns zero
 * If word is null, returns zero
 * If the file does not exist, returns zero
 * If the word is the empty string, returns zero
 * @param file_name, the name of the file to search
 * @param word, the word to search for in the file
 * @return the number of occurrences of word in the file file_name.
 * Computes occurrences only up to 100. If a word occurs
 * more than 100 times, the function returns 100.
 */
int count_occurrences(char* file_name, char* word);
```

(a) A set of inputs where `word` is NULL.

(b) A set of inputs where `word` occurs in the file `file_name` exactly once.

Name:_____

6. (**7** points) For each statement below, indicate if it is true or false.

   (a) The goal of specifications is to describe the details of how a software system is implemented.
   true false

   (b) The goal of testing is to guarantee that there are no bugs in a software system.
   true false

   (c) It is good programming practice to check preconditions explicitly when possible.
   true false

   (d) If a program allocates only a small number of objects on the heap, it is safe to assume that all allocations always succeed. There is not need to check if `malloc` or `new` return NULL.
   true false

   (e) When performing black-box testing, the test-cases are designed in terms of the specifications.
   true false

   (f) When performing white-box testing, the test-cases are designed in terms of the implementation.
   true false

   (g) Code readability is not important as long as the code is fast.
   true false

Name:_____

7. (**10** points) In a C program called `main`, an integer, `x`, is transformed by a series of four functions: `f1`, `f2`, `f3`, and `f4` as shown below:

```
// x is the program input
int x2 = f1(x);
// ... some code that does not use x2
int x3 = f2(x2);
// ... some code that does not use x3
int x4 = f3(x3);
// ... some code that does not use x4
int x5 = f4(x4);
// ... some code that does not use x5
```

   (a) (**5** points) As you test the program, you notice that for some inputs, the output `x5`, is not what you expect. Explain how you would use a debugger, such as `gdb`, to determine which one of the four functions performs the wrong computation.

Name:_____

(b) (**5** points) As you continue testing the program, you notice that for some inputs, the program takes a very long time to execute. What type of tool can you use to determine where the problem is? Name the type of tool and two types of information that this tool can provide about a program.

Name:_____

8. (**10** points) Indicate the output of the following C++ program using the provided blanks. **Be careful!** Carefully examine all function signatures.

```
// ------------------------------------------------
// Content of Simple.h
// ------------------------------------------------
#ifndef SIMPLE_H
#define SIMPLE_H

#include <iostream>
#include <string>
#include <sstream>

using namespace std;

#define DEFAULT 100

class Simple {

public:
    Simple(int value = DEFAULT);
    ~Simple();
    int  getValue();
    void setValue(int new_value);
    string toString();

private:
    int _id;
    int _value;
    static int s_counter;
};

#endif

// ------------------------------------------------
// Content of Simple.cc
// ------------------------------------------------
#include "Simple.h"

int Simple::s_counter = 1;

Simple::Simple(int value)
    : _id(s_counter),
      _value(value) {
    s_counter++;
}

Simple::~Simple() {
}
```

Name:_____

```
int Simple::getValue() {
    return _value;
}

void Simple::setValue(int new_value) {
    _value = new_value;
}

string Simple::toString() {
    stringstream s;
    s << "(" << _id << "," << _value << ")";
    return s.str();
}

// ------------------------------------------------
// Content of main.cc
// ------------------------------------------------
#include "Simple.h"

void increase_value(Simple object) {
    int new_value = 2*object.getValue();
    object.setValue(new_value);
}

void decrease_value(Simple& object) {
    int new_value = object.getValue()/2;
    object.setValue(new_value);
}

int main() {

    Simple object1(20);

    Simple object2;

    cout << object1.toString() << endl;  // Output: __________

    cout << object2.toString() << endl;  // Output: __________

    increase_value(object1);

    decrease_value(object2);

    cout << object1.toString() << endl;  // Output: __________

    cout << object2.toString() << endl;  // Output: __________

    return 0;
}
```

Name:_____

9. (**7** points) Indicate the output of the following C++ program using the provided blanks.

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

class Parent {

public:
    void print1() {
        cout << "Parent ";
    }
    virtual void print2() {
        cout << "Parent ";
    }
    void print3() {
        cout << "Parent ";
        print2();          // Note that this->print2(); would produce the same result
    }
};

class Child: public Parent {

public:
    void print1() {
        cout << "Child ";
    }
    virtual void print2() {
        cout << "Child ";
    }
    void print3() {
        cout << "Child ";
        print2();          // Note that this->print2(); would produce the same result
    }
};

int main() {

    Parent *p = new Child();

    p->print1();   // Output: ___________________

    p->print2();   // Output: ___________________

    p->print3();   // Output: ___________________

    return 0;
}
```

Name:_____

10. (**4** points) A program is composed of two threads and a shared variable `x`. Each thread performs the sequence of actions shown below.

```
// Thread 1
int t1 = x;
t1 = t1 + 1;
x = t1;

// Thread 2
int t2 = x;
t2 = t2 - 1;
x = t2;
```

A thread cannot be interrupted during the execution of an instruction (in the middle of a line). A thread can be interrupted between any two instructions (between lines).

Assume the original value of `x` is 0. If the two threads execute concurrently, what are the possible final values for `x`?

Name:_____

11. (**6** points) A program is composed of three threads and shared objects `X`, `Y`, and `Z`.

    Which one of the following locking strategies can cause a deadlock? Show the sequence of operations that lead to the deadlock and explain why the other strategy will never cause a deadlock.

```
//   Strategy 1              //   Strategy 2

// Thread 1                 // Thread 1
Acquire lock for X          Acquire lock for X
Acquire lock for Y          Acquire lock for Y
Modify X and Y              Modify X and Y
Release lock on Y           Release lock on Y
Release lock on X           Release lock on X

// Thread 2                 // Thread 2
Acquire lock for Z          Acquire lock for X
Acquire lock for X          Acquire lock for Z
Modify Z and X              Modify Z and X
Release lock on X           Release lock on Z
Release lock on Z           Release lock on X

// Thread 3                 // Thread 3
Acquire lock for Y          Acquire lock for Y
Acquire lock for Z          Acquire lock for Z
Modify Y and Z              Modify Y and Z
Release lock on Z           Release lock on Z
Release lock on Y           Release lock on Y
```