CSE 333 Midterm Exam 5/9/14

Name _____

There are 5 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, open mind.

If you don't remember the exact syntax for 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. _____ / 12

3. _____ / 28

4. _____ / 12

5. _____ / 28

Question 1. (20 points) C programming. Implement the C library function `strncpy`. The specification of `srncpy` is as follows:

Copy characters (bytes) from `src` to `dst` until either a `'\0'` character is found in `src` or until `n` characters are copied, whichever comes first. If fewer than `n` characters were copied to `dst`, the remaining characters (bytes) of `dst` should be filled with `'\0'` bytes. The function should return the original value of pointer `dst`.

```
char * strncpy(char * dst, char * src, int n) {




}
```

Question 2. (12 points) Build dependencies. Suppose we have a project with multiple source files that have the `#include` dependencies shown below.

```
***************
* foo.h       *
***************
#ifndef _FOO_H_
#define _FOO_H_
...
#endif
```

```
***************
* foo.c       *
***************
#include "foo.h"
...
```

```
***************
* main.c      *
***************
#include "foo.h"
#include "bar.h"

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

```
***************
* bar.h       *
***************
#ifndef _BAR_H_
#define _BAR_H_

#include "foo.h"
...
#endif
```

```
***************
* bar.c       *
***************
#include "bar.h"
...
```

We could use the following `gcc` command to compile these files and produce an executable program named `main`:

```
gcc -Wall -g -o main *.c
```

But we would like to do better than that by figuring out the dependencies between these C source files, the compiled `.o` files, and the final executable file `main`. The dependency information could be used to create a Makefile that only recompiles necessary files when a source file is changed. However, for this problem you only need to draw a graph showing the dependencies between the files involved in the build process.

Draw the dependency graph on the next page. You may remove this page for reference while answering the question if you wish.

Question 2. (cont) Draw a graph (diagram) showing the dependencies between the various files involved in building this program. The final executable file `main` should be drawn at the bottom, and there should be arrows from each file to all of the the files that it *directly* depends on. Remember that one file depends on another only if the file needs to be rebuilt when a file it depends on changes. Be sure that you include the intermediate `.o` files generated by the compiler as well as the original source files and headers and the final executable program.

Question 3. (28 points) Not the traditional C program. This time it comes in pairs!

Consider the following program, which does compile and execute without any errors.

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

typedef struct pair {
  int a, b;
} Pair, *PairPtr;

void xx(PairPtr one, PairPtr two, PairPtr three) {
  one->a   = three->b + 1;
  three->b = two->a;
  two->a   = 2*two->b;
  // ☞ HERE ☜ (see next page)
  printf("one = %d, %d; two = %d, %d; three = %d, %d\n",
       one->a, one->b, two->a, two->b, three->a, three->b);
}

void yy(Pair p, PairPtr q) {
  Pair w = {1, 2};
  q->b = 0;
  xx(&p, &w, q);
  printf("p = %d, %d; q = %d, %d; w = %d, %d\n",
       p.a, p.b, q->a, q->b, w.a, w.b);
}

int main() {
  Pair r = {17, 42};
  PairPtr s = (PairPtr)malloc(sizeof(Pair));
  s->a = 3;
  s->b = 33;
  yy(r,s);
  printf("r = %d, %d; s = %d, %d\n", r.a, r.b, s->a, s->b);
  free(s);
  return 0;
}
```

Answer questions about this program on the next page. You may remove this page for reference if you wish.

Question 3. (cont.) (a) (16 points) Draw a boxes 'n arrows diagram showing state of memory when control reaches the comment containing ☞HERE☜ in the middle of function `xx`. Your diagram should have three boxes showing the stack frames for functions `main`, `yy`, and xx. The stack frames should show values of all local variables. Draw each `pair struct` as a box with two labeled fields `a` and `b`. Draw an arrow from each pointer to the location that it references. Data that is allocated on the heap should be drawn in a separate area, since it is not part of any function stack frame After drawing your diagram, be sure to answer part (b) at the bottom of the page.

(b) (12 points) What output does this program produce when it is executed?

Question 4. (12 points) References please? Assume that we have a class `Point` that defines 2-D points. We want to define a new class `Circle` that uses a `Point` to specify the center of each circle. The declaration for that class is given below, but there are lots of empty spaces where perhaps things are missing.

Complete the declarations by filling in any necessary keywords or symbols. You should leave each space empty if that is appropriate, or write in some combination of `&`,`*`, `const`, `static`, `void`, `Point`, `Circle`, or whatever else is needed to declare things correctly. If something is optional or if you have choices between more than one way to fill in a blank, make the most appropriate choice.

```
class Circle {

 public:

  // ordinary constructor with center and radius parameters

  Circle( ______ Point ______ center,

                      ______ double ______ radius) ______ ;

  // copy constructor

  Circle( ______ Circle ______ copyme) ______ ;


  // return center point of this Circle

  Point _____ getCenter( )______;

  // change center of this Circle to Point p

  void _____ setCenter( _____ Point _____ p)______;

  // assignment

  Circle _____ operator=( ______ Circle ______ rhs) ______;

 private:
  Point center_;
  double radius_;
};
```

Question 5. (28 points) A little C++ programming. After building a double-linked list using C for HW1, we'd like to try something similar in C++. Here is part of the header file for a double-linked list class where each node in the list contains a simple `int` value. It includes a `struct` type `IntNode` for the nodes and a separate class that uses those nodes as components of the actual list.

(Remember that in C++, a `struct` declaration actually defines a type, and the name of the `struct` can be used directly as a type name. No `typedef` is necessary. The declarations of the `next` and `prev` pointers below are correct.)

```
struct IntNode {
  int value;       // value in this node
  IntNode *next;   // next node in the list or null if none
  IntNode *prev;   // prev node in the list or null if none
};


class IntLinkedList {
 public:
  // construct empty IntLinkedList
  IntLinkedList() : first_(nullptr), last_(nullptr) { }

  // destructor
  ~IntLinkedList();

  // add a new node with value n as the last list node
  void push_back(int n);

  // remaining operations omitted...

 private:
  IntNode * first_;   // first node in the list
  IntNode * last_;    // last node in the list
};                    // first_ and last_ are both set to
                      //   nullptr if the list is empty
```

You should assume that all dynamic memory allocation and deletion functions will succeed without errors – you do not need to check for failures or handle exceptions.

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

Question 5. (cont.) Provide implementations for method `push_back` and for the destructor for this class as they would appear in a separate implementation file `IntLinkedList.cc`. The function headers are given for you.

```
// add a new node with value n as the last list node
void IntLinkedList::push_back(int n) {









}

// destructor
IntLinkedList::~IntLinkedList() {









}
```