CSE 333 Midterm Exam 5/1/15

Name ____

There are 6 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. ______ / 18

2. ______ / 18

3. ______ / 20

4. ______ / 20

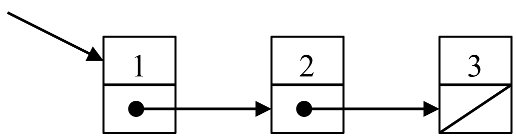
5. ______ / 18

6. ______ / 6

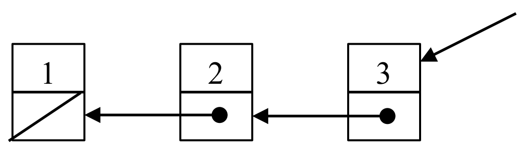
Question 1. (18 points) C pointer swizzling. For this problem write a function to reverse the order of the nodes in an integer linked list. The nodes of the list are defined as follows:

```
typedef struct Node_t {
  int val;                // value in this node
  struct Node_t * next;   // next node, or NULL if none
} Node;
```

The function reverse must reverse the order of the nodes in the original list by traversing the list in a *single* pass and update the next pointer in each node to point to the node that preceded it in the original linked list. So, for instance, if the list was originally



then, after the reverse, the list should look like this:



Your code may not allocate or delete any Nodes – it needs to modify the `next` pointers in the existing ones. You may not alter the `val` fields in any of the nodes. The result of the function should be a pointer to the node that is now the "first" node in the list, which was originally the "last" node – the node containing 3 in the example above. If the function is asked to reverse an empty list (`NULL`) it should return `NULL`.

Hint: use the space below to draw a diagram and think through which pointers you need to keep track of and how to traverse the list. The answer is fairly short once you've thought it through. When you're ready, fill in the function definition …

on…

the…

next…

page…

|
|
V

Question 1. (18 points) Complete the definition of function reverse here:

```
// Reverse the nodes in the list starting at head
// and return a pointer to the new first node, which
// is the last node in the original list.  If the
// list is empty, return NULL
Node * reverse(Node * head) {






}
```

Question 2. (18 points) Suppose we have the following files in the same directory.

File `foo.h`

```
#ifndef _FOO_H_
#define _FOO_H_

#define FOOBAR BAR - x
int foo(int x);

#endif // _FOO_H_
```

File `foo.c`

```
#include "./foo.h"
#define BAR 353

int foo(int x) {
  return FOOBAR / 10;
}
```

File `foomain.c`

```
#include <stdio.h>
#include "./foo.h"

int main(int argc, char* argv[]) {
  int nbr = foo(333);
  printf("The output of foo(333) is %d\n", nbr);
}
```

(a) (10 points) Below, write the output produced by the C preprocessor (`cpp -P` or `gcc -E`) showing exactly how file `foo.c` (*not* `foomain.c`) is rewritten by `cpp` and sent to the compiler phase of `gcc`. (The preprocessor can correctly process this file without any blocking errors.)

(b) (8 points) What output is produced when `foo.c` and `foomain.c` are compiled and the resulting program is linked and executed?

Question 3. (20 points) Bugs 'R Us. Here is a small program that has some problems (at least 2; almost certainly more than that). You should find all the bugs and give a brief explanation of each problem. You are not required to fix the bugs (and there might not be obvious fixes in some cases), but sometimes an easy way to explain a bug is to show how to correct it.

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

typedef struct point {
  int x, y, z;
} Point, *PointPtr;

PointPtr add(PointPtr a, PointPtr b) {
  Point res;
  res.x = a->x + b->x;
  res.y = a->y + b->y;
  res.z = a->z + b->z;
  return &res;
}

int main(int argc, char* argv[]) {
  Point a = {1, 2, 3};
  PointPtr b;
  b->x = 3;
  b->y = 5;
  b->z = 7;
  PointPtr c = malloc(sizeof(PointPtr));
  c = add(&a, b);
  printf("a = (%d, %d, %d)\n", a.x, a.y, a.z);
  printf("b = (%d, %d, %d)\n", b->x, b->y, b->z);
  printf("a + b = (%d, %d, %d)\n", c->x, c->y, c->z);
  free(b);
  free(c);
}
```

Question 4. (20 points) Suppose we have the following C header file, which contains information needed to declare and use a `ValStore` type, which is a data structure that holds key-value pairs. The actual implementation is private and not part of the header.

File `ValStore.h`:

```
#include <inttypes.h>              // for uint64_t, etc.

struct ValStore_impl;              // private
typedef struct ValStore_impl* ValStore;
typedef void* Value_t;

// Store key/val pair in ValStore vals
void storeValue(ValStore vals, uint64_t key, Value_t val);

// Retrieve val from ValStore vals given the key.  Result
// is NULL if no value is stored with that key
void getValue(ValStore vals, uint64_t key, Value_t *val);
```

Here is some C client code that declares a type `DataPack`. Your job is to complete two functions that store and retrieve `DataPack` values from a `ValStore`.

```
typedef struct DataPack_t {
  // details omitted -- not needed
} DataPack;
```

(a) (6 points) Write the function `storeDataPack`. Hint: this is very simple – it just needs to call `storeValue`, so all you need to do is fill in appropriate parameters.

```
// store DataPack value in ValStore with a given code as the key
void storeDataPack(ValStore vs, uint64_t code, DataPack* data) {

  storeValue(_____________, _____________, _____________);
}
```

(b) (14 points) Write the function getDataPack. Hint: use `getValue`.

```
// Given a ValStore vals and a code, return a pointer to DataPack
// stored in vals whose key equals code, or NULL if not found
DataPack * getDataPack(ValStore vals, uint64_t code) {






}
```

Question 5. (18 points) A bit of C++ hacking. We are defining two related classes: class `Point` is a simple 2-D point with integer *x* and *y* coordinate values:

File `Point.h`:

```
class Point {
 public:
  // constructor
  Point(int x, int y): x_(x), y_(y) { }

  // copy constructor
  Point(const Point & other): x_(other.x_),y_(other.y_) { }

  // accessors
  int get_x() const { return x_; }
  int get_y() const { return y_; }
  // no additional constructors, destructor, or assignment
  // defined explicitly in this class
 private:
  int x_, y_;
};
```

A `Rectangle` has two instance variables, which are pointers to `Point` objects allocated on the heap. Each `Rectangle` has its own private copies of its corner `Point`s and does not share them with other `Rectangle`s. Here is the class declaration:

File `Rectangle.h`:

```
#include "./Point.h"

// A rectangle is defined by two Point objects giving its
// upper-left and lower-right corners
class Rectangle {
 public:
  // constructor - parameters are upper-left and lower
  // right corners
  Rectangle(const Point &ul, const Point &lr);

  // copy constructor
  Rectangle(const Rectangle &other);

  // destructor
  ~Rectangle();

  // assignment
  Rectangle &operator=(const Rectangle &other);
 private:
  // Corners - privately owned copies of Point objects
  // allocated on the heap defining the corners of this
  // Rectangle.
  // Not shared with other Rectangles or other objects.
  Point * ul_;   // upper-left corner
  Point * lr_;   // lower-right corner
};
```

(Header guards omitted to save space. Feel free to detach this page for reference while you answer the rest of the question on the next page.)

Question 5. (cont) Here is the implementation of the basic constructor in `Rectangle.cc` to give a little better idea of how the instance variables are used.

```
Rectangle::Rectangle(const Point &ul, const Point &lr) {
  ul_ = new Point(ul);    // uses Point copy construcor
  lr_ = new Point(lr);
}
```

Your job is to provide implementations of the destructor and assignment operators for class `Rectangle`. The destructor is easy. Assignment is harder because you have to be careful about the differences between pointers and references, as well as the usual problems implementing assignment, dealing with heap storage and so forth.

(a) (6 points) Give an implementation of the destructor for class `Rectangle`.

```
Rectangle::~Rectangle() {



}
```

(b) (12 points) Give an implementation of assignment for class `Rectangle`.

```
Rectangle &Rectangle::operator=(const Rectangle &other) {



}
```

Question 6. (6 points) Git. Suppose you enter a git status command and see the following output:

```
bash$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   LinkedList.c
        modified:   HashTable.c

no changes added to commit (use "git add" and/or "git commit -a")
```

What commands, if any, should you execute so that your Gitlab repository is up to date with current copies of all files? If no further commands are needed and the repository is up to date, simply answer “none”.