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. _____ / 44

2. _____ / 20

3. _____ / 15

4. _____ / 20

5. _____ / 1
Question 1. (44 points) C hacking – a question of several parts.

The next several pages are questions about a linked list of 2-D points. Each point is represented by a `Point` struct containing the point’s `x` and `y` coordinates:

```c
typedef struct point {  // a 2-D point
    double x;           // x and y coordinates
    double y;
} Point;
```

A list of points is a single-linked list, where each node contains a pointer to its associated `Point` struct and a pointer to the next node in the list, if there is one.

```c
typedef struct pt_node {  // node in a linked list of Points
    Point * pt;          // point owned by this node
    struct pt_node * next; // next list node, or NULL if none
} Ptnode;
```

All of the data in the list (points and nodes) is owned by the list and allocated on the heap. Point and node structs are created and initialized when they are added to the list.

Example: A list containing the points (1.0, 2.0) and (3.0, 4.0) would be drawn like this:

![Diagram of a linked list containing points (1.0, 2.0) and (3.0, 4.0)]

Answer the questions that use this list structure on the following pages. You can detach this page from the test for reference while you are working if that is convenient.
Question 1. (cont.) (a) (15 points) The distance of a point (x,y) from the origin is calculated as $\sqrt{x^2+y^2}$. Implement the function `max_distance` below so it calculates the distance from the origin of each point in the list and returns the largest distance found. If the list is empty, the maximum distance returned should be 0.0.

Restriction: for full credit your function must use an appropriate loop to traverse the list and may not use recursion or any additional functions besides `sqrt`.

Example: if the list contains the points (1, 2) and (3, 4), the max distance returned should be 5.0 ($= \sqrt{3^2+4^2}$).

```c
#include <math.h>    // for sqrt
#include <stdlib.h>  // for NULL

// return max distance from origin of any point in the list
// (return 0.0 if the list is empty)
double max_distance(Ptnode * list) {
```

}`
Question 1. (cont.) (b) (15 points) Complete the definition of function free_list below so that it frees (returns to the heap) all of the data belonging to the list that is its argument. After freeing the list nodes and points, the function should set the list pointer supplied as its argument to NULL.

Restriction: as with part (a), for full credit your function must use an appropriate loop to traverse the list and may not use recursion or any additional functions besides free.

```c
#include <stdlib.h>  // for free, NULL

// free list and contents, then set *list to NULL
void free_list(Ptnode ** list) {
```
Question 1. (cont.) (14 points) One of the summer interns who has not taken CSE333 was asked to write a function to add the value of a new Point to the front of a list. Alas, the code has lots of problems. Indicate on the code below the changes needed so it will work *exactly* as specified. You *may not* change the specification of the function, the parameters, or the return type. You may add, remove, rearrange, or alter statements in the body of the function as needed.

// Add a new Point to the front of the list lst, with the // new point’s (x,y) values taken from parameter pt. // Update lst to point to the new node added at the front. // Return 1 if successful and 0 if not.

int push_front(Point pt, Ptnode ** lst) {

    Ptnode * node = (Ptnode*)malloc(sizeof(Ptnode*));

    Point * npt = (Point*)malloc(sizeof(Point*));

    node->pt = &pt;

    node->next = lst;

    lst = node;

    if (npt == NULL || node == NULL) {

        return 0;

    }

    return lst;
}
Question 2. (20 points) Yet another ghastly C program. As is traditional, it does compile and execute with no warnings or errors:

```c
void modifyPtr(int *ptr, int **dblPtr) {
    int *q = ptr;
    *ptr = 5;
    ptr++;
    *ptr = 6;
    *(q + 2) = 7;
    *((*dblPtr) - 1) = 8;
    // -->HERE<-- //
}

int main(int argc, char **argv) {
    int arr[4] = {1, 2, 3, 4};
    int *p = arr + 1;
    modifyPtr(&arr[0], &p);
    printf("%d %d %d %d %d\n",
           arr[0], arr[1], arr[2], arr[3], *p);
    return 0;
}
```

(a) Draw a boxes ‘n arrows diagram showing state of memory when control reaches the comment containing -->HERE<-- right before executing the return statement in function modifyPtr. Your diagram should have two boxes showing the stack frames for functions main and modifyPtr. The stack frames should include values of integer variables and an arrow from each pointer to the location that it references. Then answer part (b) at the bottom of the page.

(b) What output does this program produce when it is executed?
Question 3. (15 points) Make and program building. We have a program that consists of a header file f.h, an implementation file f.c, and a main program main.c. Normally we could use the following Makefile to build the program mumble from these source files:

mumble: main.o f.o

    gcc -o mumble main.o f.o

main.o: main.c f.h

    gcc -c -o main.o main.c

f.o: f.c f.h

    gcc -c -o f.o f.c

Unfortunately we need to build this program on a new experimental system that does not yet have a gcc command(!). Fortunately, it does have the three programs that gcc runs behind the scenes to build a program: the preprocessor cpp, the compiler itself cc1, and the loader ld. These commands work as follows:

cpp outfile infile – preprocess the program read from infile (say f.c) and write the preprocessed results to outfile (by convention named f.i if the input file is f.c).

ccl outfile infile – compile the already-preprocessed file infile (say f.i) and write the compiled code to the object file outfile (a .o file, for example f.o)

ld outfile infiles... – read the one or more .o files listed as infiles and write an executable file named outfile (mumble in our example above).

For this problem, show the changes that would be needed in the above Makefile so it would build the program as before, but using cpp, ccl, and ld instead of gcc. You can add or delete Makefile rules and Linux commands as needed. The resulting Makefile should only recompile/preprocess/load files when necessary, as is the case with the original one. Note that cpp and ccl only preprocess or compile a single file at a time, so you can’t, for instance, preprocess all of the .c files with a single command. Also (if it helps), remember that a single Makefile rule can have more than one command on successive lines to bring a file up to date.
Question 4. (20 points) In one of the exercises we implemented a C++ class `Vector` that represented 3-D points and supplied various operations on them. A `Vector` had the following representation:

```cpp
class Vector {
    ...
    private:
        double x_, y_, z_; // x, y, and z magnitudes of Vector
};
```

The operations provided included constructors (`Vector()`, `Vector(x, y, z)` and copy constructor); assignment; addition and subtraction, which produced new `Vector` values; scalar and dot-product multiplication; and stream output.

For this problem, give the declaration and implementation of a new `operator+=` for Vectors. If `u` is the Vector `(a,b,c)` and `v` is the Vector `(i,j,k)`, then `u+=v` should modify vector `u` so it has the value `(a+i, b+j, c+k)`. Note that `+=` is almost the same as regular assignment, except it computes a new value for its left operand instead of copying the right operand without changes. Self-updates like `u+=u`, and chained updates like `u+=v+=w` should also work properly.

(a) (6 points) Give the correct declaration (function prototype) of `operator+=` that should be added to class `Vector` in the header file `Vector.h`. (This is only one line)

```cpp
(a) (6 points) Give the correct declaration (function prototype) of operator+= that should be added to class Vector in the header file Vector.h. (This is only one line)
```

(b) (14 points) Give the full implementation of `operator+=` that should be added to file `Vector.cc`. 

```cpp
(b) (14 points) Give the full implementation of operator+= that should be added to file Vector.cc.
```
Question 5. (1 point) The best reason to take CSE 333 during the summer is (circle the letter of the best answer; all thoughtful, honest answers get full credit):

(a) A 9:40 class makes me wake up in the morning instead of sleeping till mid-afternoon.

(b) Hacking is much more fun than hiking.

(c) Next year I’ll get that Google internship after taking this class!

(d) I’d rather be in an air-conditioned, dark basement room instead of out in the bright, hot sunlight.

(e) Learning C is better than sitting on the shore by the sea.

(f) I get to finally figure out what all the funny symbols mean (% & : ! ->).

(g) gdb is a better video game than anything else I’ve got.

(h) Gotta get it out of the way eventually, so why not now?

(i) Best reason? Can’t think of one, but please give me my free point anyway.

(j) I do have a good reason, but I’d rather not say. Please give me my free point.

(k) You didn’t include my reason, it’s ____________________________________