# CSE 374 Final Exam

June 7, 2017

Name _____Answer Key_____ Id # ____0000000_____

There are 8 questions worth a total of 110 points (1 point per minute), and 1 extra credit question worth very few points. Please budget your time so you get to all of the questions, only attempt the extra credit if you have time left. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc.

Many of the questions have short solutions, even if the question is somewhat long.

If you don't remember the exact syntax of something, do the best you can, and make it clear you understand what you're trying to do. We will make allowances when grading.

Relax, you are here to learn. Please wait to turn the page until everyone is told to begin.

Score _____ / 110

1. _____ / 9
2. _____ / 11
3. _____ / 12
4. _____ / 15
5. _____ / 21
6. _____ / 15
7. _____ / 14
8. _____ / 13
9. _____ / ??

# Short Answer

1. **(3 Points each)**
a) Explain what happens if (in C) you return a pointer to some stack allocated data.

Stack allocated data is released when a function returns, so a pointer to it becomes a dangling pointer. (Full credit if they say something along these lines)

b) Explain why (in C) it's hard to test whether two signed integer values will overflow when added together.

Testing if they overflow by adding them to see if they overflowed is using undefined behavior. (Full credit if they mention undefined behavior, and say something about using it to test for it)

c) Briefly explain why the `make` tool is necessary, instead of just writing a shell script for every project. I.e. what part of what `make` does is not so easy to write in a few lines in bash?

make tracks dependencies, and only rebuilds the parts of your program that need rebuilding. This is non-trivial, and is why make is a tool. (Full credit if they mention dependencies and correct partial builds)

2. On your first day at a new job, your boss tells you to write a function, and they give you a specification: “Write me any correct function `sort`, such that `sort` takes in a list of numbers, and returns a list of numbers in ascending order”.

a) **(6 Points)** Explain (in English) an implementation of `sort` which meets the given specification, but is not what we would think of as a typical sorting function.

A sort function which always returns the list [1,2] meets the specification. (Full credit for any sorting algorithm that meets the English spec above, but doesn’t actually sort its input)

b) **(5 Points)** How would you change the given specification to match your intuition about what a function named `sort` should do?

The spec must also mention that the output list must be a permutation of the input list. Logically equivalent is: Every number must occur the same number of times in the input and output.

3. **Correct C, Compile Error** or **Might Crash**

**(6 Points each)** For each of the code snippets below, circle the part that *might crash*, circle the part which will be a *compile error*, or Write CORRECT below it. If you circle some part, write CRASH or COMPILE ERROR next to it to indicate which. For part b, `%zu` is the correct format string for the return type of `strlen` (unsigned integer).

a)

```
#include <stdlib.h>

struct header {
    int size;
};

int main() {
    struct header* x = (struct header*)malloc(sizeof(struct header)+512);
    x->size = 512;
    free(x); //make sure no leaks
    return 0;
}
```

CORRECT

b)

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

int main() {
    char x[5];
    x[0] = 'H';
    x[1] = 'e';
    x[2] = 'l';
    x[3] = 'l';
    x[4] = 'o';
    printf("length is %zu\n", strlen(x)); CRASH
    return 0;
}
```

4. **(15 Points)** Write a function `same_class` in C++ which takes pointers to two C++ objects as arguments, and returns an integer. It should return 1 if the objects are both objects of the same class, and 0 if they are of different classes. You may assume that all classes in question use virtual methods.

For example, if class B is a subclass of class A, you can call same_class like this:

```
A* x = new A();
A* y = new B();
A* z = new B();

if (same_class((void*)x, (void*)y)) {
    cout << "it's busted\n"; //shouldn't happen
}
if (same_class((void*)y, (void*)z)) {
    cout << "if this line is the only output, it worked\n";
}
```

Hint: Use what you know about the layout of an object in memory, and how objects are implemented in C++
Hint: While this question is technically about C++, the code you write will look a lot like C code

```
//in order to support pointers to all types of objects,
//cast to void* before calling
int same_class(void* a, void* b) {
    return (*((void**)a)) == (*((void**)b));


    //Full credit for a solution which compiles, dereferences the
pointers, and compares their equality
    //Partial credit (12/15) if solution doesn't compile due to
forgotten cast, but does everything else
    //Note: The cast to void** is necessary, as you can't dereference a
void*. Any other double pointer type would have worked as well, e.g.
int** or float**




}
```

5.

```
typedef struct node {
    int max_children;
    char* data;
    struct node** children;
} node;
```

Above is a struct declaration for a node in a tree. Each node contains `data`, a C style string, as well as some children. The integer `max_children` signifies how long the `children` array is, though each entry in that array can be an actual child or NULL.

a) **(12 Points)** Write a `new_node` function which allocates a new node on the heap, and returns a pointer to it. This function should take a parameter to indicate the maximum number of children for this new node to have, but this node should not have any actual children initially. This function should also take (as an argument) the data to initialize this node with, but make sure the node will maintain its data if the memory for the argument is reused later in the program. You may assume all the necessary headers are already included.

```
node* new_node(int max_child, char* data) {

    node* n = (node*)malloc(sizeof(node));
    assert(n);
    n->max_children = max_child;

    n->data = (char*)malloc((strlen(data) + 1) * sizeof(char));
    assert(n->data);

    strcpy(n->data, data);

    n->children = (node**)malloc(sizeof(node*) * max_child);
    assert(n->children);

    int i;
    for (i = 0; i < max_child; i++) {
      n->children[i] = NULL;
    }

    return n;
}
```

b) **(9 Points)** Write a `release` method, which takes a pointer to a node allocated on the heap, and frees all heap memory used by it or its children. By calling `release` once, all heap memory used by an entire tree should be freed.

```
void release(node* n) {
    if (n == NULL) {
     return;
    }
    int i;
    for (i = 0; i < n->max_children; i++) {
     release(n->children[i]);
    }
    free(n->data);
    free(n->children);
    free(n);
}
```

6. **(15 Points)** Sometimes a single int in C cannot represent very large numbers. In this case we can choose different representations for our numbers, depending on our needs. For this problem, consider the representation of an int array where each int is between 0 and 9, and thus represents 1 digit of the overall number. In order to record the end of the number, we put -1 in the last int. For instance, the array [3, 2, 5, 0, 0, 9, 2, 1, 5, 8, 7, -1] represents the number 32500921587.

Write a C function `numstring` which, given an int array representing a number in this fashion, returns a heap allocated string with the digits of the number. Your code may assume that every integer in the array is between -1 and 9 inclusive, that -1 is the last element, and that no element before -1 is negative (no need to check for illegal inputs).

Hint: Recall that the ascii value of '0' is 48, '1' is 49, etc. for all the digits. Thus, if you have an int x, (char)(x + 48) will give the character representing that digit.

```
char* numstring(int* arr) {
    int len = 0;
    while (arr[len] != -1) {
     len++;
    }

    char* str = (char*)malloc((len + 1) * sizeof(char));

    assert(str);

    int i;
    for (i = 0; i < len; i++) {
     str[i] = (char)(arr[i] + 48);
    }
    str[len] = '\0';

    return str;
}
```

7. **(14 Points)**

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

void copy_string(char *buf, char *src) {
      // Well, I prefer apple than berry
      // change "berry" to "apple"
      char *food = "apple"
      size_t i = 0;
      for (; i < 5; i++) {
            src[i + 7] = food[i];
      }
      i = 0;
      for (; i < sizeof(src); i++) {
            buf[i] = src[i]
      }
}


int main(int args, char **argv) {
      char *src = "I love berry";
      char *buf;
      copyString(buf, src);
      printf("%s\n", buf);
}
```

Above is a copy_string function written by a programmer who has an opinion about berries. However, this programmer has left some errors in their code. Cross out any errors you find, and write in the correct code next to them. Do not change the overall structure of the program, but rather fix each erroneous line independently.

See Next Page for fixed code

```c
#include <string.h> //FIX3

void copy_string(char *buf, char *src) {
    // Well, I prefer apple than berry
    // change "berry" to "apple"
    char *food = "apple";//FIX 1
    size_t i = 0;
    for (; i < 5; i++) {
            buf[i + 7] = food[i];
    }
    i = 0;
    for (; i < strlen(src) - 5; i++) { //FIX 2
            buf[i] = src[i]; //FIX 4
    }

}
int main(int args, char **argv) {
    char *src = "I love berry";
    char buf[13];
    copy_string(buf, src); //FIX 5
    printf("%s\n", buf);
}
```

8. A prominent internet search company has hired you to work out some bugs in their self driving cars. Suppose we have some snippet of code that sensors may use to control the speed of the car. Each piece of code may run on separate threads. The global variables `speed` and `brake` are used to control the car.

```
int speed; //set to desired speed
bool brake; //set to true to brake the car

void accelerate () {
    if (speed < 60) {
        if (!brake) {
            speed = 60;
        }
    }
}

void avoid_crash () {
    if (!brake) {
        if (speed > 30) {
            brake = true;
        }
    }
}
```

a) **(5 Points)** Suppose the self driving car is travelling down the road, and the speed limit increases, causing the `accelerate` function to be called. At the same time, a car merges in front of us, causing the `avoid_crash` function to be called (on a different thread). Explain 2 different possibilities for what could happen in that situation.

- Car accelerates to 60 (crashing into car in front)
- Car brakes
- Car accelerates and brakes at the same time

b) **(4 Points)** Now suppose another engineer tries to fix the problem by adding locks to the code as shown below:

```
int speed; //set to desired speed
bool brake; //set to true to brake the car
Lock speed_lock;
Lock brake_lock;

void accelerate () {
    speed_lock.acquire();
    if (speed < 60) {
        brake_lock.acquire();
        if (!brake) {
            speed = 60;
        }
        brake_lock.release();
    }
    speed_lock.release();
}

void avoid_crash () {
    brake_lock.acquire();
    if (!brake) {
        speed_lock.acquire();
        if (speed > 30) {
            brake = true;
        }
        speed_lock.release();
    }
    brake_lock.release();
}
```

Why might this solution not work?

We could have a situation where accelerate has the speed_lock, and avoid_crash has the brake_lock. Each will wait on the other, and thus we will have Deadlock.

c) **(4 Points)** What might be a better solution?
Use a single lock, or acquire the locks in a consistent order.

9) **DON’T ATTEMPT UNTIL FINISHED WITH THE REST OF THE TEST** Extra Credit: Same as problem 6, but each integer in the array is between -1 and 99999, representing 5 digits. Additionally, your input is a single int* which may or may not be NULL, and may contain integers which are outside of bounds. If given an invalid input (such as NULL or a malformed array), your function should allocate no memory on the heap and return NULL.