# CS 2150 Final Exam

## Name _____

You MUST write your e-mail ID on **EACH** page and put your name on the top of this page, too.

If you are still writing when "pens down" is called, your exam will be ripped up and not graded – sorry to have to be strict on this!

There are 10 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points.

**Answers for the short-answer questions should not exceed about 20 words; if your answer is too long (say, more than 30 words), you will get a zero for that question!**

This exam is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

_____

_____

_____

_____

_____

*Serious error.*
*All shortcuts have disappeared.*
*Screen. Mind. Both are blank.*

## Page 2: Miscellaneous

1. [2 points] Does the following C++ code run to completion, seg-fault, or fail to compile? If it fails to compiles or seg-faults, then explain why. If it runs to completion, what does it print out?

```
//assume #includes and using
//namespace appear up here somewhere
void myFunc(int* &x){
  *x = 2;
  x = new int(3);
  *x = 4;
}
```

```
int main(){
  int* y = new int(1);
  myFunc(y);
  cout << *y << endl;
  return 0;
}
```

2. [4 points] What is the output of the following C++ code? This code compiles (with several warnings) and runs without a seg-fault. Hint: Remember that arrays in C++ are stored in row-major order.

```
//assume #includes and using namespace appear up here somewhere
int main(){
  int a[3][4] = {0}; //fill the array with 0s
  a[1][4] = 6;
  a[1][5] = 12;
  a[2][1] = 5;
  cout << a[1][4] << a[2][0] << a[1][5] << a[2][2] << endl;
  return 0;
}
```

3. [2 points] How many unique values (i.e., memory addresses) can a pointer in C++ (e.g., int* x;) store? List any assumption(s) you make.

4. [3 points] Convert the value 0xC2E08000 from IEEE 754 floating point notation into decimal. Leave your answer as an expression in base-2 scientific notation (e.g., you can leave -5 as $-(1+\frac{1}{4})*2^2$)

## Page 3: True / False

5. [11 points] For each proposition below, circle true if it is always true, and false otherwise.

| | | |
|---|---|---|
| True | False | Amortized runtime is the same as expected runtime. |
| True | False | A heap is a particular type of binary search tree. |
| True | False | A low load factor is a more efficient use of memory. |
| True | False | Big-Omega represents the best-case runtime. |
| True | False | Java and C++ both allow multiple inheritance. |
| True | False | Statically allocated memory is stored in the heap. |
| True | False | A balanced Huffman Tree will lead to a higher compression ratio. |
| True | False | While helpful for visual representation, it is not always efficient to use node objects with pointers to represent a tree. |
| True | False | IBCM is Turing-complete, with the minor caveat that you would need more memory to do most things. |
| True | False | It is feasible to use the language whitespace to code up a polyglot. |
| True | False | Double hashing has the same worst-case time complexity for all operations as linear probing. |
| True | False | When printing out an expression tree in postfix order, parentheses are NOT necessary to indicate order of operations. |
| True | False | All trees are graphs. |
| True | False | The following x86 instruction is valid: mov rax, [4*rsi+rdx+8]. |
| True | False | The following x86 instruction is valid: mov [rax], [rsi+4]. |
| True | False | In x86, all parameters are stored in registers. |
| True | False | Topological sort can be used on an undirected graph. |
| True | False | Linked lists have better spatial locality than arrays |
| True | False | Adjacency matrices require less memory than adjacency lists for large, sparse graphs. |
| True | False | There is no known solution to the Traveling Salesman Problem, only approximations. |
| True | False | "False" is the correct answer an odd number of times in this page. |

## Page 4: IBCM / Assembly

6. [2 points] Use what you know about assembly to describe how a **stack overflow** occurs. Be as precise as you can.

7. [2 points] Using what you know about assembly and how registers/memory work, answer the following question: In the 32-bit calling convention, parameters were always placed on the stack (no matter what), but the 64-bit convention places up to six parameters in registers. Name one advantage and one disadvantage of moving to this register based scheme.

8. [4 points] The following C++ function recursively computes the binomial coefficient $\binom{n}{k}$. The assembly code below that implements this function in x86, but there is a bug in the code! Do you remember this from exam 2!? Find the bug in the assembly and briefly explain the problem.

```
int binomialCoef(int n, int k) {
  if (k == 0 || n == k) return 1;
  return binomialCoef(n-1, k) + binomialCoef(n-1, k-1);
}
```

```
binomialCoef:
  push rbx
  cmp rsi, 0
  je   done
  cmp  rsi, rdi
  je done
  dec  rdi
  call binomialCoef
  mov rbx, rax
  dec rsi
  call binomialCoef
  add  rax, rbx
  jmp  end
done:
  mov rax, 1
end:
  pop rbx
  ret
```

## Page 5: Hash Tables

9. [6 points] Insert the following integers into the hash table below (assume the integer is both the key and the value here). Use double-hashing to resolve collisions. Your hash function is $h(x) = (x * 2)$. Your second hash function is $h2(x) = (x + 1)$. Notice that the table size here is 10, so make sure you mod by 10 after you evaluate the hashed value.

   Insert: 25, 12, 13, 5, 2, 1

| Index | Value |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

10. [2 points] What is the big-theta runtime of rehashing a hash table? Choose any collision resolution strategy for your answer, but please make it clear which strategy you've chosen. Explain your answer, in one or two sentences.

## Page 6: Heaps are awesome

Suppose that instead of using a binary heap (as we saw in class), we used a **ternary heap** instead. A **ternary heap** works exactly the same as the one seen in class, except each "node" has up to three children instead of just two. We could still store a heap in an array, as in the example below. For this example, A is the root node, which has three children (C, D, and K), and so forth. Answer the following questions regarding ternary heaps.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| NULL | A | C | D | K | L | P | Z | O | H | T | Q |

11. [2 points] Draw the ternary heap above as a tree.

12. [2 points] Assume the DeleteMin() function works analogously to a binary heap. What is the worst-case big-theta runtime? Briefly explain why?

13. [3 points] Given a current index $i$ in the array, provide three expressions that will give you the left, middle, and right children respectively of the node at index $i$.

`left(i) =` `middle(i) =` `right(i) =`

14. [4 points] Complete the function below that accepts an array of integers $a$ and a length of that array $n$ and prints out the values in $a$ in sorted order. Your function may not alter the array $a$ in any way, and can only allocate one **TernaryHeap** variable. Assume the heap has a default constructor, and the following methods: void insert(int priority), int findMin(), int deleteMin(), and bool size().

```
void printSorted(int a[], int n){
```

## Page 7: Cords

This page contains information about a data structure we did not study in class called a cord. Read this page and then answer the questions on the next page. There is nothing to do on this page but read this information.

A Rope or Cord is a data structure used as an alternative to array-based strings. They allow for the swift concatenation and splitting of strings. A text editing program may use a rope to represent the text being edited, so that operations such as insertion, deletion, and random access can be done efficiently. Each cord is meant to represent a single string. For example: "Hello_my_name _is_Simon"

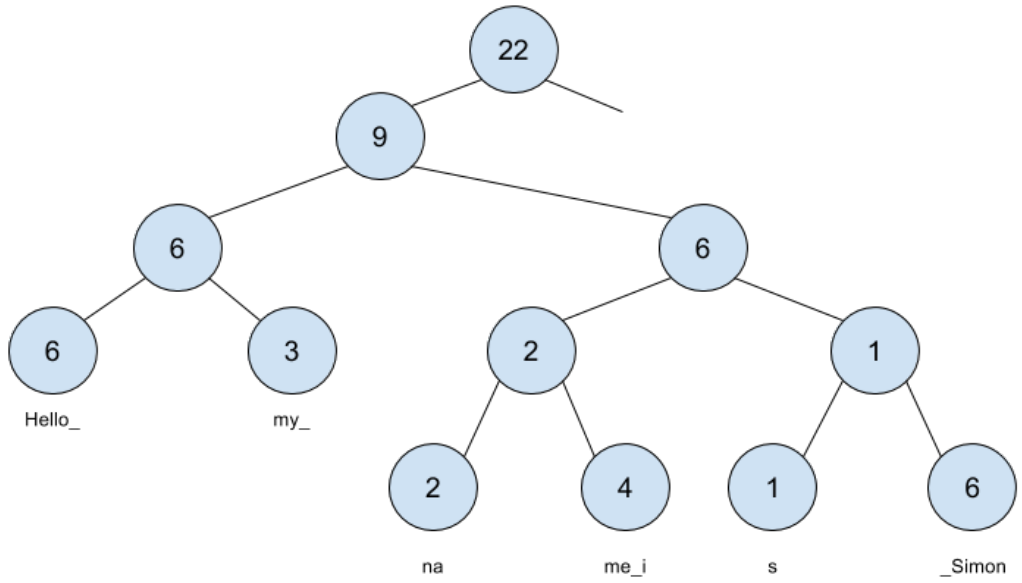
Cords are a special type of **balanced binary tree** in which:

- Each leaf node contains a substring of the overall string
- All nodes (root, interior, and leaf nodes) contain:
  - A left node pointer
  - A right node pointer
  - a integer weight, equal to the length of its substring plus the sum of all leaf nodes' weight in its left subtree. In other words: for leaf nodes this means that the weight is the length of the substring it holds; for non-leaf nodes, this means that the weight is the total string length in its left subtree.

Here is an example struct for a CordNode:

```
struct CordNode {
   CordNode* left;
   CordNode* right;
   unsigned int weight;
   string s;
};
```

Here is one possible Cord representing the string: "Hello_my_name_is_Simon"



## Page 8: Questions about cords

15. [3 points] Write a recursive function in C++ that returns the ith character of this string. Your method will take in a reference to a CordNode (initially the root) and the index to find (initially i):

    public char GetCharAt(& CORDNODE cn, int index){

16. [1 points] What is the worst-case Big-Theta time complexity of your GetCharAt function in terms of n where n is the number of nodes in the tree?

17. [2 points] Does using a cord to find the character at index $i$ of a string have a better or worse runtime than simply using a C++ string object? Explain your answer.

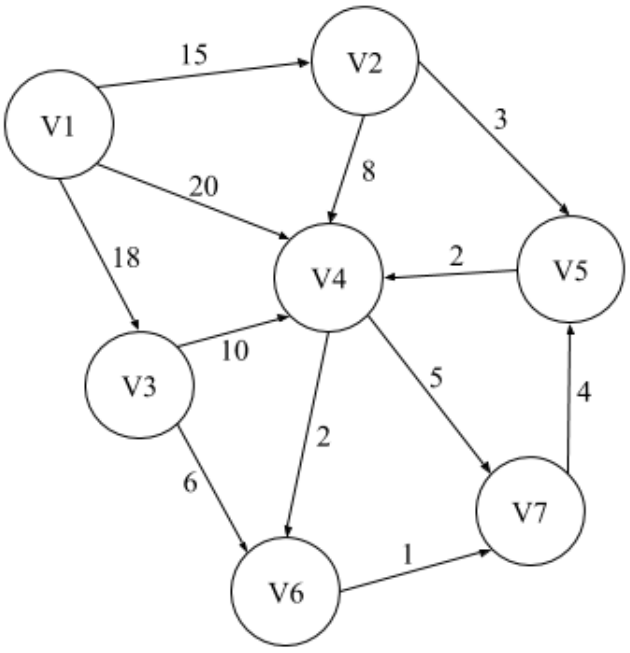
## Page 9: Graphs

18. [2 points] Why might you want to use an adjacency matrix as opposed to an adjacency list to represent a graph?

19. [3 points] Briefly explain why the runtime of *Topological Sort* is $\Theta(E + V)$. I'm looking for the intuition on why the runtime is $\Theta(E + V)$ and not $\Theta(EV)$.

20. [6 points] Run Dijkstra's algorithm on the following graph with $V1$ as the start node. Fill in the chart to show the execution of the algorithm. If a value in the chart is updated during execution, put a single line through it and write the updated value to the right of the old one.

| Node | Known | Dist | Path |
|---|---|---|---|
| V1 (S) | | $\infty$ | |
| V2 | | $\infty$ | |
| V3 | | $\infty$ | |
| V4 | | $\infty$ | |
| V5 | | $\infty$ | |
| V6 | | $\infty$ | |
| V7 | | $\infty$ | |



## Page 10: From the Lab Tutorials

21. [3 points] Given a program MyProgram.cpp select the set of correct commands from either column to compile it with debugging symbols, run the debugger, set a breakpoint at line 5, examine variable x, and then continue and exit the debugger. Circle one option from each rown (e.g., circle either 1a OR 1b, etc.). lldb and gdb should work exactly the same way for this set of commands.

| | |
|---|---|
| 1a. clang++ -g MyProgram.cpp | 1b. clang++ -S MyProgram.cpp -o MyProgram.o |
| 2a. gdb a.out | 2b. gdb MyProgram.cpp |
| 3a. breakpoint *5 | 3b. breakpoint 5 |
| 4a. start a.out | 4b. run |
| 5a. info x | 5b. print x |
| 6a. continue | 6b. restart |
| 7a. end | 7b. quit |

22. [2 points] List two things from the C language that are different in C++

23. [1 points] When creating a Makefile, what is the primary advantage of listing the project's *dependencies*?

24. [4 points] List any *eight* Unix commands that you used during this course.