# Computer Science Foundation Exam

## May 17, 2025

## Section A: BASIC DATA STRUCTURES

**NO books, notes, or calculators may be used,**
**and you must work entirely on your own.**

**PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME**

**Last Name:** ________________________________

**First Name:** ________________________________

**UCFID:** ____________________________________

**COT 3960 Term:** _____________________________

| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 5 | ALG | |
| 2 | 10 | DSN | |
| 3 | 10 | ALG | |
| TOTAL | 25 | ---- | |

**You must do all 3 problems in this section of the exam.**

**Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.**

**1)** (5 pts) ALG (Dynamic Memory Management in C)

Consider the following code that attempts to allocate a new array with the same number of rows as arr, but triple the columns, copy the values from arr into the new array (in the same slots), free the dynamically allocated memory pointed to by arr and return a pointer to the new dynamically allocated array (newarr):

```
int ** tripleCols(int ** arr, int rows, int cols)
{
    int ** newarr = malloc(sizeof(int *) * rows * 3); // point a

    for(int r = 0; r < rows; ++r)
    {
        newarr[r] = malloc(sizeof(int) * cols * 3);
        newarr[r] = arr[r]; // point b
        free(arr[r]);
    }

    free(arr);
    return newarr;
}
```

The key errors in the code are at points a and b, noted in the comments.

(a) (1 pt) What is the fix for the line of code for point a? (This one's simple, so no need to write out the new line of code, just describe the fix.)

(b) (2 pts) Why is the line of code for point b incorrect conceptually?

(c) (2 pts) Write two lines of code to replace this one line of code so that the function will work as planned.

**2)** (10 pts) DSN (Linked Lists)

We can store an integer in a linked list of nodes, where each node stores digit, in reverse order. For example, the integer 2163 would be stored in the linked list 3 → 6 → 1 → 2. Using the node struct shown below that is used to store numbers in this manner, write a **recursive** compareTo function that takes in pointers to two integer stored in this manner and returns a negative integer if the number in the list pointed to by num1 is less than the number in the list pointed to by num2, 0 if the two respective numbers are equal, or a positive integer if the number in the list pointed to by num1 is larger than the number in the list pointed to by num2. For example, compareTo(3 → 6 → 1 → 2, 4 → 6 → 1 → 2) should return a negative integer and compareTo(3 → 6 → 1 → 2, 9 → 9 → 9 → 1) should return a positive integer.

```
typedef struct node {
    int digit;
    struct node* next;
} node;


int compareTo(node* num1, node* num2) {













}
```

**3)** (10 pts) ALG (Stack)

Convert the following infix expression to postfix using a stack. Show the contents of the stack at the indicated points (A, B, and C) in the infix expression.

```
               A                   B           C
9 * 2 + 4 - 3 / (6 * (4 – 3)) + 1 * (7 - 4) / (5 + 6)
```

A B C

Note: A indicates the location in the expression **AFTER** the division operator and before the open parenthesis. B indicates the location in the expression **AFTER** the multiplication and before the open parenthesis. C indicates the location in the expression **AFTER** the open parenthesis and before the value 5.

Resulting postfix expression:

**Note: There are exactly the correct number of boxes above. These should be filled with 12 numbers and 11 operators.**

# Computer Science Foundation Exam

## May 17, 2025

## Section B: ADVANCED DATA STRUCTURES

**NO books, notes, or calculators may be used,**
**and you must work entirely on your own.**

**PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME**

**Last Name:** ________________________________

**First Name:** ________________________________

**UCFID:** ______________________________________

| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 10 | ALG | |
| 2 | 5 | ALG | |
| 3 | 10 | DSN | |
| TOTAL | 25 | ---- | |

**You must do all 3 problems in this section of the exam.**

**Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.**

**1)** (10 pts) ALG (Binary Trees)

The following code, when passed the root of a binary tree and returns a result calculated by adding the values of some nodes and subtracting the values of others. If the function whatDoesItDo is called on the root of the binary tree shown below, for each value in the tree, indicate whether it gets added (A) or subtracted (S), **with respect to the original call on the root of the tree below.** No need to state the final return value. (**Grading Note:** +1 for each correct slot, +0 for each slot left blank, -1 for each incorrect slot, minimum score is 0.)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;

int whatDoesItDo(bintreenode* root) {

    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return root->data;

    if (root->left == NULL) return root->data + whatDoesItDo(root->right);
    if (root->right == NULL) return root->data + whatDoesItDo(root->left);

    if (root->left->data > root->right->data)
        return root->data + whatDoesItDo(root->left) - whatDoesItDo(root->right);

    return root->data + whatDoesItDo(root->right) - whatDoesItDo(root->left);
}
```

```
          root
           v
           30
         /     \
       16        22
      /  \      /  \
    19   18   40    25
         /    / \
        6    14  23
```

For each open slot, either write the letter 'A' for added, or 'S' for subtracted.

| | | | | | |
|---|---|---|---|---|---|
| 30 | ______ | 16 | ______ | 22 | ______ |
| 19 | ______ | 18 | ______ | 40 | ______ |
| 25 | ______ | 6 | ______ | 14 | ______ |
| 23 | ______ | | | | |

**2)** (5 pts) ALG (Hash Tables)

Consider a hash table that uses the **quadratic probing technique** with the following hash function f(x) = (3x+4)%11. (The hash table size is 11). If we insert the values 22, 11, 44, 32, 10, 21, and 33 into the table, in that order, show where these values would end up in the table.

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Value | | | | | | | | | | | |

**3)** (10 pts) DSN (Tries)

We are maintaining a Trie for predicting the next letter for a given string. The trie node struct and its properties are discussed below.

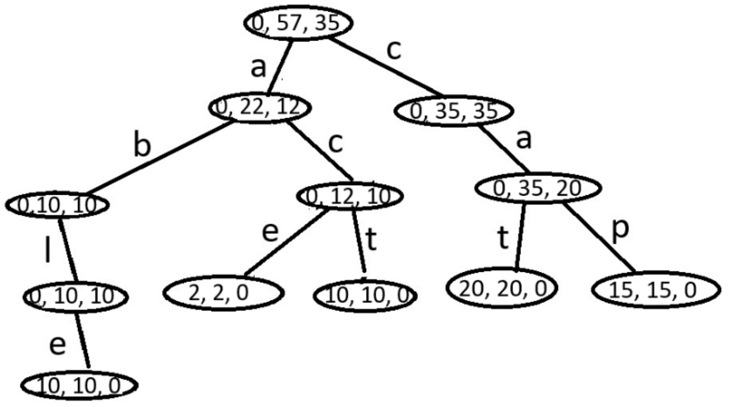
```
typedef struct trienode {
    int freq;
    int sum_prefix_freq;
    int cur_max_freq;
    struct trienode* next[26];
} trienode;
```

- **`freq:`** The frequency of the word represented by this node (i.e., how many times this specific word has been added to the dictionary). If this value is 0, it means that there is no word in the trie that ends at that node.
- **`sum_prefix_freq`**: The total frequency of all words in the dictionary that have this string as a prefix, including the string itself.
- **`cur_max_freq`**: The highest sum frequency among all child nodes of the current node.
- **`next[26]:`** These are typical child pointers of a trie node. This is an array of 26 pointers, each representing one of the possible next letters ('a' to 'z'). A pointer should be NULL if no words in the dictionary continue along that path. Typically, only a subset of these pointers will be active.

As an example, the following trie is constructed after inserting the following list of words and their frequency. The numbers inside a node represent its `freq, sum_prefix_freq, cur_max_freq,` respectively.

List of words and their frequency added to the trie.

**`cap 15, cat 20, act 10, able 10, ace 2,`**



Your goal is to **complete** the recursive function on the next page that receives the root of a trie, **t**, a string, **str**, and an integer, **k**, the current position in the string, and returns the most likely letter that follows the input string. **You may assume a unique next letter appears the most number of times** (cur_max_freq). If there is no string in the trie that **str** is a proper prefix for, then return the question mark character, '?'.

For example, if the string passed to the function is:

- predict(root, “a”, 0) should make a recursive call to predict(root->next[0], “a”, 1), which should then return ‘c’ because ‘c’ is the most likely letter to follow “a” for the sample trie.
- predict(root, “ab”, 0) will eventually return ‘l’ (lower case L) after two recursive calls.
- predict(root,“ac”, 0) will eventually return ‘t’ after two recursive calls.
- predict(root,“ace”, 0) will eventually return ‘?’ after three recursive calls because “ace” is not a proper prefix of any word in the trie.
- predict(root, “ap”, 0) will make one recursive call and then from there return ‘?’, since “ap” isn’t a prefix of any word in the trie. (In the code, this case is slightly different from the previous one.)

```
char predict(trienode *t, char *str, int k) {

    if (t == NULL) return ‘?’;

    if (k == strlen(str)) {

        // Checks if there is no string with this prefix.

        if ( ______________________________ )
            return ‘?’;

        // Looks through all possible next letters until it finds the one
        // has the most words that start with that prefix.
        // Note: part before the && is for short-circuiting to avoid null ptr.
        for (int i=0; i<26; i++) {

            if ( _____________________________ &&

                 __________________________________________________________ )
                     return (char)(‘a’+i);
        }
    }

    // We require a recursive call in this case.

    return ______________________________________________________________ ;

}
```

# Computer Science Foundation Exam

**May 17, 2025**

**Section C**

**ALGORITHM ANALYSIS**

**NO books, notes, or calculators may be used, and you must work entirely on your own.**

**PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME**

**Last Name:** ___________________________________

**First Name:** ___________________________________

**UCFID:** ________________________________________

| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 10 | ANL | |
| 2 | 5 | ANL | |
| 3 | 10 | ANL | |
| TOTAL | 25 | ---- | |

**You must do all 3 problems in this section of the exam.**

**Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.**

**1)** (10 pts) ANL (Algorithm Analysis)

Consider a n-bit binary counter, which starts with the binary representation of 0 and increments by 1 until it reaches the binary value of $2^n - 1$. For n = 3, the counter would start at 000, and then change as follows:

$$000 \rightarrow 00\underline{1} \rightarrow 0\underline{10} \rightarrow 01\underline{1} \rightarrow \underline{100} \rightarrow 10\underline{1} \rightarrow 1\underline{10} \rightarrow 11\underline{1}.$$

The underlined bits represent the ones that had to be changed. In particular, for this example, 1 + 2 + 1 + 3 + 1 + 2 + 1 = 11 bits were changed as the counter progressed from 0 to $2^n - 1$. Let f(n) equal the number of bits that are changed for an n-bit binary counter counting from 0 to $2^n - 1$. Find a closed-form formula for f(n). (For example, something like $f(n) = 2^{n-1} + 2$. A formula in terms of n without any sort of recursive function definition.) Show all of your work and put a box around your final answer.

**2)** (5 pts) ANL (Algorithm Analysis)

A O($\sqrt{n}$) search algorithm took 45 milliseconds to complete a search amongst $\boldsymbol{n}$ = 4 x $10^6$ entries. How long would it be expected for this algorithm execute a search amongst a database of $10^8$ entries, in milliseconds?

**3)** (10 pts) ANL (Summation)

Determine the following summation in terms of n, **in factorized form.** (Do NOT multiply the answer out into polynomial form. Note: Your answer should NOT have a fraction in it. It should just be terms that are multiplied.)

$$\sum_{i=1}^{2n-1} (i + 3i^2)$$

# Computer Science Foundation Exam

**May 17, 2025**

**Section D**

**ALGORITHMS**

**NO books, notes, or calculators may be used, and you must work entirely on your own.**

**PLEASE USE CAPITAL LETTERS IN WRITING YOUR NAME**

**Last Name:** ______________________________

**First Name:** ______________________________

**UCFID:** ______________________________

| Question # | Max Pts | Category | Score |
|---|---|---|---|
| 1 | 5 | DSN | |
| 2 | 10 | DSN | |
| 3 | 10 | DSN | |
| TOTAL | 25 | ---- | |

**You must do all 3 problems in this section of the exam.**

**Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.**

**1)** (5 pts) DSN (Recursive Coding)

Write a **recursive function** below so that it returns the maximum integer k such that base$^k \le$ ans. For example if base is 3 and ans is 123, then 4 should be returned since $3^4 = 81$ and $3^5 = 243$. The restricted bounds below are to ensure that integer overflow isn't an issue.

```
// Pre-condition: 1 < base <= 1000, 1 <= ans <= 1000000
int maxpowLTE(int base, int ans) {









}
```

**2)** (10 pts) DSN (Sorting)

Consider the problem of sorting a competition struct. A competition struct has three integer components: **`probSolved`**, **`totalTime`** and **`difficulty`**. One struct is greater than another if has more problems solved (**`probSolved`**) than another. If the problems solved is the same between two structs and the total time is less, then that struct is greater than the other. Finally, between two structs with the same number of problems solved and total time, if one has greater difficulty, it is greater than the other struct. If all three components are equal, the structs are equal and neither is greater than the other. Write a function called greaterThan, which takes in pointers to two competition structs and returns 1 if the struct pointed to by ptrA is greater than the struct pointed to by ptrB, according to these rules, and 0 otherwise. For example, (pS=3, tT = 200, d = 8) is greater than (3, 200, 7) but is NOT greater than (3, 199, 7). Rather (3, 199, 7) is greater than (3, 200, 8).

```
typedef struct competition {
    int probSolved;
    int totalTime;
    int difficulty;
} competition;

int greaterThan(competition* ptrA, competition* ptrB) {







}
```

**3)** (10 pts) DSN (Bitwise Operators)

User IDs are stored in a system as single integers in between 0 and $2^{25} - 1$, inclusive. Thus, each User ID can be viewed as a bitstring of length 25 (using an integer variables 25 least significant bits). When a new user ID is added, in order not to cause confusion, it's required that it differs in **at least three bits** compared to all other user IDs in the system. Write a function that takes in an array of current user IDs (**`curIDs`**), the length of that array (**`n`**), and a potential ID to be added (**`pid`**) and returns 1 if the potential ID can be added to the current list based on the previously given criteria, OR 0 if it can't be added if there exists a current ID with which the new ID differs in 2 or fewer bits. (For example, `10010000` differs with `00110000` in exactly 2 positions: $2^5$ and $2^7$. Thus, if the former was in the current ID list, the latter would NOT be an allowable password to add. **PLEASE DO NOT WRITE ANY AUXILIARY FUNCTIONS.**

```
int canBeAdded(int* curIDs, int n, int pid) {






}
```