CS 61C Summer 2022

Caroline, Justin, Peyrin Midterm

PRINT your name: _____, _____
(last) (first)

PRINT your student ID: _____

You have 110 minutes. There are 5 questions of varying credit (100 points total).

Question:	1	2	3	4	5	Total
Points:	20	25	20	25	10	100

For questions with **circular bubbles**, you may select only one choice.

- ○ Unselected option (completely unfilled)
- ● Only one selected option (completely filled)

For questions with **square checkboxes**, you may select one or more choices.

- ☐ You can select
- ■ multiple squares
- ■ (completely filled)

Anything you write that you ~~cross out~~ will not be graded. Anything you write outside the answer boxes will not be graded.

If an answer requires hex input, make sure you only use capitalized letters! For example, `0xDEADBEEF` instead of `0xdeadbeef`. Please include hex (`0x`) or binary (`0b`) prefixes in your answers unless otherwise specified. For all other bases, do not add any prefixes or suffixes.

Read the following honor code and sign your name.

> I understand that I may not collaborate with anyone else on this exam, or cheat in any way. I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that academic misconduct will be reported to the Center for Student Conduct and may further result in, at minimum, negative points on the exam.

SIGN your name: _____

Q1 *Potpourri* **(20 points)**

Q1.1 (6 points) Translate the following decimal numbers into 8-bit two's complement, unsigned, and sign-magnitude representations in the table below.

If a translation is not possible, please write "N/A". **Write your final answer in hexadecimal format, including the relevant prefix.**

Decimal	Two's Complement	Unsigned	Sign-Magnitude
128			
-12			

Q1.2 (2 points) Convert 83 to the following bases as an unsigned number. Remove any leading zeros.

Binary	Hex
0b	0x

Q1.3 (3 points) Which of the following representations have more than one representation of 0? Select all that apply.

- ☐ (A) Two's complement
- ☐ (B) Sign-magnitude
- ☐ (C) An IEEE-754 standard double-precision float
- ☐ (D) Bias notation
- ☐ (E) None of the above

Q1.4 (3 points) Which program resolves pseudoinstructions?

- ○ (A) Assembler
- ○ (B) Interpreter
- ○ (C) Linker
- ○ (D) Loader
- ○ (E) None of the above

Q1.5 (3 points) Which program can output pseudoinstructions?

- ○ (A) Assembler
- ○ (B) Compiler
- ○ (C) Linker
- ○ (D) Preprocessor
- ○ (E) None of the above

Q1.6 (3 points) Which program initializes registers to their default value?

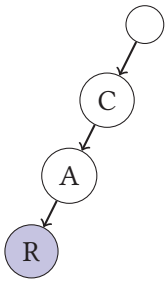
- ○ (A) Assembler
- ○ (B) Compiler
- ○ (C) Interpreter
- ○ (D) Linker
- ○ (E) None of the above

Q2 *Trie, Trie Again* **(25 points)**

In this problem, we will be implementing a trie. A trie is a data structure that stores strings in a tree-like structure, with each character in the string corresponding to a node. If two strings start with the same characters, the nodes for those characters will be shared between the two strings.

For each character in the string, our trie will create an `AlphaTrieNode` struct with two fields:

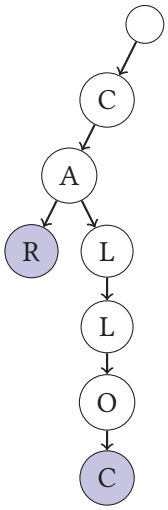
- A boolean `last` that indicates whether or not the character is the last character in a string. In the diagram, nodes with `last` set to true are shaded.
- An array `next` of 26 `AlphaTrieNode` pointers. Each element in the `next` array corresponds to a letter `a-z`. Each array element is either a pointer to the node corresponding to that letter, or `NULL` if there is no node for that letter.



This trie on the left stores one word "car".

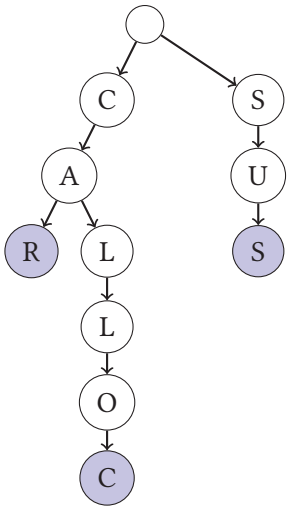
The `last` field is `false` in the C and A nodes, and `true` in the R node.

In the C node, `next[0]` should hold the address of the A node, and `next[1]`, `next[2]`, ..., `next[25]` should all be `NULL`.

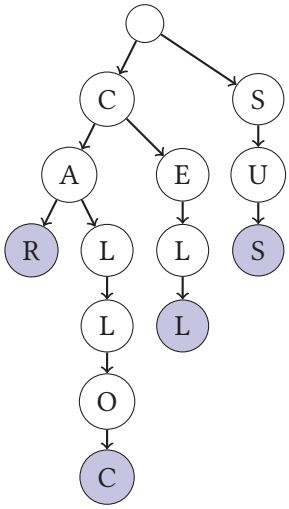


We want to insert "calloc" into the trie. Note that "car" and "calloc" both start with "ca—", so we don't create new nodes for those two characters. However, "—lloc" is different from "—r", so our trie creates new nodes for those four characters.

In the `A` node, `next[11]` (for L) and `next[17]` (for R) contain pointers, and all other pointers in the `next` array are `NULL`.

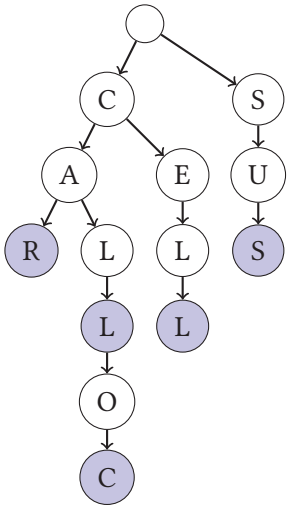


We want to insert "sus" in the trie. "sus" starts with a different character than the words so far, so we create new nodes for "s", "u", and "s", starting from the root node. Note that the second "s" corresponding to the last character of "sus" is shaded (`last` set to `true`).



We want to insert "cell" in the trie. "cell" starts with "c", but does not start with "ca—". We create new nodes for "—ell" and set `last` in the second "l" to be `true`.

Note that the "-ll-" in "calloc" and the "-ll" in "cell" are not shared, because the 2nd character of each word is different.



We want to insert "call" to our trie. "call" starts with the same letters as "calloc", so we don't need to add any nodes. However, even though the nodes are already there, there is nothing to indicate "call" is a word in the trie. We change this by updating the `last` field of the second "l" in "calloc" to be `true`. This indicates that "l" is now the end of a word.

For tries of `AlphaTrieNode`s, assume that:

1. Only lowercase letters (a-z) are supported. The ASCII values for these characters are [97,122], inclusive.
2. If multiple instances of the same string are inserted, it should not affect the trie.
3. The string argument `word` is a well-formatted string composed of only lowercase letters a-z.
4. The `node` argument is the root node of a properly initialized trie.

```
typedef struct AlphaTrieNode {
    bool last;
    struct AlphaTrieNode* next[26];
} AlphaTrieNode;

int main() {
    AlphaTrieNode* root = ... // instantiation of root (code not shown)

    // should insert "crewmate" into the trie
    insert(root, "crewmate");

    // should return false since "imposter" has not been added to the trie
    contains(root, "imposter");

    // should return true since "crewmate" is in the trie
    contains(root, "crewmate");

    return 0;
}
```

Implement the `contains` and `insert` functions below.

Q2.1 (5 points) `contains` takes in a trie root node (`node`), and a pointer to a string (`word`). It should return `true` if `word` is in the trie, and `false` otherwise.

```
1 bool contains(AlphaTrieNode* node, char* word) {
2   for (int i = 0; i < strlen(word); i++) {

3     int char_to_ascii = ____________________________________________;

4     if (node->_____________________________________________ == NULL) {

5       return ______________________________________________________;
6     }

7     node = _____________________________->_____________________________;
8   }

9   return node->_____________________________________________________;
10 }
```

Q2.2 (8 points) `insert` takes in a trie root node (`node`), and a pointer to a string (`word`). It should insert the word into the trie.

```
1 void insert(AlphaTrieNode* node, char* word) {
2   for (int i = 0; i < strlen(word); i++) {

3     int char_to_ascii = ____________________________________________;

4     if (node->_____________________________________________ == NULL) {

5       node->__________ = calloc(______, _____________________________);
6     }

7     node = node->_____________________________________________________;
8   }

9   node->_____________________________=_____________________________;
10 }
```

Consider an alternate trie implementation that supports all 256 ASCII characters instead of just 26 lowercase characters. We define a new struct, `ASCIITrieNode`, as follows:

```
typedef struct ASCIITrieNode {
    bool last;
    struct ASCIITrieNode* next[256];
} ASCIITrieNode;
```

We would like to write a function that converts a trie of `AlphaTrieNode`s to a trie of `ASCIITrieNode`s. The function should also free all `AlphaTrieNode`s in the process. You may assume that all `AlphaTrieNode`s are properly initialized.

Below, we have 3 implementations of this conversion function. For each implementation, determine whether or not it is a valid implementation. If the implementation is not valid, please provide a brief explanation (10 words or fewer).

Q2.3 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = malloc(sizeof(ASCIITrieNode));
    for (int i = 0; i < 26; i++) {
        new_node->next[i + 97] = convert(node->next[i]);
    }
    new_node->last = node->last;
    free(node);
    return new_node;
}
```

○ (A) Valid ○ (B) Invalid

Q2.4 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = calloc(1, sizeof(ASCIITrieNode));
    for (int i = 0; i < 26; i++) {
        new_node->next[i + 97] = convert(node->next[i]);
    }
    new_node->last = node->last;
    free(node);
    return new_node;
}
```

○ (A) Valid ○ (B) Invalid

Q2.5 (4 points)

```
ASCIITrieNode* convert(AlphaTrieNode* node) {
    if (node == NULL) {
        return NULL;
    }
    ASCIITrieNode* new_node = realloc(node, sizeof(ASCIITrieNode));
    for (int i = 0; i < 256; i++) {
        new_node->next[i] = NULL;
    }
    for (int j = 0; j < 26; j++) {
        new_node->next[j + 97] = convert(node->next[j]);
    }
    return new_node;
}
```

○ (A) Valid ○ (B) Invalid

Q3 *Can You Fix My Float?* **(20 points)**

Consider a 16-bit fixed point system with 1 sign bit, 5 integer bits, and 10 fraction bits. The five integer bits work just like a 5 bit unsigned integer. The 10 fraction bits continue where the integer bits left off, representing 2^{-1}, 2^{-2}, ..., 2^{-10}. For example, the bit representation `0b0 01101 1010000000` represents $(2^3 + 2^2 + 2^0) + (2^{-1} + 2^{-3}) = 13.625$.

In this question, we will compare this fixed-point system to a 16-bit floating point system that follows all conventions of IEEE-754 floating point numbers (including denorms, NaNs, etc.), with 5 bits of exponent and an exponent bias of -15.

Q3.1 (3 points) Write -22.375 in hex using the 16-bit floating point system described above.

`0x`

Q3.2 (3 points) Write -22.375 in hex using the 16-bit fixed point system described above.

`0x`

Q3.3 (3 points) How many numbers in the range $[16, 64)$ (including 16, excluding 64) can be represented by the floating point system described above?

Q3.4 (3 points) How many numbers in the range $[16, 64)$ (including 16, excluding 64) can be represented by the fixed point system described above?

Q3.5 (4 points) What is the smallest positive number representable by the above fixed point system but not the above floating point system?

Express your answer as a sum or difference of powers of two (e.g. $2^4 - 2^2 + 2^{-1}$).

Q3.6 (4 points) What is the largest positive number representable by both systems described above?

Express your answer as a sum or difference of powers of two (e.g. $2^4 - 2^2 + 2^{-1}$).

This page intentionally left (mostly) blank.
Please do not tear off any pages from the exam.

Q4 *Even Stevens* **(25 points)**

You are given a list of numbers to add up by your math professor. However, your professor doesn't like odd numbers, so they ask that you only add up the even ones.

The function `add_even_numbers` is defined as follows:

- Inputs:
  - `a0`: the address of the start of an array of 32-bit integers
  - `a1`: the number of integers in the array
- Output:
  - `a0`: the sum of all even numbers in the array

Example: Suppose input `a0` points to `[4, 5, 6, 7]`, and input `a1` holds `4`. Then output `a0` holds `10` $(4 + 6)$.

Q4.1 (15 points) Fill in the blanks in the RISC-V code below. You may not need all the blanks. Each line should contain exactly one instruction or pseudo-instruction.

```
1  add_even_numbers:
2      addi t0, x0, 0   # set t0 to be the running sum
3  loop:
4      _____________________________________________________________________
5      lw t1 0(a0)      # set t1 to be the number in the array
6      _____________________________________________________________________
7      _____________________________________________________________________
8      _____________________________________________________________________
9      _____________________________________________________________________
10     _____________________________________________________________________
11 skip:
12     _____________________________________________________________________
13     _____________________________________________________________________
14     j loop
15  end:
16     _____________________________________________________________________
17     _____________________________________________________________________
```

Q4.2 (5 points) Translate the `j loop` instruction under the `skip` label to hexadecimal. Assume that every line in the above code is filled with exactly one instruction (or pseudo-instruction that expands to one instruction).

`0x`

Optionally, for partial credit, write the offset in bytes as a decimal number in the box below.

bytes

Q4.3 (5 points) Suddenly, your professor starts hating prime numbers, so now they only want you to sum up the non-prime numbers.

Assume you are given a function `is_prime` that follows calling convention. What combination of modifications to the `add_even_numbers` function is needed in order to sum up all the non-prime numbers in the array? Select all that apply.

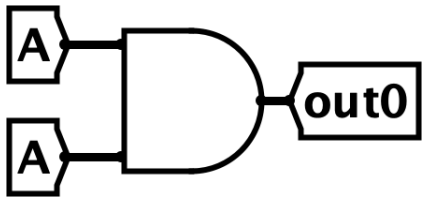
- ☐ (A) Use another register to track the number of times `is_prime` is called
- ☐ (B) Replace the code used to check if the number is even with a call to `is_prime`
- ☐ (C) Decrement the stack pointer by some amount at the start of the function, and increment the stack pointer by the same amount at the end of the function
- ☐ (D) Save some values in `a` registers instead of `t` registers
- ☐ (E) Save some values in `s` registers instead of `t` registers
- ☐ (F) Save used `a` registers onto the stack at the beginning of the function
- ☐ (G) Save used `s` registers onto the stack at the beginning of the function
- ☐ (H) Save used `t` registers onto the stack at the beginning of the function
- ☐ (I) Save another register (besides the `a`, `s`, or `t` registers) onto the stack at the beginning of the function
- ☐ (J) Restore at least one register from the stack at the end of the function
- ☐ (K) None of the above

Q5 *TF?* **(10 points)**

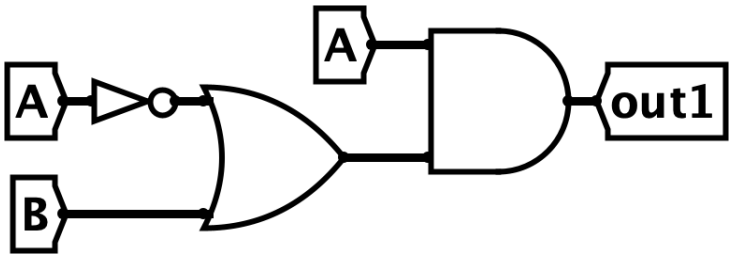
Simplify the Boolean logic for the following circuits. Your answer may only use the following characters:

Character	Description
`A, B, C, D`	Inputs
`*`	AND
`+`	OR
`^`	XOR
`~`	NOT
`( )`	Parentheses
`0, 1`	Constants

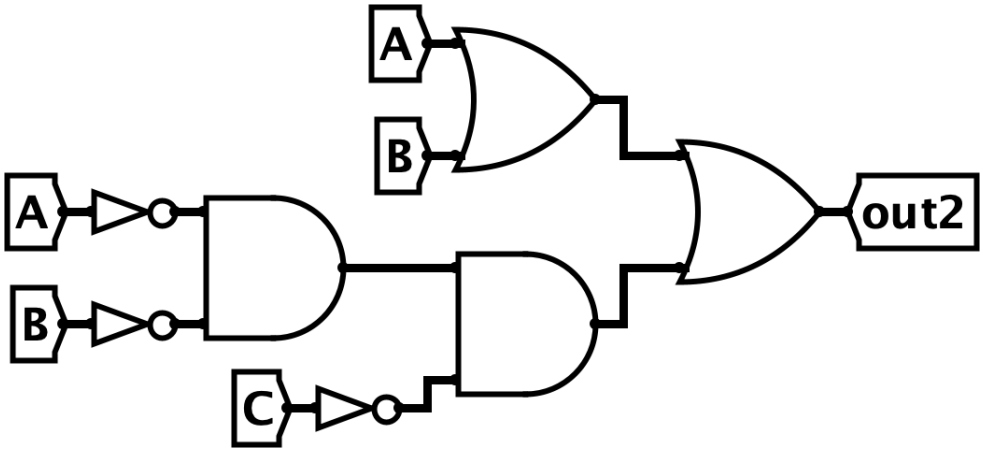
Example: the following circuit, which has Boolean logic `A * A`, can be simplified to `out0 = A`.



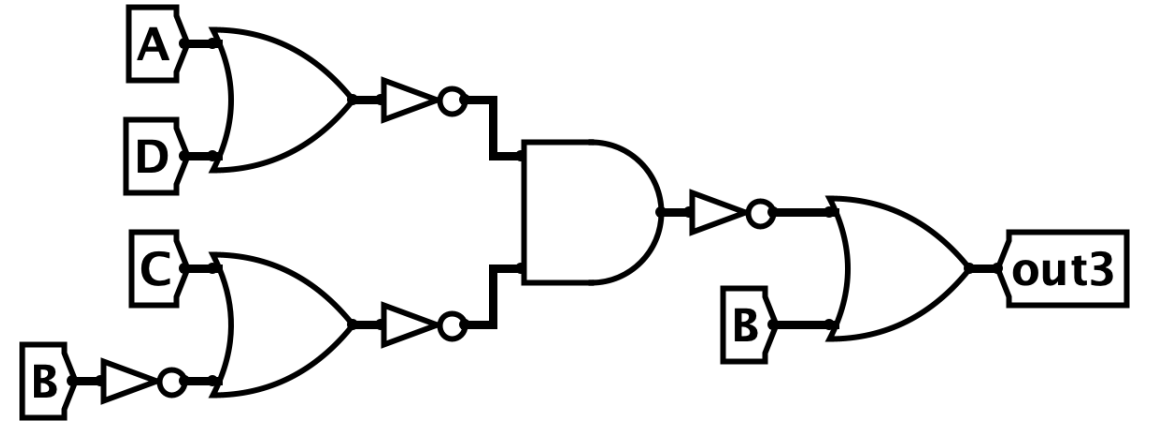
Q5.1 (2 points)



Q5.2 (4 points)



Q5.3 (4 points)



Nothing on this page will be graded.

Is there anything you want us to know?