Student Name:

Student ID:

Mock Final Exam: CSE216: Programming Abstractions, SUNY Korea Spring 2025

- This is an **ungraded** mock exam.
- Time limit: **120 minutes**. You can leave earlier if you show to TA that you have done all what you can do. Please do not simply quit and give up.
- Each problem is independent. Work at your own pace.
- Solution to the mock exam will not be distributed, but you can ask AI, Friends, TA, or Professor.
- Note: The actual final exam will be 150 minutes and include more problems.

Problem 1 — (20 points)

(1)

Write both a **context-free grammar** and a **regular expression** for a language **L** over the alphabet `{a, b}` such that **L consists of all strings that start with the character 'a'**.

Examples of accepted strings: `"a"`, `"ab"`, `"aba"`, `"abab"`
Examples of rejected strings: `""`, `"b"`, `"ba"`, `"bba"`, `"bb"`

(2)

Write both a **context-free grammar** and a **regular expression** that accept all strings over `{0, 1}` such that the number of `0`s is **divisible by 3**.
(*A string with zero `0`s is also accepted.*)

Examples of accepted strings: `"000"`, `"111"`, `"000111"`
Examples of rejected strings: `"00"`, `"10"`, `"1001"`, `"11001"`

Problem 2 — (10 points)

Suppose we define the following in OCaml:

```
let add x y = x + y
```

For each expression below, indicate whether it:

- produces an **integer**
- returns a **function**
- results in a **type error**
- (A) `add 5 1`
- (B) `add 5`
- (C) `(add 5) 1`
- (D) `add (5 1)`

Problem 3 — (30 points)

(1)

Define a new OCaml type called `shape`. This type should include:

- A `Circle` with a `float` radius
- A `Rectangle` with two `float` side lengths
- A `Triangle` with three `float` side lengths

(2)

Implement the function:

```
calculate_area : shape -> float
```

This function returns the area of the given shape. You may assume a predefined constant `pi`. Use **Heron's formula** for the triangle:

If sides are `a`, `b`, `c`, let `s = (a + b + c) / 2` Then: `area = sqrt(s *. (s -. a) *. (s -. b) *. (s -. c))` Use `sqrt` for square root.

(3)

Write the OCaml function:

```
contains : 'a -> 'a list -> bool
```

This function returns `true` if the given element exists in the list. **Do not use any built-in list functions.**
Example: `contains 4 [3; 4; 5] = true`

(4)

Write the function:

```
partition : int list -> (int list) list
```

This function splits a list into **runs of consecutive identical numbers**. Example: `partition [9; 9; 5; 6; 6; 6; 3]` → `[[9; 9]; [5]; [6; 6; 6]; [3]]`

(5)

Define:

```
genlist : int -> int -> int list
```

This function returns a list `[m; m+1; ...; n]`. If `m > n`, return the empty list `[]`.

Problem 4 — (40 points)

Assume the following OCaml type:

```
type btree = Leaf of int | Node of int * btree * btree
```

Implement the following functions:

(i) `preorder : btree -> int list`

Returns a list of node values in **preorder traversal**.

Example:

```
let t = Node(1, Node(2, Leaf 4, Leaf 5), Node(3, Leaf 6, Leaf 7))
preorder t
(* returns [1; 2; 4; 5; 3; 6; 7] *)
```

(ii) `followpath : btree -> bool list -> int list`

Follows a path in the tree guided by a list of booleans.

- `true` means go **left**
- `false` means go **right** Assume the path is always valid.

Example:

```
followpath t [true; false]
(* returns [1; 2; 5] *)
```

(iii) `height : btree -> int`

Returns the **height of the tree**, defined as the number of edges in the longest path from root to leaf.

Example:

```
height t  (* returns 2 *)
```

(iv) `balanced : btree -> bool`

Returns `true` if for **every internal node**, the heights of the left and right subtrees differ by **no more than 1**.

Examples:

```
let t1 = Node(1, Leaf 2, Node(3, Leaf 6, Leaf 7))  (* returns true *)
let t2 = Node(1, Leaf 2, Node(3, Leaf 6, Node(8, Leaf 7, Leaf 9)))  (*
returns false *)
```

Problem 5 — (10 points)

Reverse Only Letters in a String (In-Place) — C Programming

Write a C function that **reverses only the alphabetic characters** of a null-terminated string `s`. Non-letter characters (like punctuation, spaces, digits) must **remain in their original positions**.

Function prototype:

```
void reverseLetters(char* s);
```

Example Input:

```
char s[] = "a,b$c";
reverseLetters(s);
```

Output: `s` becomes `"c,b$a"`

Constraints:

- Use `strlen(s)` to get string length.
- Do **not** use other C library functions.
- Consider both uppercase and lowercase letters as alphabetic.
- Modify the string **in-place**.
- You may define helper functions.

Hint: Use two pointers — one from the start, one from the end — and skip over non-letter characters.