CSE216 Programming Abstraction

State University of New York, Korea

The mock exam is designed to help you review the topics.

It may differ significantly from the real exam.

While we will discuss the solutions to the mock exam, they will not be provided in a copy-paste format.

(1) Write a regular expression pattern to match valid music notes according to the criteria below: A music note is represented by a capital letter A to G (inclusive) followed by an optional symbol: sharp (#), flat (b), or natural (n).

```
Example valid inputs: C, D#, Fb, Gn

Example invalid inputs: H, C##, Fm, C#b
```

Note. The sharp symbol ("#") is not a special character in regular expressions. So you do *not* need to escape it with a backslash.

(2)** What is the language generated by the following grammar? Select one answer from the four choices.

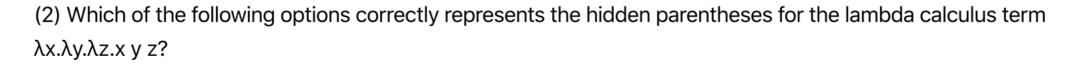
S -> aSbb |
$$\epsilon$$

- A. The set of all strings that start with 'a' and end with two 'b'.
- B. The set of all strings that contain twice as many 'b's as 'a's.
- C. The set of all strings that contain an odd number of 'a's followed by an even number of 'b's.
- D. The set of all strings that contain n 'a's followed by m 'b's, where m = 2n >= 0

Problem 2. Lambda Calculus (points = 10)

(1) Consider the following lambda calculus term: $\lambda x.x$ y z. Which of the following options correctly represents the hidden parentheses?

- A. λx.(x y) z
- B. (λx.x y) z
- C. λx.x (y z)
- D. (λx.x) y z



A. $(\lambda x.\lambda y.\lambda z.x)$ y z

B. $\lambda x.(\lambda y.\lambda z.x)$ y z

C. $\lambda x. \lambda y. (\lambda z. x y z)$

D. $\lambda x.(\lambda y.(\lambda z.x (y z)))$

What is the beta-normal form of the following lambda expressions:

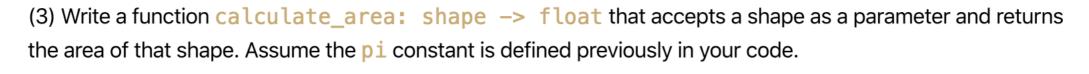
(3) (λx.λy.xy)(λz.λw.zw)

(4) $(\lambda y.\lambda x.xyy)(\lambda z.\lambda w.zww)$

Problem 3. Ocaml (points = 60)

- (1) Suppose that we have defined let add x y = x + y.
 - · Which of the following produces an integer,
 - which produces a function, and
 - which produces an error?
 - o (A) add 5 1
 - o (B) add 5
 - o (C) (add 5) 1
 - o (D) add (5 1)

(2) Define a new type **shape**. This type should represent either a **Circle**, a **Rectangle**, or a **Triangle**. Here, Circle, Rectangle and Triangle are constructors. Each shape should have parameters associated with it: a **float** radius for the Circle, two **float** sides for the Rectangle and three **float** sides for Triangle.



You can use *Heron's formula* to calculate the area of a triangle: If a, b, and c are the lengths of the sides of the triangle, and s is calculated as (a+b+c)/2, then the area of the triangle can be calculated as: Area = sqrt(s(s-a)(s-b)(s-c)). In Ocaml, you can use sqrt to get the square root of a float.

(4) Define a type tree	that is either Empty	or a Node with ar	n integer and two ch	ildren, which are als	so tree s.

(5) Write a function depth: tree -> int that calculates the depth of a tree.

(6) Update the tree type so it works with any type of data, not just integers.

type 'a new_tree = (*todo*)

(7) Write an Ocaml function product: int list -> int that returns the product of all the elements in a list of integers. The product of all the elements of an empty list is 1.

(8) Write a function power: int -> int -> int, which takes two integers a and b and returns a^b.

(9) A *predicate* is a function that returns a boolean value. In OCaml, the type of such a function can be expressed as 'a -> bool. Write a function remove_if: 'a list -> ('a -> bool) -> 'a list, which takes a list and a predicate, and removes all the elements that satisfy the predicate. Below is an example.

```
# remove_if [1;2;3;4;5] (fun x -> x mod 2 = 1);;
- : int list = [2; 4]
```

(10) Find the last two elements of a list. last_two: 'a list -> ('a * 'a) Option. Below are two examples.

```
# last_two ["a"; "b"; "c"; "d"];;
- : (string * string) option = Some ("c", "d")
# last_two ["a"];;
- : (string * string) option = None
```

Problem 4. Ocaml Application (points = 15)

Given the definition of a binary tree:

```
type 'a btree =
  | Empty
  | Node of 'a * 'a btree * 'a btree
```

(1) Implement a function height: 'a btree -> int that takes a binary tree and returns its height. The height of a binary tree is defined as the length of the longest path from the root to a leaf. An empty tree has height 0, and a tree with one node has height 1. For example, given the following tree:

```
let mytree = Node (1, Node (2, Empty, Empty), Node (3, Node (4, Empty,
Empty), Empty))
```

The function height mytree should return 3. The tree looks like:

```
1
/\
2 3
/
4
```

(2) ** Implement a function is_balanced : 'a btree -> bool that checks whether a binary tree is balanced. A binary tree is balanced if it is Empty, or the height of the two subtrees of every node differ less than or equal to one. For example, given the following tree:

```
let mytree2 = Node (1, Node (2, Node (3, Empty, Empty), Empty)
```

The function is_balanced mytree should return true, while is_balanced mytree2 should return false. The tree looks like:

```
1
/
2
/
3
```

You can use the built-in function abs in this question.

Problem 5. C Basics (points = 5)

Implement a function that takes a string as input and reverse the string in-place. The function prototype is given as: void* reverseString(char* s).

Example:

- input s: "Hello, World!"
- After reverseString(s) is executed, *s points to "!dlroW ,olleH"

Details:

- Your input s is a pointer to a string. The string consists only of printable ASCII characters and is null-terminated.
- You can use strlen(s) to get the length of a string in this problem. But you cannot use other library functions. You can get half of the length by strlen(s)/2. In C, when you perform the division of two integers, the result is also an integer that is truncated.
- Your function should reverse of the input string in-place, namely, modifies the data directly in the memory where it is already stored.

Fill in the code below.

```
void reverseString(char* str) {
   //todo
```