CSE216
Programming Abstraction

State University of New York, Korea

Final schedule

https://www.sunykorea.ac.kr/_prog/pdf_viewer/view_publication.php?id=22b721d403543488c075458b1c630659

The mock exam is the last year's final.

It may differ significantly from the year's final.

We do not provide the solutions in a copy-paste format.

Final will cover all lectures except tail-recursion

Fold_left, fold_right were not covered last year but will be covered for this year.

Part 1. Regular Expressions (Points = 9)

(1) Write a regular expression to match valid **octal numbers** in C. In C, an **octal number** is a number that starts with a `0` (zero), followed by *zero or more* octal digits (0-7). The number `0` by itself is also considered a valid octal number.

Example valid inputs:

- `0123`
- `0777`
- `05`
- `0`

Example invalid inputs:

- `123` (does not start with `0`)
- `089` (contains a digit outside the range 0-7)
- `0x77` (prefixed for hexadecimal, not octal)
- `00A7` (contains invalid character `A`)

Your task is to construct the regular expression pattern to match valid octal numbers in C accurately.

(2) In regular expressions, special characters like

```
., *, +, ?, ^, $, (, ), [, ], {, }, |, \
```

have special meanings. To match these special characters literally in a string, we need to escape them using a backslash `\`. For example, the regular expression `\$100` matches `$100`.

Write a regular expression to match IP addresses that follow this format: Four sequences of one to three digits, separated by a dot '.' (e.g., 192.168.0.1).

(3) Write a regular expression to match URLs that start with "ftp://" or "sftp://", followed by "www.", then a sequence of alphanumeric characters, followed by '.', and ends with a sequence of alphabets. Here, alphanumeric characters mean the alphabet (uppercase or lowercase) or the numbers from 0 to 9. Note that the dot `.` needs to be escaped via '`\.`' but `:` or `/` do not need to be escaped.

```
Example valid inputs: ftp://www.cse216.kr

Example invalid inputs: ftp://www.hello.2024
```

Part 2. Context-Free Grammar (Points = 9. No partial points.)

Consider the Following Grammars:

Grammar 1:

```
S → aSb | ε
```

Grammar 2:

```
S → aSa | bSb | ε
```

Grammar 3:

```
S → aS | Sb | ε
```

Questions:

(4). The following language L = {$a^n b^n$ | n ≥ 0} corresponds to:

- (A) Grammar 1
- (B) Grammar 2
- (C) Grammar 3
- (D) None of them.

(5). The following language L = {$a^n b^m$ | n ≥ 0, m ≥ 0} corresponds to:

- (A) Grammar 1
- (B) Grammar 2
- (C) Grammar 3
- (D) None of them.

(6). A palindrome is a string that reads the same forward and backward. For example, "abba," "aa," and "b" are palindromes, but "ab" and "ba" are not. The set of all strings that are palindromes over the alphabet {a, b} corresponds to:

- (A) Grammar 1
- (B) Grammar 2
- (C) Grammar 3
- (D) None of them.

Part 3. Lambda Calculus (Total Points = 18. No partial points.)

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

(7) (λa.a)(λc.λd.dc)

(8) (λa.b)(λc.λd.abc)

(9) (λa.aa)(λc.λd.cd)

(10) (λx.λy.y) (λx.x a) x

Please select a single answer for the following multi-choice questions.

(11) Which of the following is alpha-equivalent to $\lambda x.\ \lambda y.\ yx$?

- A) $\lambda y.\ \lambda x.\ yx$
- B) $\lambda x.\ \lambda y.\ xy$
- C) $\lambda y.\ \lambda x.\ xy$
- D) $\lambda x.\ \lambda y.\ xx$

(12) Given the standard lambda encoding for the boolean values TRUE and FALSE as $\lambda x.\ \lambda y.\ x$ and $\lambda x.\ \lambda y.\ y$ respectively, and considering the OR operation encoded as $\lambda p.\ \lambda q.\ p\ \mathrm{TRUE}\ q$, what is the result of reducing OR TRUE FALSE?

- A) TRUE
- B) FALSE
- C) It results in an error
- D) None of the above

Part 4. Ocaml Types (Points = 24. No partial points.)

Provide the type of the following OCaml expressions.

(13) [(3.8, 5.8) ; (0.1, 4.5)]

(14) [Some 3; None; Some 4]

(15) (true, "foo", 3.0)

(16) let f x y = x::y in f "hello" ["world"]

(17) let div x y = x/y in div 3

Provide the type of the defined Ocaml functions:

(18) let prod x y = x *. y

(19) the Fibonacci function

```
let rec fibonacci n =
  if n <= 1 then n
  else fibonacci (n - 1) + fibonacci (n - 2)
```

(20) The higher-order Ocaml function `twice` defined as:

```
let twice f x = f (f x)
```

Reminder: In Ocaml type system, the type `(a -> b) -> c` is different from `a -> b -> c`. The latter is interpreted as `a -> (b -> c)`.

Part 5. Ocaml Programming (Points = 35)

---

(21) Define a new type `geometry` in OCaml. This type should represent three 2D geometric figures: a `Circle`, a `Square`, and a `Pentagon`. The `Circle` constructor should have a `float` parameter for its radius. The `Square` constructor should have a `float` parameter for the length of its side. The `Pentagon` constructor should have a `float` parameter for the length of its sides.

(22) Write a function `calculate_perimeter: geometry -> float` in OCaml that calculates the perimeter of a given geometric figure. Use the following assumptions:

- For a sphere with radius r, the perimeter is $2\pi r$ mathematically.
- For a square with side a, the perimeter is 4a mathematically.
- For a pentagon with sides a, the perimeter is 5a mathematically.

You need to define the constant pi as 3.14159 somewhere in your code.

(23) Write an OCaml function `take: int -> 'a list -> 'a list` that takes the first `N` elements of a list.

- If `N` is greater than the length of the list, the function should return the entire list.
- If `N` is less than or equal to `0`, the function should return an empty list.

Examples:

```
take 3 [1; 2; 3; 4; 5];;     (* Output: [1; 2; 3] *)

take 0 [1; 2; 3; 4; 5];;     (* Output: [] *)

take 7 [1; 2; 3];;           (* Output: [1; 2; 3] *)
```

(24) Write an Ocaml function `product: int list -> int` that returns the product of all the elements in a list of integers. We assume the product of an empty list's elements is 1.

(25) Write an OCaml function `last: 'a list -> 'a option` that returns the last element of a list.

- If the list is empty, the function should return `None`.

Examples:

```
# last ["a"; "b"; "c"; "d"];;
- : string option = Some "d"
# last [];;
- : 'a option = None
```

(26) A Binary Search Tree (BST) is a tree data structure in which each node has at most two children, referred to as the left and right child. *For each node, all elements in the left subtree are less than the node, and all elements in the right subtree are greater. Besides, the same integer value does not appear more than once in the tree.*

In OCaml, we can represent a BST using the following type.

```
type bst =
  | Empty
  | Node of int * bst * bst
```

For example, a BST with the numbers 1, 2, and 3 could be represented as:

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

Namely,

```
  2
 / \
1   3
```

Write a function `sum_tree: bst -> int` that sums all the integers in a BST.

(27) Implement a function `insert : bst -> int -> bst` that takes a BST and an integer and returns a new BST with the integer inserted at the correct position. If the inserted integer is already in the tree, the insertion should have no effects.

For example, if you call `insert mytree 4`, the function should return:

```
Node (2, Node (1, Empty, Empty), Node (3, Empty, Node (4, Empty, Empty)))
```

Namely,

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

If you call `insert mytree 2` or `insert mytree 3`, the function should return something equal to `mytree`.

To solve this question, you can assume that the input tree is a valid BST.

Part 7. C Programming (Points = 5)

(28) Write a C function named `reverseArray` that reverses the elements of an integer array. Here is the prototype of the function:

```c
void reverseArray(int arr[], int size);
```

Your task is to:

- Define the `reverseArray` function according to the given prototype. The `arr[]` parameter is the array to be reversed, and `size` is the number of elements in the array.
- You need to reverse the array in place, meaning you do not use an additional array for the reversal process.

A `main` function has been provided for you to test your `reverseArray` function.

```c
#include <stdio.h>

void reverseArray(int arr[], int size) {
    // Your implementation goes here









}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    reverseArray(arr, 5);

    for (int i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```