Теорія мов програмування

Лабораторна робота 5. Імплементація мов функціонального програмування

5.1 Чи є комутативним додавання в C ++, Pascal, Java чи Python? Чи записуватиме `(a + b)` завжди те саме значення, що і `write (b + a)`? Розглянемо програму Паскаля на рис. 10.1 (лекція 10). Що дає ця програма? Що б ми отримали, якби вираз було записано `(b + a ())`?

5.2 Інша стратегія скорочення називається додатковим зменшенням замовлення. Використовуючи цю стратегію, найперший зліва внутрішній редекс завжди зменшується першим. Використовуйте цю стратегію для зменшення виразу на рис. 10.2. Обов'язково спершу вкажіть вираз у дужках, щоб бути впевненим, що ви пов'язали редекси.

5.3 Зменшіть вираз `(λx.y)((λx.x x)(λx.x x))` як за звичайним, так і за аплікативним зменшенням порядку.

5.4 Напишіть вирази, які обчислюють значення, описані нижче. За потреби зверніться до базової бібліотеки https://smlfamily.github.io/Basis/ .
1. Розділіть ціле число, зв'язане з `x`, на `6`.
2. Помножте ціле число `x` і дійсне число `y`, отримавши в результаті найближче ціле число.
3. Розділіть дійсне число `6.3` на дійсне число, прив'язане до `x`.
4. Обчисліть залишок від ділення цілого числа `x` на ціле число `y`.

5.5 `n!` називається факторіалом `n`. Він визначається рекурсивно як `0! = 1` і `n! = n * (n - 1)!`. Запишіть це як рекурсивну функцію в SML.

5.6 Послідовність Фібоначчі - це послідовність чисел `0, 1, 1, 2, 3, 5, 8, 13,` ... Наступні числа в послідовності отримуються шляхом додавання двох попередніх чисел у послідовності разом. Це призводить до рекурсивного визначення послідовності Фібоначчі. Що таке рекурсивне визначення послідовності Фібоначчі?
ПІДКАЗКА: Перше число в послідовності можна розглядати як нульовий елемент, потім перший елемент є наступним тощо. Отже, `fib (0) = 0`. Після досягнення визначення напишіть рекурсивну функцію SML, щоб знайти `n`-й елемент послідовності.

5.7 Нижче наведено НЕдопустимі конструкції списків у SML. Чому ні? Чи можете ви їх виправити?
- `#"a" ::["beautiful day"]`
- `"hi" ::"there"`

- `["how","are"] ::"you"`
- `[1,2.0,3.5,4.2]`
- `2@[3,4]`
- `[] ::3`

5.8 Напишіть функцію з назвою **`explode`**, яка прийме рядок як аргумент і поверне список символів у рядку. Отже, **`explode("hi")`** видасть у результаті **`[# "h", # "i"]`**.
ПІДКАЗКА: Як отримати перший символ рядка?

5.9 Використовуйте функцію додавання, щоб написати **`reverse`**. Функція **`reverse`** перевертає елементи списку. Її сігнатура наступна
`reverse = fn : 'a list -> 'a list`

5.10 Перепишіть **`reverse`**, використовуючи зіставлення шаблонів.

5.11 Яке значення **`x`** у різних пронумерованих точках у наступному виразі? Будьте обережні, це не те, що ви думаєте, якщо ви покладаєтесь на своє *імперативне* розуміння коду.

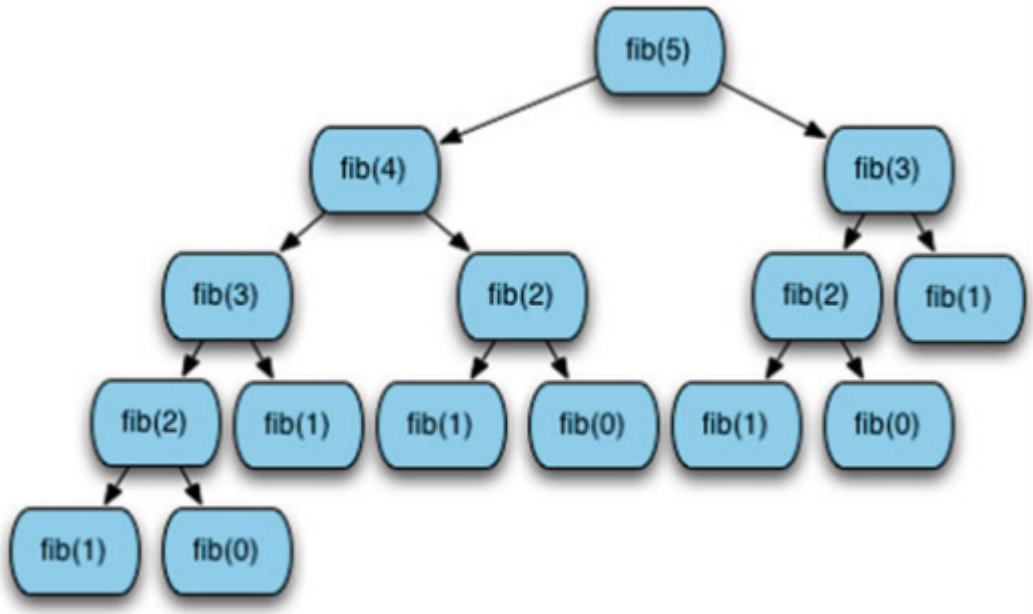
```
let val x = 10 in
    (* 1. Value of x here? *)
    let val x = x+1
    in
        (* 2. Value of x here? *)
        x
    end;
    (* 3. Value of x here? *)
    x
end
```

5.12 Визначте тип даних для цілочисельних списків. Список складається з голови та хвоста. Іноді цей конструктор називають **`cons`**. Порожній список також є списком і зазвичай називається нульовим. Однак у цій практичній задачі, щоб відрізнити від вбудованого **`nil`**, ви могли б назвати це **`nil'`**.

5.13 Напишіть функцію **`maxIntList`**, яка повертає максимальне ціле число, знайдене в одному зі списків, які ви щойно визначили в завданні 5.12. Ви можете проконсультуватися у https://smlfamily.github.io/Basis/ , щоб отримати допомогу щодо пошуку максимуму двох цілих чисел.

5.14 Один із способів довести, що функція **`fib`** на рис. експоненційна, - показати, що кількість викликів **`fib(n)`** обмежена двома експоненційними функціями. Іншими словами, існує експоненційна функція **`n`**, яка завжди

повертає менше, ніж кількість викликів, необхідних для обчислення **`fib(n)`**, і існує інша експоненціальна функція, яка завжди повертає більше, ніж кількість необхідних викликів для обчислення **`fib(n)`** для певного вибору стартового **`n`** та всіх значень, більших за нього. Якщо кількість викликів для обчислення **`fib(n)`** лежить між ними, тоді функція **`fib`** повинна мати експоненційну складність. Знайдіть дві експоненційні функції виду c^m, які обмежують кількість викликів, необхідних для обчислення **`fib(n)`**.



5.15 Розглянемо функцію **`reverse`** із завдання 5.10. Функція додавання викликається **`n`** разів, де **`n`** - довжина списку. Скільки операцій **`cons`** відбувається кожного разу, коли викликається додаток? Яка загальна складність функції реверсу?

5.16 Покажіть стек під час виконання в точці, коли факторіал **`0`** виконується, коли початковий виклик був факторіалом **`6`**.

5.17 Використовуйте шаблон акумулятора, щоб розробити більш ефективну функцію **`reverse`**. Не використовуйте функцію **`append`** в ефективній функції **`reverse`**.
ПІДКАЗКА: Що ми намагаємось накопичити? У чому полягає ідентичність цієї операції?

5.18 Напишіть функцію, яка з урахуванням непідготовленої функції з двох аргументів поверне викривлену форму функції, щоб вона брала свої аргументи по черзі.
Напишіть функцію, яка, надавши функцію, що працює, яка приймає два аргументи по черзі, поверне необроблену версію заданої функції.

Виконані завдання надішліть на пошту викладачеві.