 **GIK Institute of Engineering Sciences and Technology, Topi**   
 **Spring 2023 (FCSE) Midterm Exam**   
 **24th March 2023, 8:00 am – 9:00 am**

|  |  |  |
| --- | --- | --- |
| **Course Code: AI302** | **Course Name: Structure and Interpretation of Computer Programs** | |
| **Instructor name: Taj Muhammad Khan** | | **Section: AI** |
| **Vetter name:** | | **Vetter signature:** |
| **Student Name:** | | **Registration No:** |

* Read each question completely before answering it. There are **7 questions.**
* In case of any ambiguity, you may make an assumption. Write it but your assumption should not contradict any statement in the question paper.
* Write the answer in the space below each question. **Answers written with pencil will get zero marks**.
* Do rough work with pencil and then write final answer with a pen. Use last few pages for rough work.
* Read the whole paper first.

.

**Time**: 70 minutes. **Max Marks**: 50

|  |  |  |
| --- | --- | --- |
| Question | Total Marks | Obtained |
| Question # 1 | 5 |  |
| Question # 2 | 5 |  |
| Question # 3 | 5 |  |
| Question # 4 | 5 |  |
| Question # 5 | 10 |  |
| Question # 6 | 10 |  |
| Question # 7 | 10 |  |
| Total | 50 |  |

Question 1: Translate the following C++ codes into Scheme.

a). (x+y) – 2;

b). func1(x-2, func2(y-3+4), func3(z));

3) int collatz (int x) {

if (x%2==0)

return x/2;

else

return 3\*x+1;

}

Question 2: Given below are two versions of my-length procedure which returns the length of the given list. Comment on their order of growth in terms of both *time* and *space*. Which one is recursive and which iterative?

|  |  |
| --- | --- |
| Version 1 | Version 2 |
| |  | | --- | | ([define](https://cs.brown.edu/courses/cs173/2008/Manual/reference/define.html#(form._((lib._scheme/base..ss)._define))) (my-length lst) | | ([cond](https://cs.brown.edu/courses/cs173/2008/Manual/reference/if.html#(form._((lib._scheme/private/letstx-scheme..ss)._cond))) | | (([null?](https://cs.brown.edu/courses/cs173/2008/Manual/reference/pairs.html#(def._((lib._scheme/list..ss)._empty~3f))) lst) 0] | | ([else](https://cs.brown.edu/courses/cs173/2008/Manual/reference/if.html#(form._((lib._scheme/private/letstx-scheme..ss)._else))) ([+](https://cs.brown.edu/courses/cs173/2008/Manual/reference/numbers.html#(def._((quote._~23~25kernel)._+))) 1 (mylength ([cdr](https://cs.brown.edu/courses/cs173/2008/Manual/reference/pairs.html#(def._((lib._scheme/list..ss)._rest))) lst)))))) | | |  | | --- | | ([define](https://cs.brown.edu/courses/cs173/2008/Manual/reference/define.html#(form._((lib._scheme/base..ss)._define))) (my-length lst) | | ([define](https://cs.brown.edu/courses/cs173/2008/Manual/reference/define.html#(form._((lib._scheme/base..ss)._define))) (helper lst len) | | ([cond](https://cs.brown.edu/courses/cs173/2008/Manual/reference/if.html#(form._((lib._scheme/private/letstx-scheme..ss)._cond))) | | (([null?](https://cs.brown.edu/courses/cs173/2008/Manual/reference/pairs.html#(def._((lib._scheme/list..ss)._empty~3f))) lst) len) | | ([else](https://cs.brown.edu/courses/cs173/2008/Manual/reference/if.html#(form._((lib._scheme/private/letstx-scheme..ss)._else))) (helper ([cdr](https://cs.brown.edu/courses/cs173/2008/Manual/reference/pairs.html#(def._((lib._scheme/list..ss)._rest))) lst) ([+](https://cs.brown.edu/courses/cs173/2008/Manual/reference/numbers.html#(def._((quote._~23~25kernel)._+))) len 1))))) | | (helper lst 0)) | |

Question 3: The *ith* repetition of a function *f, f i*, is defined by the following, for i >= 0 and for all arguments x:

*f i(x) = x* , if i = 0

*= f ( f i-1(x) )* , otherwise.

Write a Scheme procedure (repeat f i x) which compute *f i(x)* while generating a recursive process. The output should be as in the examples below:

(repeat (lambda (x) (\* x 2)) 0 1) ==> 1

(repeat (lambda (x) (\* x 2)) 3 1) ==> 8

(repeat (lambda (x) (\* x 2)) 4 1) ==> 16

(repeat (lambda (x) (+ x 10)) 5 9) ==> 59

Question 4: In Scheme, write a procedure, minimum, that takes a non-empty list of numbers, items, and returns an item that is no greater than any other item in items. The following are examples:

(minimum (list 3 4 2 4 2)) ==> 2

(minimum (list 2)) ==> 2

(minimum (list 7 5 99 22 5 -1 3)) ==> -1

In your solution, you may not use the built-in Scheme procedure min.

.

Question 5: In Scheme, write a procedure, merge, that takes two sorted lists of numbers, first and second, and returns a sorted list containing the numbers from both lists. The following are examples.

(merge (list 1 2 8 11 21) (list 3 4 7 12 13 20)) ==> (1 2 3 4 7 8 11 12 13 20 21)

(merge (list 1 1 2 8 11 21 100) (list 3 3 4 7 11 12 13 20)) ==> (1 1 2 3 3 4 7 8 11 11 12 13 20 21 100)

(merge (list 1) (list 1 2)) ==> (1 1 2)

(merge nil (list 1 2)) ==> (1 2)

(merge (list 1 2) nil) ==> (1 2)

(merge nil nil) ==> ()

Question 6: The procedure accumulate-n is similar to accumulate except that it takes as its third argument a sequence of sequences, which are all assumed to have the same number of elements. It applies the designated accumulation procedure to combine all the first elements of the sequences, all the second elements of the sequences, and so on, and returns a sequence of the results. For instance, if s is a sequence containing four sequences, ((1 2 3) (4 5 6) (7 8 9) (10 11 12)), then the value of (accumulate-n + 0 s) should be the sequence (22 26 30). Rewrite the following definition of accumulate-n to fill in the missing expressions ⟨??⟩:

(define (accumulate-n op init seqs)

(if (null? (car seqs))

nil

(cons (accumulate op init **⟨??⟩** )

(accumulate-n op init **⟨??⟩** ))))

Question 7: What will be the output of the combination (func (list (list 1 2) (list 3 (list 4 5)))) given the code below? The list looks like this => ((1 2) (3 (4 5))).

(define (func x)

(define (helper x r)

(cond ((null? x) nil)

((not (pair? x)) (cons x r))

(else (append (helper (cdr x) r)

(helper (car x) r)))))

(helper x nil))

Reference:

(define (accumulate op initial sequence)

(if (null? sequence)

initial

(op (car sequence)

(accumulate op

initial

(cdr sequence)))))

(define (filter predicate sequence)

(cond ((null? sequence) nil)

((predicate (car sequence))

(cons (car sequence)

(filter predicate

(cdr sequence))))

(else (filter predicate

(cdr sequence)))))

(define (enumerate-interval low high)

(if (> low high)

nil

(cons low

(enumerate-interval

(+ low 1)

high))))

(define (even-fibs n)

(define (next k)

(if (> k n)

nil

(let ((f (fib k)))

(if (even? f)

(cons f (next (+ k 1)))

(next (+ k 1))))))

(next 0))

(define (append list1 list2)

(if (null? list1)

list2

(cons (car list1)

(append (cdr list1)

list2))))