MAKERERE UNIVERSITY

SEMESTER ONE 2024/2025 ACADEMIC YEAR

SCHOOL COMPUTING AND IMFORMATICS TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF SCIENCE IN COMPUTER SCIENCE

MCS 7105

STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS

Module 1 Group Assignment

GROUP C:

NAME	**REGISTRATION NUMBER**	**STUDENT NUMBER**
AMPEIRE EDGAR	2024/HD05/21915U	2400721915
BUGEMBE JOHN PAUL	2024/HD05/26501U	2400726501
AHARIKUNDIRA BAB LETISHA	2024/HD05/21949U	2400721949

Exercise 1.6

When evaluating procedures, we evaluate the body of the procedure with each formal parameter replaced by the corresponding argument.

In the case for; -

(define (sqrt-iter guess x)

(new-if (good-enough? guess x)

guess

(sqrt-iter (improve guess x)

x)))

It is going to keep evaluating the *strt-iter* procedure and will go into an infinite loop, hence the procedure *new-if* will not run at all.

For example, when evaluating *(sqrt-iter 1.0 9)*

Substituting

(new-if (good=enough? 1.0 9)

1.0

(sqrt-iter (improve 1.0 9) 9))

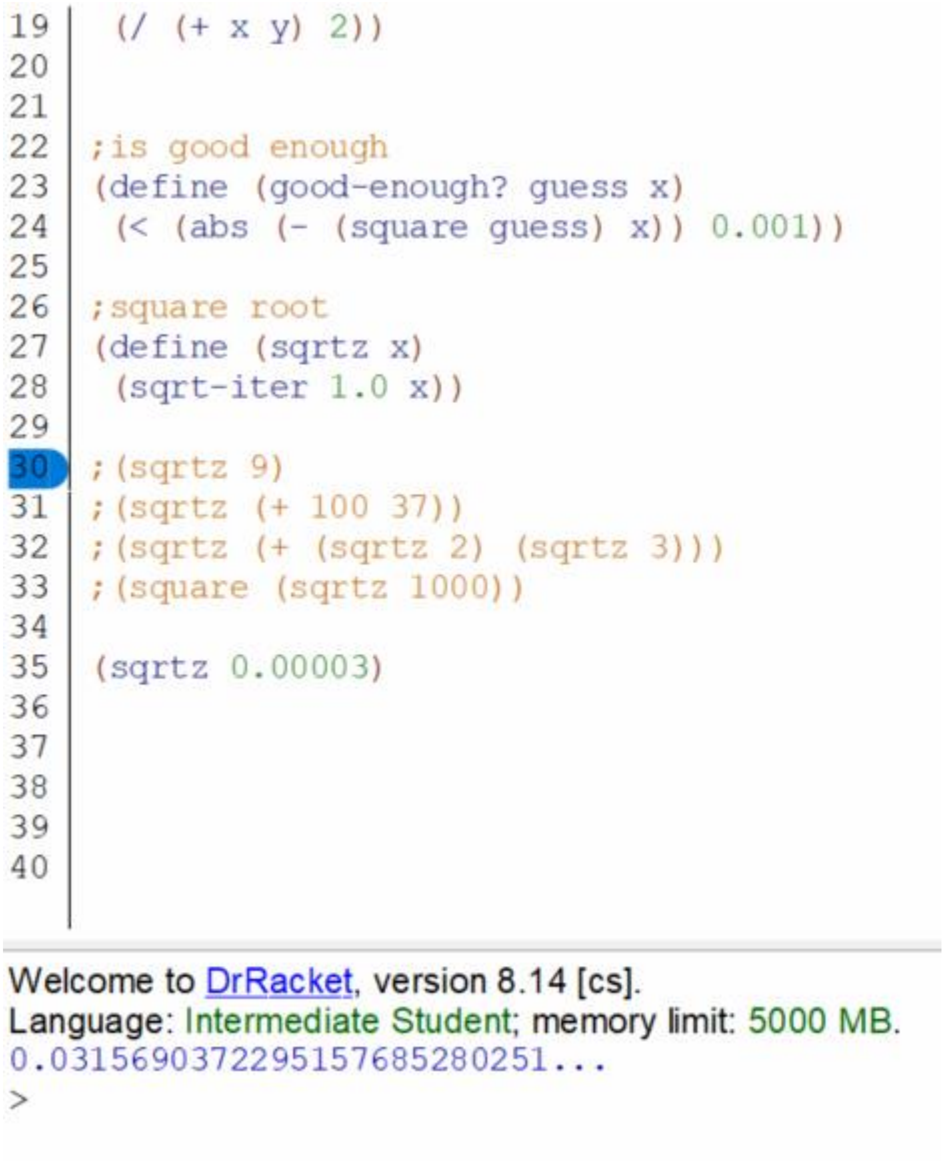
In this case we keep calling the sqrt-iter hence not calling the new -if function.

Excerise 1.7

For very small numbers, for example finding the square root of 0.00003, the absolute of the difference between the square of the guess and 0.00003 reaches 0.001 too early hence the evaluation of *sqrt-iter* stops too early hence giving us an inaccurate answer.

For example, running

(sqrt 0.003) in Dr Racket will give us



```
 (/ (+ x y) 2))


;is good enough
(define (good-enough? guess x)
 (< (abs (- (square guess) x)) 0.001))

;square root
(define (sqrtz x)
 (sqrt-iter 1.0 x))

;(sqrtz 9)
;(sqrtz (+ 100 37))
;(sqrtz (+ (sqrtz 2) (sqrtz 3)))
;(square (sqrtz 1000))

(sqrtz 0.00003)
```

```
Welcome to DrRacket, version 8.14 [cs].
Language: Intermediate Student; memory limit: 5000 MB.
0.0315690372295157685280251...
>
```

For very large numbers, like finding the square root of 1000000000

(sqrt 1000000000),

Large real numbers are encoded in computers using floating-point representation, which has a limited precision, for each iteration of procedure *sqrt-iter*, the average of guess (/ x guess) get much closer.

This leads us to run out of precision and *(- (square guess) x))* will always be greater than 0.001.

Therefore, for very large numbers, the program will run for a very long time and can even run into an infinite loop.

Improvement

The first step is to redefine *good-enough?* by considering the difference between the current guess and the improved guess, focusing on when that difference becomes small.

```
(define (good-enough? previous-guess guess)
  (< (abs (/ (- guess previous-guess) guess)) 0.00000000001))
```

The entire program now becomes

```
(define (square x) (* x x))

(define (good-enough? previous-guess guess)

  (< (abs (/ (- guess previous-guess) guess)) 0.00000000001))

(define (sqrt-iter guess x)

  (if (good-enough? guess (improve guess x))

      guess

      (sqrt-iter (improve guess x) x)))

(define (improve guess x)

  (average guess (/ x guess)))

(define (average x y)

  (/ (+ x y) 2))

(define (sqrt x)

(sqrt-iter 1.0 x))
```

Exercise 1.8

```
(define (cube x) (* x x x))


(define (good-enough? previous-guess guess)
  (< (abs (/ (- guess previous-guess) guess)) 0.00000000001))


(define (cube-root-iter guess x)
  (if (good-enough? (improve guess x) guess)
      guess
      (cube-root-iter (improve guess x) x)))


(define (improve guess x)
  (/ (+ (/ x (* guess guess)) (* 2 guess)) 3))



(define (cube-root x)
  (cube-root-iter 1.0 x))
```

Using this we can find the cube root of 27

```
(cube-root-iter 27)
```