

# CPSC-354 Report

Jackson Goldberg  
Chapman University

October 9, 2022

## Abstract

My Report.

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Homework</b>	<b>1</b>
2.1	Week 1 . . . . .	1
2.2	Week 2 . . . . .	2
2.3	Week 3 . . . . .	3
2.4	Week 4 . . . . .	3
2.5	Week 5 . . . . .	3
2.6	Week 6 . . . . .	7
<b>3</b>	<b>Project</b>	<b>7</b>
3.1	Specification . . . . .	7
3.2	Prototype . . . . .	7
3.3	Documentation . . . . .	7
3.4	Critical Appraisal . . . . .	7
<b>4</b>	<b>Conclusions</b>	<b>7</b>

## 1 Introduction

Hi there. My name is Jackson Goldberg. I am a senior here at chapman University and I am taking programming languages. I like playing chess. Enjoy reading my report.

## 2 Homework

### 2.1 Week 1

I have added my program into this github repo. It is a simple python program that loops infinitely until a and b are equal. I ran it with python 3 and manually input 9 and 33 yielding the answer 3.

## 2.2 Week 2

This week we were tasked with creating simple recursive programs in Haskell. These are my solutions for all of the assigned functions. These functions can also be found in a Haskell file titled "Main.hs".

---

```
-- Takes a list of char and returns a list of char. This works by assigning the element at odd
  positional values of a list into an empty list using the zip function which is what it
  returns.
select_evens :: [[a]] -> [[a]]
select_evens xs = [x | (x,i) <- zip xs [0..], odd i]

-- Example:
select_evens ["a","b","c","d","e"] =
  [] : (select_evens ["b","c","d","e"]) =
    ["b"] : (select_evens ["c","d","e"]) =
      ["b"] : (select_evens ["d","e"]) =
        ["b","d"] : (select_evens ["e"]) =
          ["b","d"] : (select_evens []) =
            ["b","d"]
```

---

I referenced [this](#).

---

```
-- Same logic as above just assigns based on even position.
select_odds :: [[a]] -> [[a]]
select_odds xs = [x | (x,i) <- zip xs [0..], even i]

-- Example:
select_odds ["a","b","c","d","e"] =
  ["a"] : (select_odds ["b","c","d","e"]) =
    ["a"] : (select_odds ["c","d","e"]) =
      ["a","c"] : (select_odds ["d","e"]) =
        ["a","c"] : (select_odds ["e"]) =
          ["a","c","e"] : (select_odds []) =
            ["a","c","e"]
```

---

```
-- Uses the filter to function to create a list of all matching elements to the ones provided.
  If the length is greater than 0 then it's a member.
member :: Int -> [Int] -> IO Bool
member x li = do
  let xs = filter(== x) li
  if length xs == 0
  then
    return (False)
  else do
    return (True)

-- Example:
member 2 [5,2,6] =
1 = filter(== 2) li
if length xs == 0
  else do
    return (True)
True
```

---

```
-- Uses ++ to concatenate lists.
append :: [Int] -> [Int] -> [Int]
append l1 l2 = l1 ++ l2
```

```
--Example:
append [1,2] [3,4,5] =
  [1,2] ++ [3,4,5] =
  [1,2,3,4,5]
```

---

```
-- Recursively assigns element to back of new list, creating a reverse list.
revert :: [Int] -> [Int]
revert [] = []
revert (x:xs) = revert xs ++ [x]
```

```
--Example:
revert [1,2,3] =
  [3]: revert[1,2] =
  [3,2]: revert[1] =
  [3,2,1]
```

---

```
-- Compares two strings using <=.
less_equal :: [Int] -> [Int] -> IO Bool
less_equal l1 l2 = do
  if last l1 <= last l2
  then
    return (True)
  else do
    return (False)
```

```
--Example:
less_equal [1,2,3] [2,3,2] =
  3 > 2
  False
```

---

## 2.3 Week 3

This week we were tasked with finishing the hanoi file supplied to us. The full file can be found in Hanoi.txt. Hanoi is used 31 times in the file. You can express this as formula because it doesn't matter the starting blocks since the process will always be the exact same. you will always need to reduce to the highest block so the number is nonconsequential. There are many cool visuals of this online which show what it looks like using a graphical interpretation. There is a simple break down for however many N you have.

## 2.4 Week 4

This week we worked with context free grammars and parse trees. Part one can be found [here](#), part 2 can be found [here](#).

## 2.5 Week 5

This week we are doing a lot of lambda calc stuff with abstract syntax trees.

Using the parser to generate linearized abstract syntax trees:

---

Input:

\ x . x a

[Abstract Syntax]

Prog (EApp (EAbs (Id "x") (EVar (Id "x")))) (EVar (Id "a")))

Output:

a

---

Input:

\ x . x a

[Abstract Syntax]

Prog (EAbs (Id "x") (EApp (EVar (Id "x")) (EVar (Id "a"))))

Output:

\ x . x a

---

Input:

\ x . \ y . x a b

[Abstract Syntax]

Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x")))) (EVar (Id "a"))) (EVar (Id "b")))

Output:

a

---

Input:

\ x . \ y . y a b

[Abstract Syntax]

Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "y")))) (EVar (Id "a"))) (EVar (Id "b")))

Output:

b

---

Input:

\ x . \ y . x a b c

[Abstract Syntax]

Prog (EApp (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x")))) (EVar (Id "a"))) (EVar (Id "b"))) (EVar (Id "c")))

Output:

a c

---

---

Input:  
 $\backslash x . \backslash y . y a b c$

[Abstract Syntax]

```
Prog (EApp (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "y")))) (EVar (Id "a")) (EVar (Id "b")))) (EVar (Id "c")))
```

Output:  
b c

---

---

Input:  
 $\backslash x . \backslash y . x a (b c)$

[Abstract Syntax]

```
Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x")))) (EVar (Id "a")) (EApp (EVar (Id "b")) (EVar (Id "c"))))
```

Output:  
a

---

---

Input:  
 $\backslash x . \backslash y . y a (b c)$

[Abstract Syntax]

```
Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "y")))) (EVar (Id "a")) (EApp (EVar (Id "b")) (EVar (Id "c"))))
```

Output:  
b c

---

---

Input:  
 $\backslash x . \backslash y . x (a b) c$

[Abstract Syntax]

```
Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x")))) (EApp (EVar (Id "a")) (EVar (Id "b")))) (EVar (Id "c")))
```

Output:  
a b

---

---

Input:  
 $\backslash x . \backslash y . y (a b) c$

[Abstract Syntax]

```
Prog (EApp (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "y")))) (EApp (EVar (Id "a")) (EVar (Id "b")))) (EVar (Id "c")))
```

---

Output:  
c

---

Input:  
 $\backslash x . \backslash y . x (a b c)$

[Abstract Syntax]

Prog (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "x"))))) (EApp (EApp (EVar (Id "a")) (EVar (Id "b")))) (EVar (Id "c"))))

Output:  
 $\backslash yx0 . a b c$

---

Input:  
 $\backslash x . \backslash y . y (a b c)$

[Abstract Syntax]

Prog (EApp (EAbs (Id "x") (EAbs (Id "y") (EVar (Id "y"))))) (EApp (EApp (EVar (Id "a")) (EVar (Id "b")))) (EVar (Id "c"))))

Output:  
 $\backslash yy0 . yy0$

---

Evaluate using pen-and-paper the following expressions:

---

$(\backslash x.M) N$

$N = \text{argument}$

$M = \text{function operation}$

$(\backslash x.x) a = a$

$\backslash x.x a = \backslash x.x a$  cannot be reduced further because there are no parentheses

$(\backslash x.\backslash y.x) a b = (\backslash x.(\backslash y.x) a) b$   
 $= (\backslash x.x) a$   
 $= a$

$(\backslash x.\backslash y.y) a b = (\backslash x.(\backslash y.y) a) b$   
 $= (\backslash y.y) b$   
 $= b$

$(\backslash x.\backslash y.x) a b c = ((\backslash x.(\backslash y.x) a) b) c$   
 $= ((\backslash y.a) b) c$   
 $= (\backslash y.a) b c$   
 $= a c$

$(\backslash x.\backslash y.y) a b c = ((\backslash x.(\backslash y.y) a) b) c$   
 $= ((\backslash y.y) b) c$   
 $= b c$

$(\backslash x.\backslash y.x) a (b c) = ((\backslash x.(\backslash y.x)) a) (b c)$   
 $= (\backslash y.a) (b c)$   
 $= a$

$(\backslash x.\backslash y.y) a (b c) = ((\backslash x.(\backslash y.y)) a) (b c)$   
 $= (\backslash y.y) (b c)$   
 $= b c$

$(\backslash x.\backslash y.x) (a b) c = ((\backslash x.(\backslash y.x)) (a b)) c$

$$\begin{aligned}
&= (\backslash y. a b) c \\
&= a b \\
(\backslash x. \backslash y. y) (a b) c &= ((\backslash x. (\backslash y. y)) (a b)) c \\
&= (\backslash y. y) c \\
&= c \\
(\backslash x. \backslash y. x) (a b c) &= (\backslash x. (\backslash y. x)) (a b c) \\
&= (\backslash y. a b c) \\
(\backslash x. \backslash y. y) (a b c) &= (\backslash x. (\backslash y. y)) (a b c) \\
&= (\backslash y. y)
\end{aligned}$$


---

All of the hand written notes can be found [Here](#).

## 2.6 Week 6

This week for homework we did lambda calc beta reduction on exponentiation. The file containing my solution can be found in week6.hs [Here](#) (Has .hs for sake of parentheses convenience).

## 3 Project

Introductory remarks ... Over the summer I had an internship and I primarily worked using rust. I didn't get to learn it very well but the little I did learn got me really interested in it. For my project I would like to do a deep dive into rust and look at the advantages it has over a similar language like C then show off a project with it.

### 3.1 Specification

### 3.2 Prototype

### 3.3 Documentation

### 3.4 Critical Appraisal

...

## 4 Conclusions

(approx 400 words)

In the conclusion, I want a critical reflection on the content of the course. Step back from the technical details. How does the course fit into the wider world of programming languages and software engineering?

## References

[PL] [Programming Languages 2022](#), Chapman University, 2022.