

CPSC-354 Report

Stephanie Munday
Chapman University

September 25, 2022

Abstract

Short summary of purpose and content.

Contents

1	Introduction	1
2	Homework	1
2.1	Week 1	1
2.2	Week 2	2
2.3	Week 3	4
2.4	Week 4	5
3	Project	7
3.1	Specification	7
3.2	Prototype	7
3.3	Documentation	7
3.4	Critical Appraisal	7
4	Conclusions	7

1 Introduction

This is the report for CPSC 354 Programming Languages. It will contain homework for each week, as well as project work and analysis.

2 Homework

This section will contain your solutions to homework.

2.1 Week 1

HW 1 - Greatest Common Divisor

```
def gcd(n, m):  
    while n != m:  
        if n > m:  
            n = n-m  
        else:
```

```
        m = m-n
    return n
```

The code above implements Euclid's algorithm to find the greatest common divisor in python. Below is an explanation given sample input gcd(9,33).

While $n \neq m$, the code will compare whether or not n is greater than m . If $n > m$, n will become $n - m$. Otherwise if $n < m$, m will become $m - n$. When $n == m$, the greatest common divisor has been found.

Keeping this logic in mind, let $n = 9$, $m = 33$.

```
gcd(9,33) =
gcd(9,24) =
gcd(9,15) =
gcd(9,6) =
gcd(3,6) =
gcd(3,3) =
3
```

Since $n == m$ and the value of both is 3, the greatest common divisor is 3 for this example.

2.2 Week 2

HW 2 - Recursion in Functional Programming

```
select_evens :: [a] -> [a]
select_evens [] = []
select_evens (x:(y:xs)) = y:select_evens(xs)

select_odds :: [a] -> [a]
select_odds [] = []
select_odds (x:(y:xs)) = x:select_odds(xs)

member :: (Eq a) => a -> [a] -> Bool
member a [] = False
member a (x:xs)
    | a == x = True
    | otherwise = a `member` xs

append :: (Ord a) => [a] -> [a] -> [a]
append [] [] = []
append [] ys = ys
append (x:xs) (ys) = x:append(xs) (ys)

revert :: [a] -> [a]
revert [] = []
revert (x:xs) = append (revert xs) [x]

less_equal :: (Ord a) => [a] -> [a] -> Bool
less_equal [] [] = True
less_equal (x:xs) (y:ys)
    | x > y = False
    | otherwise = xs `less_equal` ys
```

The code above implements select_evens, select_odds, member, append, revert, less_equal as recursive functions in Haskell. Below are explanations showing computations for given inputs.

Select Evens example:

Select Evens ["a","b","c","d"]

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

Select Odds example:

Select Odds ["a","b","c","d"]

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

Member example:

Member 2 [5,2,6]

```
member 2 [5,2,6] =  
  2 != 5 : (member 2 [2,6]) =  
  2 != 5 : (2 == 2) =  
  True
```

Append example:

Append [1,2,3] [4,5]

```
append [1,2,3] [4,5] =  
  1 : (append [2,3] [4,5]) =  
  1 : (2 : (append [3] [4,5])) =  
  1 : (2 : (3 : (append [] [4,5]))) =  
  1 : (2 : (3 : [4,5])) =  
  [1,2,3,4,5]
```

Revert example:

Revert [1,2,3]

```
revert [1,2,3] =  
  append(revert [2,3], [1]) =  
  append(revert [2,3], [1]) : append(revert [3], [2]) =  
  append(revert [2,3], [1]) : append(revert [3], [2]) : append(revert [], [3]) =  
  [3,2,1]
```

Less Equal example:

Less Equal [1,2,3] [2,3,4]

```
less_equal [1,2,3] [2,3,4] =  
  1 !> 2 : (less_equal [2,3] [3,4]) =
```

```

1 !> 2 : (2 !> 3 : (less_equal [3] [4])) =
1 !> 2 : (2 !> 3 : (3 !> 4 : (less_equal [] []))) =
True

```

2.3 Week 3

HW 3 - Towers of Hanoi

```

hanoi 5 0 2
  hanoi 4 0 1
    hanoi 3 0 2
      hanoi 2 0 1
        hanoi 1 0 2 = move 0 2
        move 0 1
        hanoi 1 2 1 = move 2 1
      move 0 2
      hanoi 2 1 2
        hanoi 1 1 0 = move 1 0
        move 1 2
        hanoi 1 0 2 = move 0 2
      move 0 1
      hanoi 3 2 1
        hanoi 2 2 0
          hanoi 1 2 1 = move 2 1
          move 2 0
          hanoi 1 1 0 = move 1 0
        move 2 1
        hanoi 2 0 1
          hanoi 1 0 2 = move 0 2
          move 0 1
          hanoi 1 2 1 = move 2 1
      move 0 2
    hanoi 4 1 2
      hanoi 3 1 0
        hanoi 2 1 2
          hanoi 1 1 0 = move 1 0
          move 1 2
          hanoi 1 0 2 = move 0 2
        move 1 0
        hanoi 2 2 0
          hanoi 1 2 1 = move 2 1
          move 2 0
          hanoi 1 1 0 = move 1 0
      move 1 2
    hanoi 3 0 2
      hanoi 2 0 1
        hanoi 1 0 2 = move 0 2
        move 0 1
        hanoi 1 2 1 = move 2 1
      move 0 2
      hanoi 2 1 2
        hanoi 1 1 0 = move 1 0
        move 1 2
        hanoi 1 0 2 = move 0 2

```

In order to solve the puzzle, the moves are as follows:

```
move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2
move 0 1
move 2 1
move 2 0
move 1 0
move 2 1
move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2
move 1 0
move 2 1
move 2 0
move 1 0
move 1 2
move 0 2
move 0 1
move 2 1
move 0 2
move 1 0
move 1 2
move 0 2
```

The word "hanoi" appears in the computation 31 times.

This computation can be expressed as a formula that works for moving any number of disks n as:

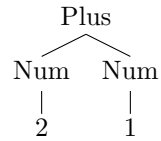
```
hanoi 1 x y = move x y

hanoi (n+1) x y =
  hanoi n x (other x y)
  move x y
  hanoi n (other x y) y
```

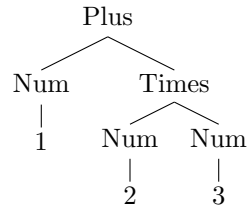
2.4 Week 4

HW 4 - Parsing and Context-Free Grammars

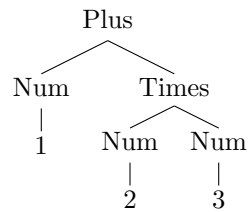
```
Abstract Syntax Tree: 2 + 1
  Plus (Num 2) (Num 1)
```



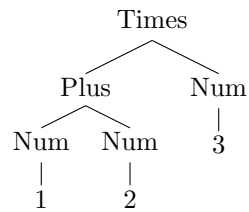
Abstract Syntax Tree: $1 + 2 * 3$
 Plus (Num 1) (Times (Num 2) (Num 3))



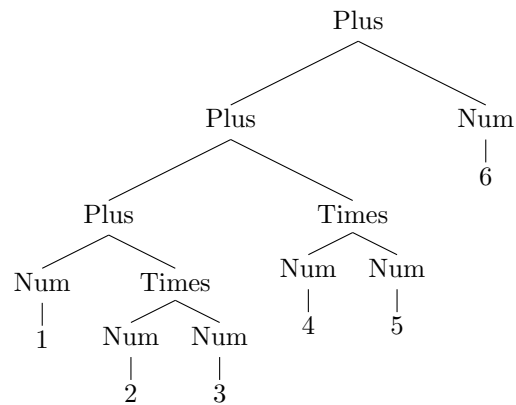
Abstract Syntax Tree: $1 + (2 * 3)$
 Plus (Num 1) (Times (Num 2) (Num 3))



Abstract Syntax Tree: $(1 + 2) * 3$
 Times (Plus (Num 1) (Num 2)) (Num 3)

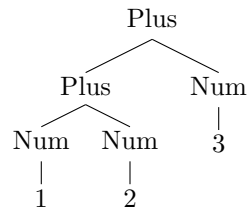


Abstract Syntax Tree: $1 + 2 * 3 + 4 * 5 + 6$
 Plus (Plus (Plus (Num 1) (Times (Num 2) (Num 3))) (Times (Num 4) (Num 5))) (Num 6)



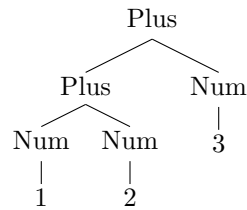
Abstract Syntax Tree: $1 + 2 + 3$

Plus (Plus (Num 1) (Num 2)) (Num 3)



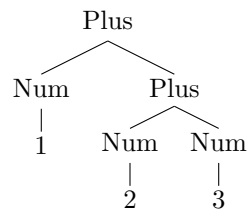
Abstract Syntax Tree: $(1 + 2) + 3$

Plus (Plus (Num 1) (Num 2)) (Num 3)



Abstract Syntax Tree: $1 + (2 + 3)$

Plus (Num 1) (Plus (Num 2) (Num 3))



The abstract syntax tree of $1+2+3$ is identical to the one of $(1+2)+3$, but not the one of $1+(2+3)$.

3 Project

Introductory remarks ...

The following structure should be suitable for most practical projects.

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.

[P] [Punctuation](#), StackExchange, 2022.

[S] [Spacing](#), StackExchange, 2022.

[T] [Trees](#), Massachusetts Institute of Technology, 2022.