CPSC-354 Report

Sharon Chang
Chapman University

September 11, 2022

Abstract

Placeholder

Contents

1 Introduction

Replace this entire Section **??** with your own short introduction.

1.1 General Remarks

First you need to need to download and install LaTeX.[1] For quick experimentation, you can use an online editor such as Overleaf. But to grade the report I will used the time-stamped pdf-files in your git repository.

LaTeX is a markup language (as is, for example, HTML). The source code is in a `.tex` file and needs to be compiled for viewing, usually to `.pdf`.

If you want to change the default layout, you need to type commands. For example, `\medskip` inserts a medium vertical space and `\noindent` starts a paragraph without indentation.

Mathematics is typeset between double dollars, for example

$$x + y = y + x.$$

1.2 LaTeX Resources

I start a new subsection, so that you can see how it appears in the table of contents.

1.2.1 Subsubsections

Sometimes it is good to have subsubsections.

1.2.2 Itemize and enumerate

- This is how you itemize in LaTeX.

- I think a good way to learn LaTeX is by starting from this template file and build it up step by step. Often stackoverflow will answer your questions. But here are a few resources:

  1. Learn LaTeX in 30 minutes
  2. LaTeX – A document preparation system

---

[1] Links are typeset in blue, but you can change the layout and color of the links if you locate the `hypersetup` command.

1.2.3 Typesetting Code

A typical project will involve code. For the example below I took the LaTeX code from stackoverflow and the Haskell code from my tutorial.

```haskell
-- run the transition function on a word and a state
run :: (State -> Char -> State) -> State -> [Char] -> State
run delta q [] = q
run delta q (c:cs) = run delta (delta q c) cs
```

Short snippets such as `run :: (State -> Char -> State) -> State -> [Char] -> State` can also be directly fitted into text. There are several ways of doing this, for example, `run :: (State -> Char -> State) -> State ->` is slightly different in terms of spaces and linebreaking (and can lead to layout that is better avoided), as is

`run :: (State -> Char -> State) -> State -> [Char] -> State`

For more on the topic see Code-Presentations Example.

Generally speaking, the methods for displaying code discussed above work well only for short listings of code. For entire programs, it is better to have external links to, for example, Github or Replit (click on the "Run" button and/or the "Code" tab).

1.2.4 More Mathematics

We have already seen $x + y = y + x$ as an example of inline maths. We can also typeset mathematics in display mode, for example
$$\frac{x}{y} = \frac{xy}{y^2},$$

Here is an example of equational reasoning that spans several lines:

$$\begin{aligned}
\text{fib}(3) &= \text{fib}(1) + \text{fib}(2) & \text{fib}(n+2) &= \text{fib}(n) + \text{fib}(n+1) \\
&= \text{fib}(1) + \text{fib}(0) + \text{fib}(1) & \text{fib}(n+2) &= \text{fib}(n) + \text{fib}(n+1) \\
&= 1 + 0 + 1 & \text{fib}(0) &= 0, \text{fib}(1) = 1 \\
&= 2 & & \text{arithmetic}
\end{aligned}$$

1.2.5 Definitons, Examples, Theorems, Etc

Definition 1.1. This is a definition.

Example 1.2. This is an example.

Proposition 1.3. *This is a proposition.*

Theorem 1.4. *This is a theorem.*

You can also create your own environment, eg if you want to have Question, Notation, Conjecture, etc.

1.3 Plagiarism

To avoid plagiarism, make sure that in addition to [**?**] you also cite all the external sources you use. Make sure you cite all your references in your text, not only at the end.

2 Homework

2.1 Week 1

```python
def numInput():
    while True:
        numStr = input("Enter a number: ")
        try:
            num = int(numStr)
            if num > 0:
                return num
            else:
                print("The number must be greater than 0.")
                continue
        except:
            print("Invalid input.")

def gcd(numA, numB):
    if (numA > numB):
        result = gcd(numA-numB, numB)
    elif (numA < numB):
        result = gcd(numA, numB-numA)
    else:
        result = numA
    return result

# Main
numA = numInput()
print("First number acquired.")
numB = numInput()
print("Second number acquired.")

result = gcd(numA, numB)
print("The GCD of " + str(numA) + " and " + str(numB) + " is: " + str(result) + ".")
```

I chose to write my GCD program in Python.

The numInput function first gets inputs from the user and makes sure that they are valid inputs to perform the gcd function on. The while loop traps the user until they input a valid non-zero integer. This repeats twice to obtain two numbers.

After that, the two numbers are put into the gcd function. If number A is greater than number B, it recursively calls on the function again to find the GCD between the difference of A and B and number B. If number B is greater than number A, it recursively calls on the function again to find the GCD between number A and the difference of B and A. This repeats until the two numbers being compared are equal, after which the function returns the equal number. This equal number is the GCD of the two given numbers. This recursive function works because the method of subtraction will repeatedly lower the numbers until they inevitably equal out to yield a result.

2.2 Week 2

```
select_evens [] = []
select_evens (x:xs) | length (x:xs) >= 2 = (head xs) : (select_evens (tail xs))
                    | length (x:xs) == 1 = []
```

```
                    | otherwise = x:xs

select_odds [] = []
select_odds (x:xs) | length (x:xs) >= 1 = x : (select_odds (tail xs))
                   | otherwise = x:xs

member i [] = False
member i (x:xs) | i == x = True
                | otherwise = member i xs

append [] [] = []
append xs [] = xs
append [] ys = ys
append xs ys = head xs : (append (tail xs) ys)

revert [] = []
revert xs = (last xs) : revert (init xs)

less_equal [] [] = True
less_equal xs [] = False
less_equal [] ys = True
less_equal xs ys | head xs <= head ys = less_equal (tail xs) (tail ys)
                 | otherwise = False
```

For my select evens function, I set the base case to return an empty list. The recursive function has 3 cases. In the first case, provided the given list has a length of 2 or greater, the head of the list xs is taken as the first element of the list. This is the second element of the given list because x is the head of list x:xs. After that the tail of the list xs is put back into select evens, which removes the first two elements from the list x:xs. In the second case, if the length of the list is 1, it also returns an empty list. Otherwise, it just returns the given list.

My select odds function is similar to my select evens function, with a base case of an empty list. If the length of the given list is greater than or equal to 1, the head of list x:xs, which is the value x, is taken as the first element of the list. The tail is then passed back into select odds for the same reasons as the select evens function. If the length is too short, it returns the given list.

The member function returns a base case of False given any input with an empty list. When given a list, as long as the list is greater than or equal to 1, it checks to see if the given input value matches the head of the given list. If it matches, it returns true, otherwise, it passes the tail of the list back into the member function.

When given an empty set of lists, the append function returns an empty list. If only one of the given lists is empty, it returns the non-empty list. When given two non-empty lists, the head of the first list is taken as the first element of the list. It is attached to the resulting list from appending the tail of the first list with the whole of the second list. Eventually, this empties out the first list, which will cause the entirety of the second list to be appended onto the elements of the first list.

The base case of the revert function returns an empty list when given an empty list. When given a non-empty list, the last element of the given list is taken as the first element of the generated list, and then the input list minus its last element is passed back into the revert function.

When the two lists given to the less equal function are both empty, the function returns a true. If the first list is non-empty but the second list is, the function returns false. If the second list is non-empty but the first one is, the function returns true. If both lists are non-empty lists, the head of both lists is compared to one another. If the head of the first list is less than or equal to the second list, the function continues by passing on the tails of both lists back into itself. Otherwise, if the head of the first list is greater than the head of the second list, the function stops and returns a false.

3 Project

Placeholder

3.1 Specification

Placeholder

3.2 Prototype

Placeholder

3.3 Documentation

Placeholder

3.4 Critical Appraisal

Placeholder

4 Conclusions

Placeholder

References

[PL] Programming Languages 2022, Chapman University, 2022.