Programming Paradigms Report

Name: Harry Langham

ID: 15624847

Task One:

Functional Programming:

Functional Programming can be defined as the process of building software with the composition of pure functions. It is declarative rather than imperative and therefore this type of paradigm runs through pure functions in its working.

The structure of functional programming seems to be easier to follow with short blocks of code being concise and simplistic compared to other paradigms. This means that unit testing for applications can be done more easily as it is more straightforward to test the blocks in order than other paradigms with large class structures. In context of the text editor, small blocks of code that control certain functions in the editor can be tested and altered more easily than in other paradigms making it a good choice for function-based applications like the text editor.

A further positive of functional programming in this context is that the code is more closely linked to the problem domain allowing for the route of errors and problems to be more easily identified in the debugging process especially when using GHCI. Often, instead of a simple error code, the application will show a more in-depth explanation of why the specific Haskell app didn't compile. This is very helpful and important in the creation of this project and would be invaluable in further work with this application and compiler.

On the other hand, a disadvantage of using functional programming is the need for recursion meaning application can be more memory intensive than intended. However, on newer machines with 16GB of RAM this is rarely a problem unless the application running is large or highly recursive on multiple modules. In relation, Sorting algorithms used in other modules uses around 650MB on large arrays of millions of integers, meaning the recursion would need to reside with many million integers or values to cause any memory leaks.

Another disadvantage of functional programming is that its hard to create a wholly working application rather than a group of pure functions which aren't too difficult to make.

Procedural Programming:

Procedural programming is a paradigm focussed on a linear approach. This paradigm uses a lot of subroutines and procedures to perform computations that are bound by scope. It's also known as imperative programming. This approach contains data and modules that operate on the data, these are taken as separate entities when the code is finished and compiled.

An advantage of procedural programming based on the context of a text editor is that you would be able to add new subroutines as you wish and due to its procedural nature can be slotted in to a separate procedure. Also, the applications which use procedural are generally faster working and for small message manipulation would be very advantageous.

On the other hand, Procedural programming can be difficult to maintain and manage especially with larger programs which could come as an issue if the text editor would have anymore features added as this could become an increasing issue. When debugging this could become difficult to locate and remove bugs and errors in the code especially if other subroutines are impacted by adding a new feature.

Due to safety concerns highlighted in the brief, it is believed that the use of procedural programming wouldn't be relevant due to its security issues with allowing data to be easily accessible in the whole program. As the client is a VIP who doesn't want their data to be leaked or stolen the choice of this paradigms is beyond the question and will not be effective for this given context and client.

Logical Programming:

Logical programming is a paradigm in which facts and rules are used as statements to be expressed within a formal logic system. Some logical languages are only concerned with output and what needs to be accomplished and not how that is performed.

Logical programming is very flexible in its approach and understanding, meaning that it does not worry about the implementation so that features and function can be reused across the entire program in many ways comparable to polymorphism in the object-oriented paradigm. It also isn't too concerned with how they are set out and is more varied in it's uses and outputs meaning it'll be easier to understand, maintain and manage.

A major disadvantage is that logical programming is highly inefficient and can only deal with logical Boolean statements such as true or false. This severely limits the types of languages you can use as many are simply only calculating Boolean statements. For the given context, this paradigm wouldn't be useful as it would struggle to analyse and manipulate messages and keys in a text editor implementation.

Object-Oriented Programming:

OOP is a model which concerns itself with objects, classes and data. The general idea is that you can manipulate objects themselves, with the focus on logic that's needed to achieve the manipulation being taken away. The big focus on class structure is a big thing in this paradigm and would work exceptional well in the text editor as the class structure would have a main text editor parent class and sub classes relating to the various manipulations.

An advantage of using this type of paradigm is that software development is easier to achieve and more applicable to this paradigm meaning the creation of the text editor would be easier to send out as a working application. This is in part due to the modularity of the paradigm as its meant to be broken in to many different working parts with a branching main class to manipulate the many parts. Objects can be reused with polymorphism as previously explained and this can be used across the whole application and developed into functional objects. This would make the text editor to have many classes and objects that can reused across the entire program. In the context if the text editor, if a section needed to be added that was in relation to another object then it could be reused and altered into a copy of this object to develop the needed feature in the text editor.

Due to the modularity, errors can be isolated to specific objects, functions and classes meaning that debugging and error checking can be completed without having to change a large amount of code.

This is ideal for a text editor with many different sections that could through up errors at any one time.

On the other hand, OOP can be seen as very challenging due to the languages that you have to pick from and their intricacies. The implementation of polymorphism and inheritance may be easy in a small application but as the application builds it will become increasingly challenging and may need a skilled programmer to code and they may take more time to create the application. The particular IDES and languages that use wholly OOP methods are rare and have very little documentation compared to many other languages and paradigms meaning it would be harder to create the application compared to other paradigm and languages with a larger following.

The paradigm that I will choose to use:

After going through all of the above paradigms, I have chosen Functional programming due to it having a large following compared to pure OOP with a large amount of documentation and help compared to the other choice of SmallTalk which seems to have very little documentation that I found helpful. The use of pure declarative functions means that he implementation of a text editor on text messages and keys would be better suited to functional. Also, the code is very concise and easier to understand meaning creating the application was easier for me than trying to use SmallTalk. The ability to debug the functional paradigm easily means that you can remove errors and change parts of the code without altering the other functions which is very helpful.

In conclusion, I believe this is the appropriate paradigm for the brief given due to its concise nature, being easily maintainable and having a large amount of documentation. It is also secure enough for the client to use as they set security as a prominent priority.

Task Two:

Name:

Text Editor – an ADT representing an editorial application for simple textual inputs

Sets:

T	The set of text editors {(x, y, z, b)}
B	The set for clipboard characters {}
C	The set of characters {a…z U A…Z U 0…9}
N	The set for numbers ($\forall$ n.n $\in$ Z $\wedge$ n $\geq$ 0)

Syntax:

Create:	$\perp$	$\rightarrow$	T
Destroy:	T	$\rightarrow$	$\perp$
Init:	T	$\rightarrow$	T

Copy:	T	→	B
Cut:	T	→	B
Paste:	T	→	T
Delete:	T	→	T
countLength:	T X N	→	T
cursorStart:	T	→	(X)
cursorStartWord:	T	→	(X)
cursorEnd:	T	→	(Z)
cursorLeft:	T	→	(X)
cursorRight:	T	→	(Z)
highlightAll:	T	→	(X,Z)
highlightAllRight:	T	→	(Z)
highlightAllLeft:	T	→	(X)
readFile:	T	→	T
writeFile:	T	→	T

Semantics:

Pre-create() :: true
Post-create(o) :: o = []

Pre-destroy(T) :: true
Post-destroy((_,_,_,_);o):: o = □

Pre-init(T) :: true
Post-init((_,_,_,_);o):: o = (x,y,z,b)

Pre-copy(T) :: true
Post-copy((x,y,z,_);o):: o = (x,y,z,b)

Pre-cut(T) :: true
Post-cut((x,y,z,_);o):: o = (_,_,_,b)

Pre-paste(T) :: true
Post-paste((_,_,_,b);o):: o =(x,y,z,_)

Pre-delete(T) :: true
Post-delete((x,y,z,_);o)::o = (x,y,z,_)

Pre-countLength(T , N) :: true
Post-countLength((x,y,z,b);o):: o = length(x+b)

Pre-cursorStart(T) :: true

Post-cursorStart((x,y,z,_);o):: o = (x,_,_,_)

Pre-cursorStartWord(T) :: true
Post-cursorStartWord((x,y,z,_);o):: o = (x,_,_,_)

Pre-cursorEnd(T) :: true
Post-cursorEnd((x,y,z,_);o):: o = (_,_,z,_)

Pre-cursorRight(T) :: true
Post-cursorRight((x,y,z,_);o):: o = (_,_,z,_)

Pre-cursorLeft(T) :: true
Post-cursorLeft((x,y,z,_);o):: o = (x,_,_,_)

Pre-highlightAll(T) :: true
Post-highlightAll((x,y,z,_);o):: o = (x,_,_,_)

Pre-highlightAllRight(T) :: true
Post-highlightAllRight((x,y,z,b);o):: o = (x,z,_,b)

Pre-highlightAllLeft(T) :: true
Post-highlightAllLeft((x,y,z,b);o):: o = (_,x,z,b)

Pre-readFile(T) :: true
Post-readFile((x,y,z,_);o):: o = T

Pre-writeFile(T) :: true
Post-writeFile((x,y,z,b);o):: o = (_,_,_,b)

Author:

Harry Langham (15624847@students.lincoln.ac.uk)

April 2019

Task Three:

Transcript of Text Editor Program:

```
import Data.List

import System.IO

data TextEditor = TextEditor ([Char], [Char], [Char], [Char]) deriving (Show)

-- Sets the type for the string.


-- Insert text string

insertText :: TextEditor

insertText = TextEditor ("I went for a run ", "", "on a rainy day", [])

-- This sets the text in the string that shall be analysed throughout the application.
```

```
-- Count length of string
countLength :: TextEditor -> Int
countLength (TextEditor (x, y, z, b)) = length x + length z
-- This counts the characters to both the right and the left of the cursor in the string.

-- Cursor to left
cursorLeft :: TextEditor -> TextEditor
cursorLeft (TextEditor (x, y, z, b)) = (TextEditor((init x), [], ([last x] ++ z), b))
-- This drops the end character in the string, effectively shifting the cursor once to the left.

-- Cursor to right
cursorRight :: TextEditor -> TextEditor
cursorRight (TextEditor (x, y, z, b)) = (TextEditor((x ++ [head z]), [], (tail z), b))
-- This drops the first character in the string, effectively shifting the cursor once to the right.

-- Cursor to beginning of string
cursorToStart :: TextEditor -> TextEditor
cursorToStart (TextEditor (x, y, z, b)) = (TextEditor ([], [], x, z))
-- This moves the cursor to the beginning of the string.

-- Cursor to end of string
cursorToEnd :: TextEditor -> TextEditor
cursorToEnd (TextEditor (x, y, z, b)) = (TextEditor (x, z, [], []))
-- This moves the cursor all the way to the end of the string.


-- Highlight all characters in string
highlightAll :: TextEditor -> TextEditor
highlightAll (TextEditor (x, y, z, b)) = (TextEditor ([], (x++z),[], b))
-- This selects all of the characters in the string.
```

```
-- Highlight all characters to the right of the cursor
highlightAllRight :: TextEditor -> TextEditor
highlightAllRight (TextEditor (x, y, z, b)) = (TextEditor (x, z, [], b))
-- All characters to the right of the cursor are highlighted.

-- Highlight all characters to the left of the cursor
highlightAllLeft :: TextEditor -> TextEditor
highlightAllLeft (TextEditor (x, y, z, b)) = (TextEditor ([], x, z, b))
-- All characters to the left of the cursor are highlighted.

-- Cut function
cutText :: TextEditor -> TextEditor
cutText (TextEditor (x, y, z, b)) = (TextEditor (x, [], z, y))
-- Text in the string is highlighted and cut to the clipboard.


-- Copy function
copyText :: TextEditor -> TextEditor
copyText (TextEditor (x, y, z, b)) = (TextEditor (x, y, z, y))
-- Text in the string is highlighted and copied to the clipboard.


-- Paste function
pasteText :: TextEditor -> TextEditor
pasteText (TextEditor (x, y, z, b)) = (TextEditor ((x++b), [], z, b))
-- Text from the string that is saved to clipboard is pasted.

-- Delete function
deleteText :: TextEditor -> TextEditor
deleteText (TextEditor (x, y, z, b)) = (TextEditor(x, [], (tail z), b))
```

```
-- This function deletes all characters in the string.


-- Read file
retrieveFile = do
   let file = "textString.txt" -- This tells the application which file to retrieve.
   contents <- readFile file -- This reads the contents of the file.
   return contents -- this displays the contents of the file.


-- Write file
writeTextFile :: TextEditor -> IO()
writeTextFile (TextEditor (x, y, z, b)) =
      writeFile "writtenTextFile.txt" (x ++ y ++ z)
 -- This function writes the string to a text file under the given name.
```

Video Demonstration:

https://youtu.be/jCXuYkAbUjU

References:

Medium. 2017. Master the Javascript Interview. What is Functional Programming?

[Online] Available at: https://medium.com/javascript-scene/master-the-javascript-interviewwhat-is-functional-programming-7f218c68b3a0 [Accessed 24 April 2019]

Belatrix Software Dev Blog. 2019. Why you should consider Functional Programming. [Online]

Available at: https://www.belatrixsf.com/blog/why-you-shouldconsider-functional-programming/. [Accessed 25 April 2019]

What is Logic Programming? 2018. [Online]

Available at: https://www.computerhope.com/jargon/l/logic-programming.htm

[Accessed 25 April 2019]