Introduction

Our language is called Wordy. This language has a functional paradigm. This language is created to perform manipulations on strings.

The Wordy language is useful to perform various operations on strings, such as counting words in large strings, splitting strings, inserting into strings into a specific location and removing a word at a location.

Our language also has features to create functions and name variables.

Design

Core Features - With matching feature menu feature requirements:

We determined core features by thinking of the most basic operations that can be performed on a string which then combined can be used to create more complex operations.

- **Sentence:** *Basic data types and operations.*

A basic string that is the basis for all of our further features. This can encompass just one word as well as multiple words such as a sentence.

- **Num:** *Basic data types and operations.*

A basic integer which will return results for the operations in the string.

- **Count:** *Basic data types and operations.*

A feature that counts the number of words within a sentence. This was implemented as one of the core features because it is a useful operation for many of the more complex features inside of the program..

- **Split:** *Basic data types and operations.*

A feature will split a given string into a list of strings that is each one word in size. This feature is required to perform deeper level operations on a one string to individual words.

- **Cap:** *Basic data types and operations.*

A feature that capitalizes the first word of the string. This is useful to perform operations on if needed when the string of multiple words is split into individual words using the previous Split command.

- **Low:** *Basic data types and operations.*

A feature that lower cases the first word of a string. Same as above, this will be useful on split strings if needed to operate.

- **Reverse:** *Basic data types and operations.*

A feature that reverses the order of words within a sentence. Useful operation for higher order functions.

- **Insert:** *Basic data types and operations.*

A feature that inserts a string into a sentence at a given location. One of the primary features of our languages, to edit strings content.

- **Remove:** *Basic data types and operations.*

A feature that removes words at a given position from a given sentence. Again, one of the primary features of languages to edit strings.

The two features above utilize **Invertibility**, the combination the insert and remove statement can reverse anything that was previously added or removed from string.

- **Let:** *Variables/local names*

Allows the user to give a custom name to an operation. Important feature for building more complex programs.

- **Ref:** *Variables/local names*

A feature references the variable which was created with Let, this is dependent on the other in order for variable naming to work.

- **Fun:** *Procedures/functions with arguments - Recursion/loops.*

A feature that allows users to create a function with the use of our language. This will allow the user to construct programs which are longer and span for multiple commands. Will also allow recursion to be used with this function.

- **App:** *Procedures/functions with arguments - Recursion/loops.*

A feature that calls the function in the first parameter, and passess the function of the second parameter. This is a required feature in order for the Fun to work and users to define custom functions. Will also allow for recursion.

- **Equ:** *Conditionals*

A feature that compares either two integers, bools, or strings. Important operation to design program functionality, to compare strings.

- **IfElse:** *Conditionals*

A conditional used to branch in our language. Operation is important for conditionals and loops.

All of our core features utilize **Orthogonality,** as none of the features overlap in a way that one does the same thing as the other one.

Additional 3 Point Features:

Strings and operations (1)

Our language is based around string manipulation, with many features relating to this requirement.

Static type system (2)

Our language uses a static type system that will allow our programs to be checked for type errors prior to execution.

Library Features:

We determined library features by thinking of useful and frequently used operations which can be implemented with the use of our core features.

- Compare Word Count:

A feature that compares word count of two strings, true if same, false if not.

- Insert Word at end of sentence:

A feature that will insert the specified string at end of another given string

- Insert Word at beginning of Sentence:

A feature that will insert the specified string at beginning of another given string

Sugar Features:

We determined sugar features by thinking of frequently used operations that can be shortened into keywords.

True: two string values are the same. This makes it simpler and shorter to implement the operation rather than the regular way using the Equ command.

False: two string values are different.

And: If both strings are true, use the our If else function. Shorter execution than original.

Or: If either string is true, will also use If else function. Shorter execution than original.

Safety Features:

Using a static type system with the function **typeExpr**, each of the core features has a case to confirm that there is a valid input and will produce an Error result if input is not valid. This check is occurring before program execution, therefore this is a static type.

Runtime errors can still occur when, for example, variables are referenced outside of their scope. We handle this case within the command semantics, returning a “RuntimeError”.

Implementation

Our semantic domain is as follows.

Expr -> Value

We have decided on this domain by having the data type of Epr where all the commands of our language are given, so every input of the language will be in a form of an Expr. The output of Value is another data type that captures all of the possible outputs of our language, even errors. Therefore, this is a valid domain of our language.

This language is used to write programs that manipulate strings.