Go Backwards: An Object Oriented Approach

COMSW4115 - Programming Language and Translators

Tahmid Munat (tfm2109), Shaquan Nelson (sdn2115), Peter Richards (pfr2109), Julian Silerio (jjs2245), Catherine Zhao (caz2114)

I. Describe the language that you plan to implement

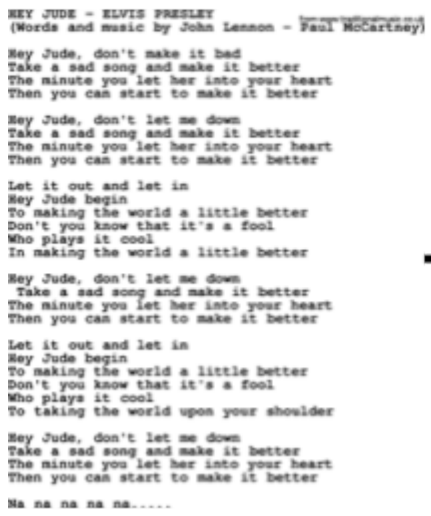
GoBackwards offers an innovative approach to Go and its problems. Go is robust, concurrent, and simple. The simplicity allows users to learn it within a night. Go is a low level language that separates it from others like C because it has garbage collection. As Go becomes more mainstream, the pros for Go get stronger while the cons amplify. Go is not a fully object oriented programming language meaning while you can create structures that share attributes, but there is not inheritance. In Go, a user also cannot overload operators, or add new keywords. This means that users cannot create data types which constricts users for the sake of speed. In addition, Go does not handle GUI very well.

GoBackwards wants to solve these problems by allowing the user to create classes and be able to store them inside of data structures. We want to create a language that allows for quick package imports with extended capabilities. GoBackwards takes a step backwards from Go and attempts to solve the problems that users have found to be annoying and constricting.

II. Explain what sorts of programs are meant to be written in your language

Our goal is to build a very robust low level language that can solve some of the pitfalls from Go. The language will be set up in a similar fashion to Go, so the typical data structures, search algorithms, and file system management will be available in our language. Therefore, this language should technically be fit to be used as a substitute for the languages taught in the lower level computer science classes, and any programs written in such classes, from palindrome checkers to image processing through RGB tuples to creating and searching through trees is feasible with our language.

Our final sample program is to convert song lyrics into a NFA structure. The program will read a text file containing the lyrics to a song. Then it will parse the text into sections and finally create an NFA structure for this particular song. This program demonstrates the capability of handling a advanced data structures, and graphical user interfaces.



HEY JUDE - ELVIS PRESLEY
(Words and music by John Lennon - Paul McCartney)

Hey Jude, don't make it bad
Take a sad song and make it better
The minute you let her into your heart
Then you can start to make it better

Hey Jude, don't let me down
Take a sad song and make it better
The minute you let her into your heart
Then you can start to make it better

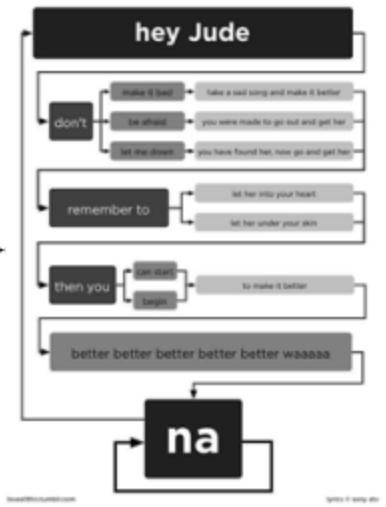
Let it out and let in
Hey Jude begin
To making the world a little better
Don't you know that it's a fool
Who plays it cool
In making the world a little better

Hey Jude, don't let me down
Take a sad song and make it better
The minute you let her into your heart
Then you can start to make it better

Let it out and let in
Hey Jude begin
To making the world a little better
Don't you know that it's a fool
Who plays it cool
To taking the world upon your shoulder

Hey Jude, don't let me down
Take a sad song and make it better
The minute you let her into your heart
Then you can start to make it better

Na na na na na.....


hey Jude
don't
make it bad
take a sad song and make it better
be afraid
you were made to go out and get her
let me down
you have found her, now go and get her
remember to
let her into your heart
let her under your skin
then you
can start
begin
to make it better
better better better better better waaaaa
na

III. Explain the parts of your language and what they do

The section will contain syntax of the program. In particular, data types, numerical operation, logical operation, keywords, control flow, function.

Data Types - Primitive	
int	Integer
double	Floating point number
char	Character
boolean	True, False
string	String

Table 1 - Data Type, Primitive

Data Types - Object	
class	
type	
list	Structure to store values of any type
null	Zip, zill, nada

set	Another structure to store values

Numerical Operations	
+, -, *, /, %	Arithmetic operators
!=, >, <, >=, <=	Comparators
+=, -=, *=, /=	Perform operation and assign to variable

Logical Operations	
and, &&	AND
or, ||	OR
not, !	NOT

IV. Include the source code for an interesting program in your language

https://github.com/pr/GoBackwards

Hello World

```
//Hello World in GoBackwards

package main
import **fmt**
import **nfa**

func main() {
    nfa == NewNFA("Hello", initial)
    nfa.AddState("World", final}

    nfa.AddTransition("Hello" => "World")

    nfa.render
}
```

NFA Represented in GoBackwards

```
//Heavily based on https://github.com/kkdai/nfa, modified for
GoBackwards

//A text file should be parsed, and generate a file like this
one for rendering
//Turn the words in the text file into a NFA
//nfa.render will print out in command line a representation of
the NFA

package main
import **fmt**
import **nfa**

func main() {

  nfa == NewNFA(0, initial)
  nfa.AddState(1)
  nfa.AddState(2)
  nfa.AddState(3, final)
  nfa.AddState(4)

  nfa.AddTransition(0 => 1)
  nfa.AddTransition(1 => 2)
  nfa.AddTransition(2 => 0)
  nfa.AddTransition(2 => 3)
  nfa.AddTransition(3 => 4)
  nfa.AddTransition(4 => 4)

  nfa.render

}
```