UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA

Pikachu Esoteric Language

~Put your pika in everything~

Urcan Denisa-Teodora, 30236

Short history of Pikachu

Its nature is to store up electricity. Forests where nests of Pikachu live are dangerous, since the trees are so often struck by lightning.

Motivation and objective

In today's job market, most Pikachus are forced to live a life of obedience and servitude to cruel Pokemon trainers. They are regularly forced to fight other Pokemon for sport, often in front of large audiences. Clearly this is unethical and a gross violation of Pikachu rights. I firmly believe that to successfully end this inpikachu practice, we first need to provide Pikachus around the world with suitable alternative career options. And the Pikachu programming language intends to do just that. Today, there are innumerable human programmers around the world. IT industries are recruiting millions and millions of human beings. One of the things that's enabling this is that all programming languages out there today are designed in human language, and tailored towards human beings.

Well, not anymore!

Design Principles

The language should be easily usable by Pikachus. Programs should be readable and writable by any Pikachu.

The language should utilise only elements of Pikachu language. This is actually a very good thing since human language is often messy and complicated. But since Pikachu is based on the language of the Pikachus, it can make do with only the following three syntax elements : pi, pika, pikachu.

Coding in Pikachu should be easy if you're a Pikachu. However, if you're a human being and write a bunch of programs in Pikachu, your ability to speak/read/write in the language of Pikachus should improve significantly.

Project Description

Pikachu esoteric language is based on the sounds made by Pikachu from Pokémon, designed specifically to be usable by Pikachus. It is a pretty complex language with only 3 syntax elements: pi, pika and pikachu, and makes use of 2 pikachus (stacks). The complexity of this project will be given by the operations on these 2 stacks. If the two stacks are not specified, the operations will be normal (there is a section down the page with rules that apply in this case). The project is designed using Lex&Yacc, the Lex part being used for syntax recognition. The Yacc part will make use of 2 stacks, combined with some C language for the logical operations.

Project Rules

Since there are only 3 syntax elements, there is a set of rules to operate on them. First and the most important one, repeating the same syntax element 3 times in a row is not allowed. All the rest and therefore the following rules apply to the operations on stacks. They (the stacks/pikachus) will be called Pi Pikachu and Pika Pikachu. So, if a line

ends with pi pikachu or pika pikachu (which it would be the name of one of the two stacks) the following operations will be performed on one of the two stacks, depending on the command preceding it:

- pi pika will add the top element of the pikachu to the next top element, and push the result on to the pikachu
- pika pi will subtract the top element of the pikachu from the next top element, and push the result on to the pikachu
- pi pikachu will multiply the top element of the pikachu to the next top element, and push the result on to the pikachu
- pikachu will divide the top element of the pikachu to the next top element, and push the result on to the pikachu
- pika pikachu will pop the value on top of the pikachu and print it
- pikachu pikachu will pop the value on top of the pikachu and print its equivalent value in ASCII characters
- *blank* will pop the value on top of the pikachu
- *n number of terms* will push the number of terms

If a line doesn't end in the name of one of the two pikachus, there are the following rules:

- pi pika will copy the top of pi pikachu to pika pikachu
- pika pi is the reverse of the precedent operation, so will copy the top of pika pikachu to pi pikachu
- pikachu pikachu will check if the top of pi pikachu and pikachu pikachu are equal, and if they are they will go to line number n, where n is number of terms of immediate next line
- pika pika is also the reverse of precedent operation, so it will check if the top of the two stacks are inequal.

Initially inputs will be added into pi pikachu stack, if there are multiple inputs they will be separated through spaces. The order in which the inputs are pushed on pi pikachu stack is also the order of inputs.

Implementation

The implementation of this project was made using lex&yacc. This includes some simple C libraries, such as stdio, stdlib and string. To work with a personalised stack I created my own files (.c and .h), a set of rules so a word is no repeated more than 3 times in a row, and the specific implementation of those, including the rules mentioned in the above section. All of these are explained in details in the below paragraph.

—lex—

The lex program defined is made to recognise the 3 sounds of a Pikachu: pi, pika and pikachu. This grammar only specifies the rules, so the yacc program will handle them instead of creating the set of regular expressions from here (this is just a personal choice of handling data).

—yacc—

The yacc program implementation has multiple purpose. At the beginning, the sample of yacc program was used to recognise the first and the most important rule of Pikachu programming language: only the 3 sounds of Pikachu can be present, and each one can not be repeated more than 3 times in a row. The development of this language involved operations on two stacks, all basic operations (including adding

-pushing- a sound into a pikachu stack, getting the top of it -pop- etc.) are called in yacc. To simplify the program and the amount of code, the files that are handling this type of data are made separately (stack.h, stack.c) also explained below. The code that recognise those 3 sounds and establish a rule among them, adds each sound in the default pikachu stack (piStack). Afterwards, the last 2 words of piStack are poped out. If the result (defined as a concatenation between the 2 poped words) is the other stack (pikaStack) all the words from the current line will be pushed into that stack. The rest of the code takes again the next 2 words, and if the result is matching one of the rules mentioned in the Project Rules section, they will be handled as defined above.

—C—

The C part defines a header stack.h which contains the definitions for the basic operations on stack, personalised for this problem. Their implementation will be found on stack.c file.

Project examples

—testing the operation that jumps over lines—
pi pi pikachu pika pika pikachu
pikachu pikachu pika pikachu pikachu
pika pi pika pikachu
pi pi pika pika
pika pi pika pi
<<EOF>>

—testing ***pi pika*** before the name of a pikachu stack
pikachu pi pi pika pika pikachu
<<EOF>>

—testing ***pikachu pikachu** before the name of a pikachu stack
pikachu pi pikachu pikachu pika pikachu
<<EOF>>

—a simple program that prints “Hello World”—
pikachu pika pikachu pika pika pi pi pika pikachu pika pikachu pi pikachu pi pikachu pi pika pi pikachu pikachu pi pi pika pika pikachu pika pikachu pikachu pi pika pi pika pika pi pikachu pikachu pi pikachu pi pika pikachu pi pikachu pika pikachu pi pikachu pikachu pi pikachu pika pika pikachu pi pikachu pi pi pikachu pikachu pika pikachu pi pika pi pi pika pika pikachu pikachu pi pi pikachu pi pikachu
pikachu pikachu pi pikachu

pikachu pika pika pikachu pika pikachu pikachu pika pika pikachu pikachu pi pi pikachu pika pikachu pika pika pi pika pikachu pikachu pi pika pika pikachu pi pika pi pika pi pikachu pi pikachu pika pika pi pi pika pi pika pika pikachu pikachu pika pikachu pikachu pika pi pikachu pika pi pikachu pi pika pika pi pikachu pika pi pika pikachu pi pi pikachu pika pika pi pika pi pikachu

pikachu pikachu pi pikachu

pikachu pika pi pika pika pikachu pika pikachu pi pikachu pi pi pika pi pikachu pika pi pi pika pikachu pi pikachu pi pi pikachu pikachu pika pikachu pikachu pika pi pikachu pi pika pikachu pi pikachu pika pika pikachu pika pi pi pikachu pikachu pika pika pikachu pi pika pikachu pikachu pi pika pikachu pikachu pika pi pi pikachu pikachu pi pikachu pi pikachu pi pikachu pi pika pikachu pi pikachu pika pikachu pi pikachu pi pika

pikachu pikachu pi pikachu

pika pi

pikachu pikachu pi pikachu

pikachu pi pikachu pi pi pikachu pi pikachu pika pikachu pikachu pi pikachu pikachu pika pi pi pika pikachu pika pikachu pi pi pikachu pika pi pi pikachu pika pika pi pika pika pikachu pika pikachu pi pi pika pikachu pi pikachu pika pikachu pikachu pika pi pi pikachu pikachu pi pika pikachu pi pikachu pika pikachu pikachu pika pi pikachu pikachu pika pikachu pi pikachu pi pika pikachu pi pikachu pi pika pi pikachu

pikachu pi pikachu

pi pika

pikachu pikachu pi pikachu

pikachu pikachu pi pika pikachu pi pika pika pi pi pika pi pikachu pi pika pi pika pi pika pikachu pika pi pi pikachu pi pikachu pi pika pi pika pika pikachu pi pikachu

pikachu pikachu pi pikachu

pikachu pi pikachu pika pikachu pi pika pi pikachu pikachu pika pika pi pi pikachu pi pika pi pikachu pi pika pikachu pi pika pi pika pikachu pikachu pika pika pikachu pikachu pi pi pikachu pi pikachu pi pikachu pi pi pikachu pikachu pi pikachu pi pikachu pi pika pika pikachu pikachu pika pi pika pikachu pi pikachu pi pi pika pikachu pika pi pikachu pi pika pi pi pikachu pikachu pika pika pikachu pika pika pikachu pi pika pi pika pikachu pi pika pikachu pika pi pika pikachu

pikachu pikachu pika pikachu

pikachu pikachu pika pikachu

pi pi pikachu pi pikachu pika pika pi pikachu pika pika pi pi pika pika pikachu pi pi pikachu pi pika pi pika pikachu pi pikachu pi pikachu pika pikachu pi pi pika pika pi pika pika pi pika pikachu pikachu pi pikachu pika pi pi pika pi pi pikachu pikachu pika pi pi pika pika pi pika pikachu pi pikachu pi pi pika pi pika pika pikachu pika pi pika pikachu pi pikachu pikachu pi pi pika pi pika pika pikachu pikachu pi pikachu

pikachu pikachu pi pikachu

pikachu pi pikachu pikachu pika pikachu pikachu pika pika pikachu pikachu pika pikachu pi pika pikachu pika pika pi pikachu pi pi pika pi pi pikachu pika pika pikachu pikachu pika pikachu pikachu pi pika pi pi pikachu

pikachu pika pi pi pikachu pikachu pika pikachu pika pi pikachu pi pika pi pika pikachu pika pi pikachu pi pikachu pikachu pi pika pikachu pi pikachu pikachu pi pika pi pikachu pikachu pi pikachu pika pika pi pi pikachu
pikachu pi pi pika pi pi pikachu pika pikachu pikachu pika pika pi pi pika pikachu pi pikachu pi pi pika pi pika pi pi pika pikachu pi pika pi pikachu pika pikachu pika pi pi pika pi pi pikachu pi pikachu pikachu pika pi pikachu pi pi pika pi pikachu pi pi pika pi pi pikachu pika pikachu pika pikachu pika pi pikachu pikachu pi pi pika pika pikachu
pikachu pikachu pi pikachu
pikachu pikachu pika pikachu

Future development and difficulties

I did managed to finish quite well the project. However, the things that put me in a difficult situation it was definitely the modelling of the rule with no more than 3 repetitions of the same word in a row, and also the jump operation, which goes to a certain line.

As for the development of this project, I would definitely think of ways to model the substract, multiply and divide operation. Those were not implemented because I did not understand quite right the meaning of them, and they were not explained so I should think of a personal implementation.

Images with the tests mentioned above

```
WORKING WITH ***PI PIKACHU***
pikachu pika pika pikachu pi pi
result = pika pikachu
WORKING WITH ***PIKA PIKACHU***
pika pikachu pi pi
result = pikachu pika

WORKING WITH ***PI PIKACHU***
pikachu pikachu pika pikachu pikachu
result = pikachu pikachu
result *NONE OF STACKS MENTIONED* = pikachu pikachu
TOP OF PIKA = pikachu, TOP OF PI = pikachu
RULE *top of stacks equal* go to line n

I AM AT LINE 3, MOVING OVER 4 LINES
WORKING WITH ***PI PIKACHU***
pika pikachu pikachu pika pi pika
result = pikachu pika
result *NONE OF STACKS MENTIONED* = pikachu pika

I AM AT LINE 4, MOVING OVER 3 LINES
WORKING WITH ***PI PIKACHU***
pikachu pika pi pika pika pika pi pi
result = pika pikachu
WORKING WITH ***PIKA PIKACHU***
pi pi pika pi pika pika pika pi pi
result = pi pi

I AM AT LINE 5, MOVING OVER 2 LINES
WORKING WITH ***PI PIKACHU***
pi pika pi pika
result = pika pi
result *NONE OF STACKS MENTIONED* = pika pi
ERROR ***COULD NOT STEP OVER LINES ***(11111)
```

```
WORKING WITH ***PI PIKACHU***
pikachu pika pika pi pi pikachu
result = pika pikachu
WORKING WITH ***PIKA PIKACHU***
pika pi pi pikachu
result = pi pika
RULE *pi pika* add the top element of the pikachu ***pikachu*** to the next top
element ***pi*** and push the result into pikachu
```

```
WORKING WITH ***PI PIKACHU***
pikachu pika pikachu pikachu pi pikachu
result = pika pikachu
WORKING WITH ***PIKA PIKACHU***
pikachu pikachu pi pikachu
result = pikachu pikachu
RULE *pikachu pikachu* pop the value on top and print ASCII
TOP pikachu, ASCII 112 105 107 97 99 104 117
```

Bibliography

http://trove42.com/pikachu-syntax-rules/