Arabic Linguistic formation in Python for Compiler design

Compilers Construction

By:
Sultan Aljohani (391002603)
Omer Khan (391007603)

Supervised By:
Prof.Ahmed El Khodur

Faculty of Computer and Information Systems

Islamic University of Madinah

Contents

1 Introduction

the object of this project is to make a program that put the right ”تشكيل” on the Arabic sentence

2 Implementation

in this section we will discuss how we develop this project, to make this section easy to understand we will about implementation in terms of logic and programming where in the logic section we will explain the logic of the program and the algorithm used in it and in programming section we will discuss how we implement our logic and which tools we used

2.1 logic

to describe the logic of the program we can be dived into four steps:

2.1.1 Scan the text

the text will be scanned from the file or as input and saved in the memory so the program can do operations on it

2.1.2 Put token of words

the program will take scanner input and split it into words and append each word to a list called words then it will check the type of each word then append the type into another list called tokens, the type checking process is depending on four lists that are imported from four text files that represent correct words for each type, as an example, the lists will look like this:

<name-human> →
احمد | عمر | سلطان
<verb> →
شرب | ذهب | اكل
<name-object> →
البرتقال | التفاح | الماء
<noun> →
صادق | لذيذ | بارد

2.1.3 Apply grammar on the text

depending on the token list the program will check if tokens are sorted in the right sequence according to the grammar, the grammar used in this program is

<text>→ <sentence> |<sentence> <text>
<sentence> → <verbal sentence> | **<noun sentence>**
<verb sentence> → <name-human> <verb> | **<name-object> <name-human> <verb>**
<noun sentence> → <noun> (**<name-human>** | **<name-object>**)
<name-human> →
احمد | عمر | سلطان
<verb> →
شرب | ذهب | اكل
<name-object> →
البرتقال | التفاح | الماء
<noun> →
صادق | لذيذ | بارد

the program will check if the text is following the grammar the according to the grammar the تشكيل will be put in each word in this way:
(<name-human>|<name-object>) <verb>:
(<name-human>|<name-object>): تشكيل → ضمه
<verb>:تشكيل → فتحه
(<name-human>|<name-object>) (<name-human>|<name-object>) <verb>:
<verb> تشكيل → فتحه
(<name-human>|<name-object>):تشكيل → ضمه
(<name-human>|<name-object>) تشكيل → فتحه
<noun> <name>:
<noun> تشكيل → ضمتان
<name> تشكيل → ضمه

after inserting the right تشكيل for each word then it will append to a list called result

2.1.4 Output the result

the program will take the words in the results list then it will join them to make one text then it will print out to the user and also write it to .txt file

2.1.5 Algorithm

In this section, we show the algorithm in the previous logic represent as pseudocode

Algorithm 1 Token the words

Require: $parser \neq NULL$
Require: $verb[] \geq NULL$
Require: $name[] \geq NULL$
Require: $noun[] \geq NULL$

$token[] \leftarrow NULL$
$i \leftarrow 0$
while $parser[i] \neq NULL$ **do**
  if $parser[i]$ in verb **then**
    $token[i] \leftarrow$ "$verb$"
  else if $parser[i]$ in name **then**
    $token[i] \leftarrow$ "$name$"
  else if $parser[i]$ in noun **then**
    $token[i] \leftarrow$ "$noun$"
  else
    $token[i] \leftarrow$ "$not - define$"
  end if
  $i \leftarrow i + 1$
end while

Algorithm 2 Apply grammar on words

Require: $parser \neq NULL$
Require: $verb[] \neq NULL$
Require: $name[] \neq NULL$
Require: $noun[] \neq NULL$
Require: $token[] \neq NULL$

$FATHA \leftarrow u'u064e'$
$DAMMA \leftarrow u'u064f'$
$DAMMATAN \leftarrow u'u064c'$
$result[] \leftarrow NULL$
$i \leftarrow 0$
while $token[i] \neq NULL$ **do**
  if $token[i] ==$ "$verb$" **then**
    if $token[i+1] ==$ "$name$" **then**
      if $token[i+2] ==$ "$name$" **then**
        $result[i] \leftarrow parser[i] + FATHA$
        $result[i+1] \leftarrow parser[i] + DAMMA$
        $result[i+2] \leftarrow parser[i+2] + FATHA$
      else
        $result[i] \leftarrow parser[i] + FATHA$
        $result[i+1] \leftarrow parser[i] + DAMMA$
      end if
    else
      $result[i] \leftarrow$ "$ERROR$"
    end if
  else if $token[i] == name$ **then**
    if $token[i+1] ==$ "$noun$" **then**
      $result[i] \leftarrow parser[i] + DAMMA$
      $result[i+1] \leftarrow parser[i+1] + DAMMATAN$
    else
      $result[i]$"$ERROR$"
    end if
  else
    $result[i] \leftarrow$ "$ERROR$"
  end if
end while

2.2 Programming

in this section, we will discuss the tools that we use to implement the logic and how we code it

the tools we use are:

1. **Python:** Python is a programming language that provides us with readable and high-performance code we use it in this project because python methods are very good at dealing with strings and it supports Arabic which may be difficult to deal with in other programming languages

2. **mercury:** is a tool that we use to make easy-to-use GUI for the user which will increase ease of use for the user

to show how we implement the logic as code we will show the python code for each step

2.2.1 Scan the text

```
f = open(filename,"r")
scanner = f.read()
parser = scanner.split()
```

Listing 1: Scan text from file

```
scanner = input("             ")
parser = scanner.split()
```

Listing 2: Scan text from input

2.2.2 Put token of words:

```
for t in parser:
    if t in verb:
        token.append("verb")
    elif t in name_human:
        token.append("name_human")
    elif t in name_object:
        token.append("name_object")
    elif t in noun:
        token.append("noun")
    else:
        token.append("notD")
```

Listing 3: Put token of words

2.2.3 Apply grammar on the text

```
for i in range(len(token)):
    match token[i]:
        case "verb":
            # <verb sentence>
            match token[i+1]:
                case "name_human":
                    if (i+2 <= len(token)-1):
                        match token[i+2]:
                            case "name_object":
                                # <object> <name_human> <verb>
                                result.append(parser[i]+FATHA)
                                result.append(parser[i+1]+DAMMA)
                                result.append(parser[i+2]+FATHA)
                                result.append("\n")
                            case default:
                                result.append(parser[i]+FATHA)
                                result.append(parser[i+1]+DAMMA)
                                result.append("\n")
                    else:
                        # <name_human> <verb>
                        result.append(parser[i]+FATHA)
                        result.append(parser[i+1]+DAMMA)
                        result.append("\n")
                case "name_object":
                    if (i+2 <= len(token)-1):
                        match token[i+2]:
                            case "name_human":
                                # <object> <name_human> <verb>
                                result.append(parser[i]+FATHA)
                                result.append(parser[i+1]+DAMMA)
                                result.append(parser[i+2]+FATHA)
                                result.append("\n")
                            case default:
                                result.append(parser[i]+FATHA)
                                result.append(parser[i+1]+DAMMA)
                                result.append("\n")
                    else:
                        # <name_object> <verb>
                        result.append(parser[i]+FATHA)
                        result.append(parser[i+1]+DAMMA)
                        result.append("\n")
                case default:
                    result.append(parser[i])
                    result.append(parser[i+1])
                    result.append("      ")
                    result.append("\n")

```

```
48
49          case "name_human":
50              # <noun sentence>
51              if (i+1 <= len(token)-1):
52                  match token[i+1]:
53                      case "noun":
54                          result.append(parser[i]+DAMMA)
55                          result.append(parser[i+1]+DAMMATAN)
56                          result.append("\n")
57
58          case "name_object":
59              # <noun sentence>
60              if (i+1 <= len(token)-1):
61                  match token[i+1]:
62                      case "noun":
63                          result.append(parser[i]+DAMMA)
64                          result.append(parser[i+1]+DAMMATAN)
65                          result.append("\n")
66          case default:
67              result.append("      ")
68              result.append("\n")
```

Listing 4: Apply grammar

2.2.4 Output the result:

```
1      r = " "
2      print("parser: {}".format(parser)) # words in the txt
3      print("tokens: {}".format(token)) # type of each word
4      print("text: {}".format(scanner)) # text befor
5      print("result: \n  {}".format(r.join(result))) # text after
```

Listing 5: Print the output

```
1  try:
2      output_dir = "output_directory"
3      with open(os.path.join(output_dir, "mshkl_text.txt"), "w")
       as fout:
4      fout.write(r.join(result))
5      print("          ")
6      print(os.path.abspath(os.path.join(output_dir, "mshkl_text.
       txt")))
7  except Exception as ex:
8      print(ex.__str__)
```

Listing 6: Save the output to .txt file