НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №5**

з дисципліни **«**Системне програмування 2**»**

Варіант 3

Виконала:

студентка 3 курсу гр. ІО-64

Бровченко А. В.

Перевірив:

Павлов В. Г.

Київ 2018 р.

**Тема:** Створення та на­стро­ювання низхідних синтаксичних аналізаторів на базі використання метамов Бекуса.

**Мета**: Одержання навичок створення механізмів синтаксичного розбору методом рекурсив­ного спуска і створення обробника синтаксичних помилок вхідного тексту.

**Варіант:** 

**Лістинг програми:**

**Main.py**

**from** Lab5\_SP2.Stack **import** Stack  
**import** Lab5\_SP2.Utils **as** ut  
  
  
**def** parse\_to\_tree(string):  
 string += **" "** output = **""** i = 0  
 tabs = **""** st = Stack()  
 st1 = Stack()  
 **while** len(string) > 0:  
  
 root = string[:string.index(**" "**)]  
 **if** root == **"for"**:  
 root = string[:string.index(**"do"**) + 2]  
 string = string[len(root) + 1:]  
 output += **"\t"** \* i + **"for\_loop statement\n"** i += 1  
 **elif** root == **"if"**:  
 root = string[:string.index(**"then"**) + 4]  
 string = string[len(root) + 1:]  
 st1.push(i)  
 output += **"\t"** \* i + **"if\_node\n"** + **"\t"** \* i +**" bool\_expression\n"** + **"\t"** \* i + **" then\_node\n"** i += 1  
 **elif** root == **"begin"**:  
 string = string[len(root) + 1:]  
 st.push(i)  
 *#output += "\t" \* i + "begin\_node" + "\n"* **elif** root == **"end;"**:  
 string = string[len(root) + 1:]  
  
 *#output += "\t" \* st.pop() + "end\_node" + "\n"* **elif** root == **"else"**:  
 string = string[len(root) + 1:]  
 i = st1.pop()  
 output += **"\t"** \* i + **"else\_node"** + **"\n"** i += 1  
 **else**:  
 root = string[:string.index(**"end;"**)]  
 output += **"\t"** \* i + **"expression"** + **"\n"** string = string[len(root):]  
 i += 1  
  
 print(output)  
 *# print(string)***def** check\_syntax(string):  
 check = **True** tokens = [**"for"**, **"to"**, **"do"**, **"begin"**, **"if"**, **"then"**, **"else"**,  
 **"begin"**, **"end"**, **":="**, **">"**, **"<"**, **">="**, **"<="**, **"="**,  
 **"("**, **")"**, **";"**, **"["**, **"]"**]  
 inp = string.split(**" "**)  
  
 **if not** ut.begin\_end\_checker(inp):  
 **return False  
  
 if not** ut.brackets\_checker(string):  
 **return False  
  
 if not** ut.for\_loop\_checker(inp):  
 **return False  
  
 if not** ut.if\_else\_checker(inp):  
 **return False  
  
 if not** ut.check\_semicol(inp):  
 **return False  
  
 return True  
  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *#input\_str = "for i := 1 to n do begin if (a < b) then begin a := a; end; end;"* input\_str = **"if (a < b) then begin for i := 1 to n do begin end; end;"  
  
 if** check\_syntax(input\_str):  
 print(**"Syntax is correct"**)  
 print(**"Tree:"**)  
 parse\_to\_tree(input\_str)  
 print(**"Table:"**)  
 ut.print\_table(input\_str.split(**" "**))  
 **else**:  
 print(**"Syntax error"**)

**Utils.py**

**from** re **import** search  
**from** Lab5\_SP2.Stack **import** Stack  
  
tokens = {**":="**: **"Assign"**, **"-"**: **"Minus"**, **"+"**: **"Plus"**,  
 **"["**: **"Left bracket"**, **"]"**: **"Right Bracket"**,  
 **";"**: **"Semicolon"**, **"="**: **"Equals"**,  
 **">"**: **"Greater"**, **"<"**: **"Lower"**,  
 **">="**: **"Greater or equals"**, **"<="**: **"Lower or equals"**,  
 **"\*"**: **"Multiply"**, **"/"**: **"Divide"**, **"%"**: **"Modular"**,  
 **"for"**: **"For"**, **"to"**: **"To"**, **"do"**: **"Do"**, **"begin"**: **"Begin"**,  
 **"end;"**: **"End"**}  
  
  
**def** begin\_end\_checker(sss):  
 st = Stack()  
 balanced = **True  
 for** t **in** sss:  
 **if** t == **"begin"**:  
 st.push(t)  
 **elif** t == **"end;"**:  
 **if** st.isEmpty():  
 balanced = **False  
 else**:  
 st.pop()  
 **return** balanced **and** st.isEmpty()  
  
  
**def** for\_loop\_checker(inp):  
 **try**:  
 for\_ind = inp.index(**"for"**)  
 **except** ValueError:  
 **return True  
 try**:  
  
 to\_ind = inp.index(**"to"**)  
 do\_ind = inp.index(**"do"**)  
 **except** ValueError:  
 **return False  
  
 if** for\_ind > to\_ind \  
 **or** to\_ind > do\_ind \  
 **or** to\_ind == for\_ind + 1 \  
 **or** do\_ind == to\_ind + 1:  
 **return False  
  
 if not** is\_var(inp[for\_ind + 1]):  
 **return False  
  
 if** inp[for\_ind + 2] != **":="**:  
 **return False  
  
 if not** is\_var(inp[for\_ind + 3]) \  
 **and not** is\_num(inp[for\_ind + 3]):  
 **return False  
  
 if not** is\_var(inp[to\_ind + 1]) \  
 **and not** is\_num(inp[to\_ind + 1]):  
 **return False  
 return True  
  
  
def** if\_else\_checker(inp):  
 **try**:  
 if\_ind = inp.index(**"if"**)  
 then\_ind = inp.index(**"then"**)  
  
 **except** ValueError:  
 **return False  
  
 try**:  
 else\_ind = inp.index(**"else"**)  
 **except** ValueError:  
 else\_ind = 1000  
  
 **if** if\_ind > then\_ind **or** else\_ind < then\_ind **or** then\_ind == if\_ind + 1:  
 **return False  
  
 if** inp[if\_ind + 1][0] != **"("** \  
 **or** inp[then\_ind - 1][-1] != **")"**:  
 **return False  
  
 if not** is\_var(inp[if\_ind + 1][1:]) \  
 **and not** is\_num(inp[if\_ind + 1][1:]) \  
 **or not** is\_var(inp[then\_ind - 1][:-1]) \  
 **and not** is\_num(inp[then\_ind - 1][:-1]):  
 **return False  
  
 if** inp[then\_ind + 1] != **"begin"** \  
 **and not** is\_var(inp[then\_ind + 1]) \  
 **or** else\_ind != 1000 \  
 **and** inp[else\_ind + 1] != **"begin"** \  
 **and not** is\_var(inp[else\_ind + 1]):  
 **return False  
  
 return True  
  
  
def** check\_semicol(inp\_list):  
 **for** i **in** range(len(inp\_list)):  
 **if** inp\_list[i] == **"end;" or** inp\_list[i] == **"else"**:  
 **if** inp\_list[i - 1] != **"else" and not** search(**";"**, inp\_list[i - 1]) **and** inp\_list[i-1] != **"begin"**:  
 **return False  
 return True  
  
  
def** is\_var(ss):  
 res = **False  
 if** len(ss) == 0:  
 res = **False  
 elif** search(**"[A-Za-z]"**, ss[0]):  
 res = **True  
 elif not** search(**"[A-Za-z0-9\_]\*"**, ss[1:]):  
 res = **False  
 return** res  
  
  
**def** is\_num(ss):  
 res = **False  
 if** len(ss) == 0:  
 res = **False  
 elif** search(**"[0-9]"**, ss[0]) **and not** search(**"[A-Za-z]"**, ss):  
 res = **True  
 return** res  
  
  
**def** brackets\_checker(sss):  
 st = Stack()  
 balanced = **True** index = 0  
 **while** index < len(sss) **and** balanced:  
 symbol = sss[index]  
 **if** symbol == **"["**:  
 st.push(symbol)  
 **elif** symbol == **"]"**:  
 **if** st.isEmpty():  
 balanced = **False  
 else**:  
 st.pop()  
 index += 1  
 **return** balanced **and** st.isEmpty()  
  
  
**def** print\_table(inp\_list):  
 **for** string **in** inp\_list:  
 **if** string[0] == **"("**:  
 string = string[1:]  
 print(**"( is a token '('"**)  
 **if** string[-1] == **")"**:  
 string = string[:-1]  
 print(**") is a token ')'"**)  
 **if** string[0] == **"(" and** string[-1] == **")"**:  
 string = string[1:-1]  
 print(**"( is a token '('"**)  
 print(**") is a token ')'"**)  
 **if** string[-1] == **";"**:  
 string = string[:-1]  
 print(**"; is a token 'Semicolon'"**)  
 **if** string **in** tokens:  
 print(string + **" is a token '%s'"** % tokens[string])  
 **else**:  
 has\_tokens = **False** has\_numbers = **False  
 for** c **in** string:  
 **if** c **in** tokens:  
 has\_tokens = **True  
 if** search(**r"[0-9]"**, c):  
 has\_numbers = **True  
 if** has\_tokens:  
 **for** c **in** string:  
 **if** c **in** tokens:  
 print(c + **" is a token '%s'"** % tokens[c])  
 **elif** search(**r"[A-Za-z]"**, c):  
 print(c + **" is a var '%s'"** % c)  
 **elif** search(**r"[0-9]"**, c):  
 print(c + **" is a const '%s'"** % c)  
 **elif** has\_numbers:  
 print(string + **" is a const '%s'"** % string)  
 **else**:  
 print(string + **" is a var '%s'"** % string)

**Stack.py**

**class** Stack:  
 **def** \_\_init\_\_(self):  
 self.items = []  
  
 **def** isEmpty(self):  
 **return** self.items == []  
  
 **def** push(self, item):  
 self.items.append(item)  
  
 **def** pop(self):  
 **return** self.items.pop()  
  
 **def** peek(self):  
 **return** self.items[len(self.items) - 1]  
  
 **def** size(self):  
 **return** len(self.items)

**Контрольні приклади**

**Приклад 1:**

D:\Program\_Files\Python36\python.exe D:/Python/sys\_prog/Lab5\_SP2/Main.py

if (a < b) then begin for i := 1 to n do begin end; end;

Syntax is correct

Tree:

if\_node

bool\_expression

then\_node

for\_loop statement

Table:

if is a var 'if'

( is a token '('

a is a var 'a'

< is a token 'Lower'

) is a token ')'

b is a var 'b'

then is a var 'then'

begin is a token 'Begin'

for is a token 'For'

i is a var 'i'

:= is a token 'Assign'

1 is a const '1'

to is a token 'To'

n is a var 'n'

do is a token 'Do'

begin is a token 'Begin'

; is a token 'Semicolon'

end is a var 'end'

; is a token 'Semicolon'

end is a var 'end'

Process finished with exit code 0

**Приклад 2:**

D:\Program\_Files\Python36\python.exe D:/Python/sys\_prog/Lab5\_SP2/Main.py

if (a < b) then begin for i := 1 to n do begin end end;

Exception: forget semicolon after "end"

Exception: wrong number of "begin" and "end" words

Syntax error

Process finished with exit code 0

**Приклад 3:**

D:\Program\_Files\Python36\python.exe D:/Python/sys\_prog/Lab5\_SP2/Main.py

if (a < b) then for i := 1 to n do begin end; end;

Exception: wrong number of "begin" and "end" words

Syntax error

Process finished with exit code 0

**Приклад 4:**

D:\Program\_Files\Python36\python.exe D:/Python/sys\_prog/Lab5\_SP2/Main.py

if (a < ) then begin for i := 1 to n do begin end; end;

Exception: wrong statement after "if"

Exception: if-then-else checker

Syntax error

Process finished with exit code 0

**Приклад 5:**

D:\Program\_Files\Python36\python.exe D:/Python/sys\_prog/Lab5\_SP2/Main.py

if then begin for i := 1 to n do begin end; end;

Exception: if-then-else checker

Syntax error

Process finished with exit code 0