Міністерство освіти і науки, молоді та спорту України

Національний Технічний Університет України

“Київський Політехнічний Інститут ім. Ігоря Сікорського”

Факультет прикладної математики

Кафедра СПіСКС

**Розрахунково-графічна робота**

*з дисципліни*

“Інженерія програмного забезпечення. Основи проектування трансляторів”

Тема: "Розробка Синтаксичного Аналізатора"

Виконав:

студент групи КВ-42

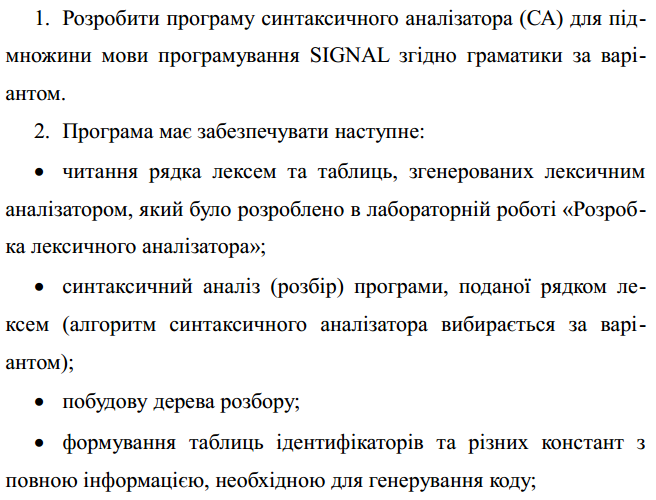
Сахнік Іван

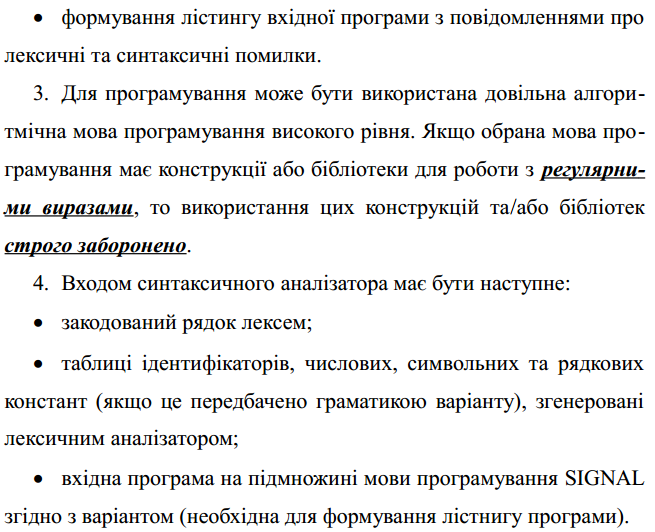
Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ 2017р.

**Постановка завдання**





Варіант 15

1. <signal-program> --> <program>

2. <program> --> PROGRAM <procedure-identifier> ;

<block> ;

3. <block> --> BEGIN <statements-list> END

4. <statements-list> --> <statement> <statements-

list> | <empty>

5. <statement> --> <unsigned-integer> : <statement> |

<variable-identifier> := <unsigned-integer> ; |

<procedure-identifier> <actual-arguments> ; |

GOTO <unsigned-integer> ; |

LINK <variable-identifier> , <unsigned-integer> ; |

IN <unsigned-integer>; |

OUT <unsigned-integer>; |

RETURN ; | ; |

($ <assembly-insert-file-identifier> $)

6. <actual-arguments> --> ( <variable-identifier> <actual-arguments-list> ) | <empty>

7. <actual-arguments-list> --> ,<variable-identifier> <actual-arguments-list> |<empty>

8. <variable-identifier> --> <identifier>

9. <procedure-identifier> --> <identifier>

10. <assembly-insert-file-identifier> --> <identifier>

11. <identifier> --> <letter><string>

12. <string> --> <letter><string> | <digit><string> | <empty>

13. <unsigned-integer> --> <digit><digits-string>

14. <digits-string> --> <digit><digits-string> | <empty>

15. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

16. <letter> --> A | B | C | D | ... | Z

Тестова програма

PROGRAM PROG;

BEGIN

01: A:=10;

B:=20;

MYPROC (A, B);

GOTO 01;

LINK C, 15;

(\*\*\*\*\*\*\* \*\*\*

Hello world!!!

\* \*\*)

IN 55;

OUT 755;

RETURN;

($ ASSEMBLE $)

END;

Код програми

#Parser

#Top-down analysis

#Recursive descent algorithm

from InformationTables import \*

from LexicalAnalyzer import\*

from Tree import \*

error\_table = { "Expected ';'!" : -1,

"Expected 'PROGRAM'!" : -2,

"Wrong ID!" : -3,

"Expected 'BEGIN'!" : -4,

"Expected 'END'!" : -5,

"Wrong NUMBER" : -6,

"Expected '('!" : -7,

"Expected ')'!" : -8,

"Expected ','!" : -9,

"Expected '$)'!" : -10,

"Expected ':'!" : -11,

"Statement ERROR" : -12 }

string\_lexems = runLexicalAnalyzer("MyTestProgram.txt")

tree = Tree()

count = 0 #counter to go to the next token

#function returns the key if the value is found

def scan(dictionary, value):

for key, v in dictionary.items():

if v == value:

return key

def printError(err\_number):

tree.add(err\_number)

tree.current\_element = tree.current\_element.parent\_element

tree.print\_tree()

print("ERROR", err\_number, "!!!", scan(error\_table, err\_number))

quit()

def program(count):

tree.add('program')

lexem = string\_lexems[count]

if lexem == 401:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

procedure\_identifier(lexem);

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

count += 1

count = block(count)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

lexem = string\_lexems[count]

else:

printError(-2)

tree.current\_element = tree.current\_element.parent\_element

def identifier(lexem):

tree.add('identifier')

if lexem >= 1000:

tree.add (scan (identificators\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-3)

tree.current\_element = tree.current\_element.parent\_element

def procedure\_identifier(lexem):

tree.add('procedure-identifier')

identifier(lexem)

tree.current\_element = tree.current\_element.parent\_element

def variable\_identifier(lexem):

tree.add('variable-identifier')

identifier(lexem)

tree.current\_element = tree.current\_element.parent\_element

def assembly\_insert\_file\_identifier(lexem):

tree.add('variable-identifier')

identifier(lexem)

tree.current\_element = tree.current\_element.parent\_element

def block(count):

tree.add('block')

lexem = string\_lexems[count]

if lexem == 402:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-4)

count += 1

count = statements\_list(count)

lexem = string\_lexems[count]

if lexem == 403:

tree.add(scan(key\_words\_dictionary, lexem))

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-5)

tree.current\_element = tree.current\_element.parent\_element

return count;

def unsigned\_integer(lexem):

tree.add('unsigned-integer')

if lexem in range (500, 600):

tree.add (scan (const\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-6)

tree.current\_element = tree.current\_element.parent\_element

def actual\_arguments(count):

tree.add('actual-arguments')

lexem = string\_lexems[count]

if lexem == 59:

tree.add('empty')

tree.current\_element = tree.current\_element.parent\_element

tree.current\_element = tree.current\_element.parent\_element

return count

else:

if lexem == 40:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-7)

count += 1

lexem = string\_lexems[count]

variable\_identifier(lexem)

count += 1

lexem = string\_lexems[count]

if lexem != 41:

count = actual\_arguments\_list(count)

lexem = string\_lexems[count]

if lexem == 41:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-8)

else:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

tree.current\_element = tree.current\_element.parent\_element

return count

def actual\_arguments\_list(count):

tree.add('actual-arguments-list')

lexem = string\_lexems[count]

if lexem == 44:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-9)

count += 1

lexem = string\_lexems[count]

variable\_identifier(lexem)

count += 1

lexem = string\_lexems[count]

if lexem != 41:

count = actual\_arguments\_list(count)

tree.current\_element = tree.current\_element.parent\_element

return count

def statements\_list(count):

tree.add('statement-list')

lexem = string\_lexems[count]

if lexem == 403:

tree.add('empty')

tree.current\_element = tree.current\_element.parent\_element

tree.current\_element = tree.current\_element.parent\_element

return count

else:

count = statement(count)

count += 1

lexem = string\_lexems[count]

if lexem != 403:

count = statements\_list(count)

tree.current\_element = tree.current\_element.parent\_element

return count

def statement(count):

tree.add('statement')

lexem = string\_lexems[count]

if lexem == 404:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

unsigned\_integer(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

elif lexem == 405:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

variable\_identifier(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 44:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-9)

count += 1

lexem = string\_lexems[count]

unsigned\_integer(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

elif lexem==408:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

elif lexem == 406 or lexem == 407:

tree.add (scan (key\_words\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

unsigned\_integer(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

elif lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

elif lexem == 302:

tree.add (scan (separetors\_comp\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count +=1

lexem = string\_lexems[count]

assembly\_insert\_file\_identifier(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 303:

tree.add (scan (separetors\_comp\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-10)

elif lexem in range(500,600):

unsigned\_integer(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 58:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-11)

count += 1

count = statement(count)

elif lexem >= 1000:

prev\_lexem = string\_lexems[count]

count += 1

lexem = string\_lexems[count]

if lexem == 301:

variable\_identifier(prev\_lexem)

tree.add (scan (separetors\_comp\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

count += 1

lexem = string\_lexems[count]

unsigned\_integer(lexem)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

else:

procedure\_identifier(prev\_lexem)

count = actual\_arguments(count)

count += 1

lexem = string\_lexems[count]

if lexem == 59:

tree.add (scan (separetors\_dictionary, lexem) )

tree.current\_element = tree.current\_element.parent\_element

else:

printError(-1)

else:

printError(-12)

tree.current\_element = tree.current\_element.parent\_element

return count

def signal\_program():

program(count)

# if \_\_name\_\_ == '\_\_main\_\_':

# signal\_program()

# tree.print\_tree()

**Стандартний вміст таблиць**

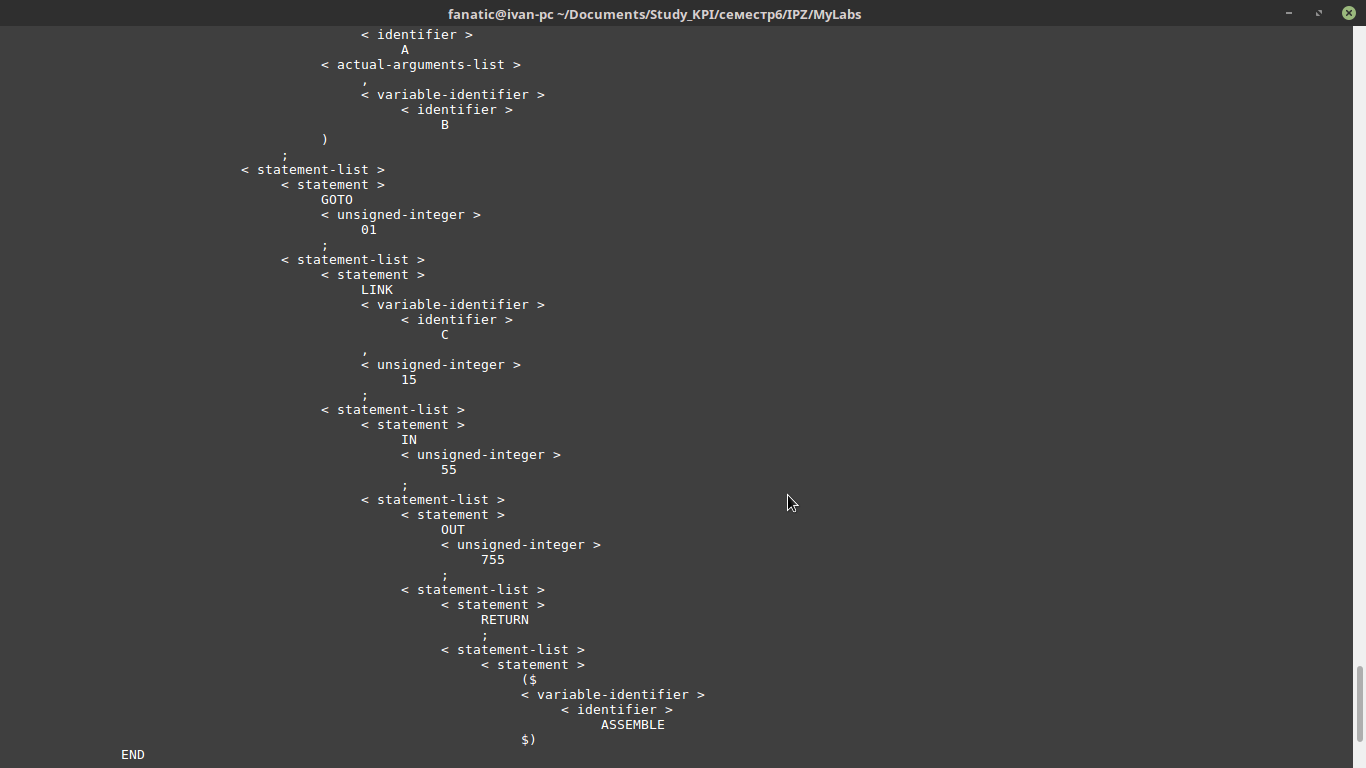
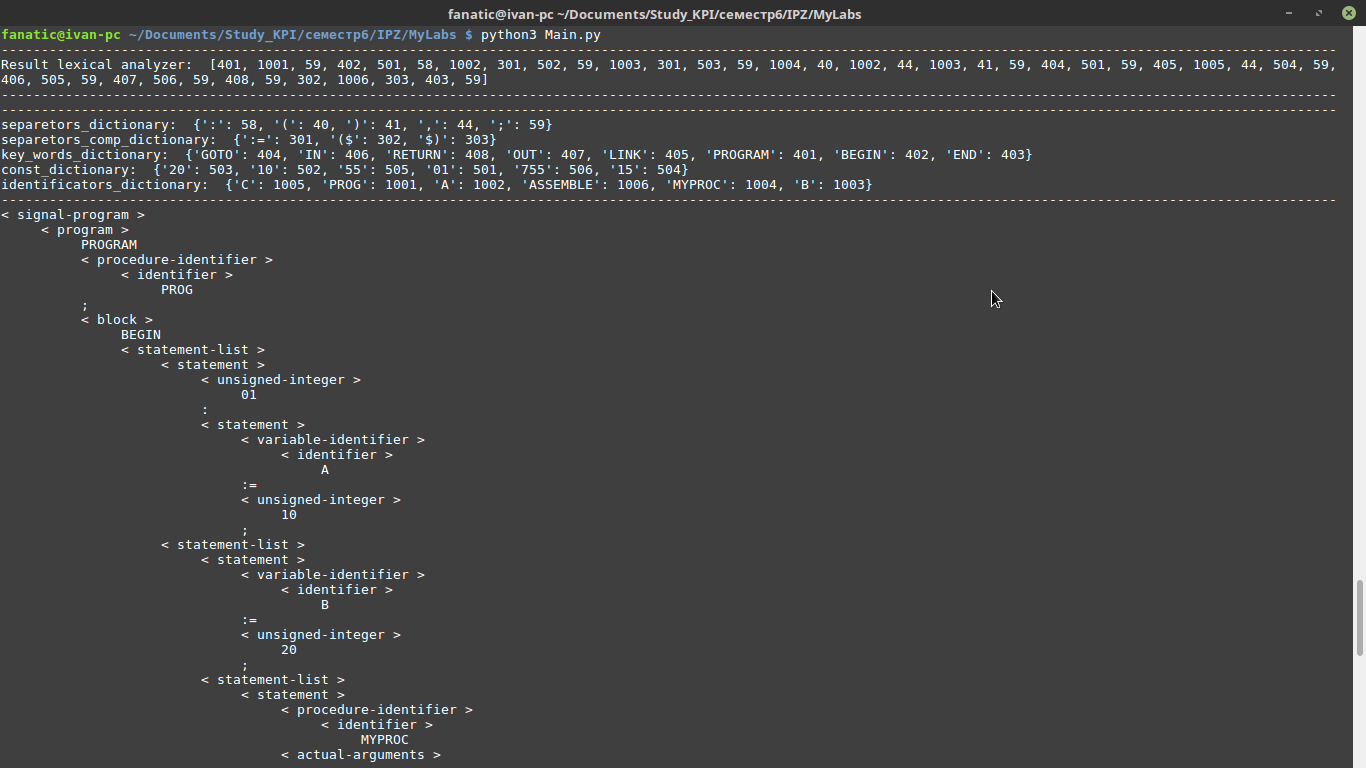
**Таблиця роздільників**

|  |  |
| --- | --- |
| **Код** | **Значення** |
| 59 | ; |
| 58 | : |
| 44 | , |
| 40 | ( |
| 41 | ) |
| 301 | := |
| 302 | ($ |
| 303 | $) |

**Таблиця ключових слів**

|  |  |
| --- | --- |
| **Код** | **Значення** |
| 401 | PROGRAM |
| 402 | BEGIN |
| 403 | END |
| 404 | GOTO |
| 405 | LINK |
| 406 | IN |
| 407 | OUT |
| 408 | RETURN |

**Приклад роботи програми**

****