НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ І ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №5

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

Виконав:

студент групи ІО-42, ФІОТ

Коваленко В’ячеслав Сергійович

ЗК : 4209

м. Київ 2016 р.

Варіант



Лістинг програми

import Analysis as analize

expression = 'if (a>b) then begin end else begin end;'

def main():

analize.parse(expression)

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

main()

expression = ''

tree = []

main\_sequence = []

condition\_token\_list = []

def print\_token\_list(new\_token\_list):

for token in new\_token\_list:

print(token, ' |', t.general\_list.get(token, 'Variable'))

def check\_condition(condition\_list):

#print('Check condition ', condition\_list)

if len(condition\_list) == 0:

exc\_print('empty condition')

return False

token\_list = []

for token in condition\_list:

indexes = []

for i in range(len(token)):

if token[i] in t.brackets\_list:

indexes.append(i)

if len(indexes) > 0:

last = 0

for i in range(len(indexes)): token\_list.append(token[last:indexes[i]])

token\_list.append(token[indexes[i]])

if i != len(indexes) - 1:

last = indexes[i] + 1

else: token\_list.append(token[indexes[i] + 1:])

else:

token\_list.append(token)

token\_list = [x for x in token\_list if x != '']

if token\_list[0] == '(' and token\_list[-1] == ')':

token\_list.pop(-1)

token\_list.pop(0)

else:

exc\_print('error occurred in condition')

return False

new\_token\_list = []

for token in token\_list:

has\_sequence = False

if len(token) > 1:

start = 0

for i in range(len(token)):

if i >= start:

is\_double = False

try:

# check i and i+1

if token[i] + token[i + 1] in t.compare\_list[:4]: new\_token\_list.append(token[start:i])

new\_token\_list.append(token[i] + token[i + 1]) has\_sequence = True

start = i + 2

is\_double = True

except (IndexError):

pass

if not is\_double:

if token[i] in t.compare\_list[4:]:

new\_token\_list.append(token[start:i])

new\_token\_list.append(token[i])

start = i + 1

has\_sequence = True

if not has\_sequence:

new\_token\_list.append(token)

else:

new\_token\_list.append(token[start:])

else:

new\_token\_list.append(token)

new\_token\_list = [x for x in new\_token\_list if x != '']

for el in new\_token\_list:

if el in t.tree\_sequence:

exc\_print('undefined token')

return False

bracket\_list = []

for el in new\_token\_list:

if el in t.brackets\_open:

bracket\_list.append(el)

elif el in t.brackets\_close:

if len(bracket\_list) != 0:

if el == ')' and bracket\_list[len(bracket\_list)-1] == '(':

bracket\_list.pop(len(bracket\_list)-1)

elif el == ']' and bracket\_list[len(bracket\_list)-1] == '[':

bracket\_list.pop(len(bracket\_list)-1)

else:

bracket\_list.append(el)

if len(bracket\_list) != 0:

exc\_print('error occurred in brackets')

return False

for i in new\_token\_list:

if i == '=':

exc\_print('wrong condition statement')

return False

if len(token\_list) == 0:

exc\_print('wrong condition statement')

return False

global main\_sequence

main\_sequence.append('(')

main\_sequence.extend(new\_token\_list)

main\_sequence.append(')')

condition\_token\_list = new\_token\_list[:]

has\_comparison\_operator = False

has\_operation\_operator = False

for token in new\_token\_list:

if token in t.condition\_body:

has\_comparison\_operator = True

if token in t.operation\_list:

has\_operation\_operator = True

if has\_comparison\_operator:

if has\_operation\_operator:

index = 0

count = 0

for i in range(len(new\_token\_list)):

if new\_token\_list[i] in t.condition\_body:

count += 1

if count == 2:

break

else:

index = i

if count > 1:

exc\_print('wrong condition statement')

return False

if index == 0 or index == len(new\_token\_list)-1:

exc\_print('wrong condition statement')

return False

new\_ind = 1

while(True):

if new\_token\_list[index-new\_ind] not in t.general\_list:

if index-new\_ind == 0:

break

else:

if new\_token\_list[index - new\_ind - 1] in t.operation\_list:

if index - new\_ind - 1 == 0:

exc\_print('wrong condition statement')

return False

else:

new\_ind += 2

else:

exc\_print('wrong condition statement')

return False

else:

exc\_print('wrong condition statement')

return False

else:

if len(new\_token\_list) > 3:

exc\_print('wrong condition statement')

return False

else:

try:

if (new\_token\_list[0] not in t.condition\_body) and (new\_token\_list[1] in t.condition\_body) and (new\_token\_list[2] not in t.condition\_body):

return True

else:

exc\_print('wrong condition statement')

return False

except IndexError:

exc\_print('wrong condition statement')

return False

else:

if has\_operation\_operator:

exc\_print('wrong condition statement')

return False

else:

if len(new\_token\_list) > 1:

exc\_print('wrong condition statement')

return False

elif new\_token\_list[0] in t.general\_list:

exc\_print('wrong condition statement')

return False

return True

def full\_check\_body(body\_list):

#print('Full\_check body ', body\_list)

# CASE 3, 4 ELSE, CASE 2, then

if len(body\_list) == 0:

return True

is\_begin = False

for el in body\_list:

indx = el.find('begin')

if indx != -1:

if indx == 0 and len(el) == 5:

is\_begin = True

try:

if indx == 0 and el[5] == ';':

exc\_print('error occurred')

return False

except IndexError:

pass

if is\_begin:

if body\_list[0] == 'begin':

# find end

if body\_list[-1] == 'end;':

# check other words like end; end

for el in t.tree\_sequence:

if el in body\_list[1:-1]:

exc\_print('error in body. Tip (token)')

return False

return True

else:

exc\_print('error occurred with end')

return False

else:

exc\_print('error occurred with begin')

return False

else:

# check other words like end; end

for el in t.tree\_sequence:

if el in body\_list:

exc\_print('error in body. Tip (token)')

return False

return True

def check\_body(body\_list, b\_e=False, semi\_colon=False):

if b\_e:

if len(body\_list) == 0:

exc\_print('error in begin-end block')

return False

if body\_list[0] == 'begin':

if semi\_colon:

if body\_list[-1] == 'end;':

return True

else:

exc\_print('error in end. Tip (semi\_colon)')

else:

if body\_list[-1] == 'end':

return True

else:

exc\_print('error in end. Tip (semi\_colon)')

else:

exc\_print('error in begin-end block')

return False

return True

def choose\_type(token\_list):

#print(token\_list)

if token\_list[0] != 'if':

exc\_print('not if-then-else expression')

return False

tree.append('If\_Node')

main\_sequence.append('if')

then\_index = -1

if 'then' in token\_list:

then\_index = token\_list.index('then')

# cut if-condition

condition\_list = token\_list[1:then\_index]

if check\_condition(condition\_list):

# print condition in tree

tree.append('Boolean\_Expression')

else:

return

if 'else' in token\_list:

else\_index = token\_list.index('else')

if then\_index > else\_index:

exc\_print('error in block then-else')

return False

tree.append(['Then\_Node'])

if check\_body(token\_list[then\_index + 1:else\_index], b\_e=True, semi\_colon=False):

tree[2].append('Then\_Body')

else:

return False

tree.append(['Else\_Node'])

if full\_check\_body(token\_list[else\_index + 1:]):

tree[3].append('Else\_Body')

else:

return False

else:

print('Warning : no else statement')

tree.append(['Then\_Node'])

if full\_check\_body(token\_list[then\_index + 1:]):

tree[2].append('Then\_Body')

else:

return False

else:

exc\_print('"then" block not found')

return False

def print\_tree(arg, tr):

indent = arg

if len(tr) > 0:

for i in tr:

if isinstance(i, list):

print\_tree(2, i)

else:

print(' ' \* indent, i)

indent += 1

def parse(text, b=False):

token\_list = expression.split()

temp = token\_list[:]

choose\_type(token\_list)

if b:

then\_ind = temp.index('then')

if then\_ind != -1:

for i in range(then\_ind, len(token\_list)):

main\_sequence.append(token\_list[i])

for el in temp:

print(el, ' | ', t.general\_list.get(el, 'Variable'))

print\_tree(0, tree)

def exc\_print(text):

print('Exception : ', text)