Ain Shams University
Faculty of Engineering

User Manual for Parser

CSE422: System Software - Academic project

Cairo, 2018

- Acknowledgements:

Thanks to my friends and colleagues for their dedication, great effort and excellent teamworking skills.

- Who we are:

Our Awesome Team	Section
Doaa Kamel Abdelmoneim Ali	2
Mohamed Mohamed El-Morsy	3
Mahmoud Hamdy Abd El-Gawwad	3

- Content:

- Introduction
- How to run?
- Snapshots

- Introduction:

1. Scanner is implemented in C++ under Windows OS using Visual Studio IDE

2. Parser is implemented in python under Windows OS using Python Jupyter

3. The parser calls the scanner and take the scanner’s output(tokens) as its input and continue

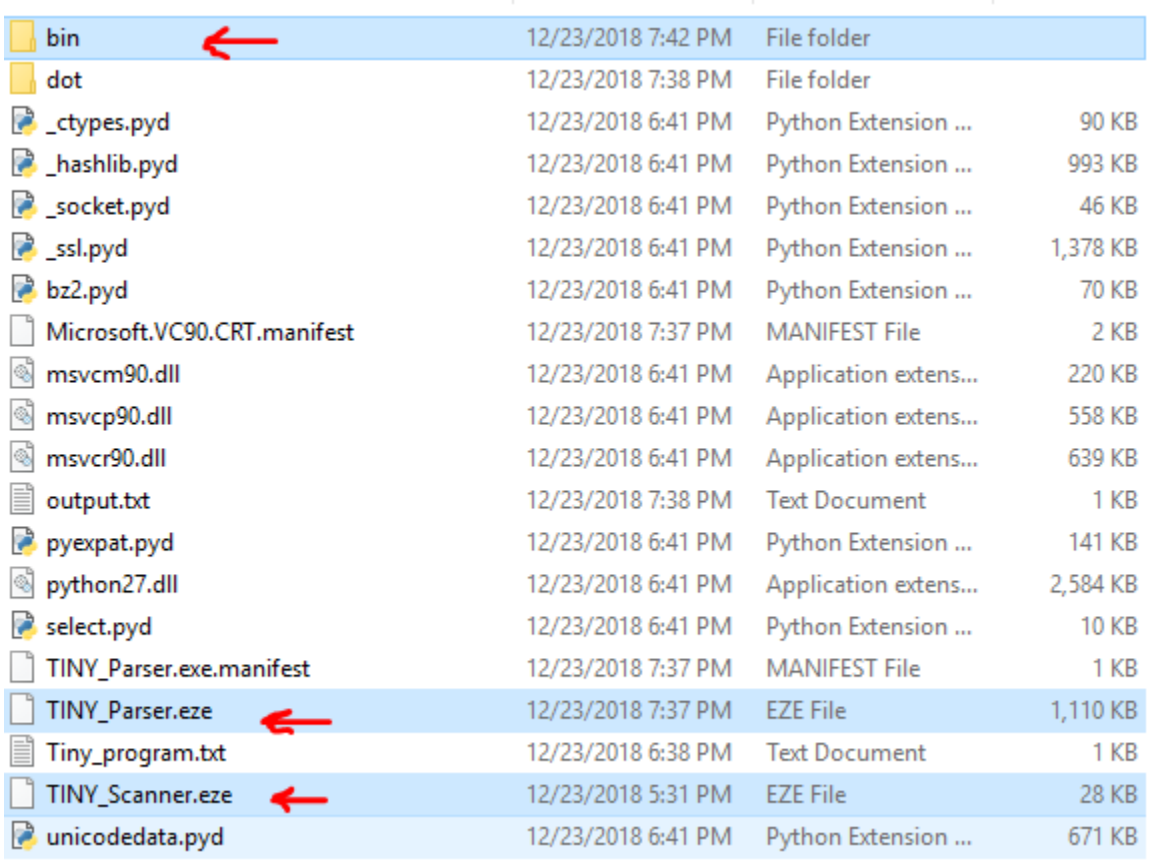
- How to run:

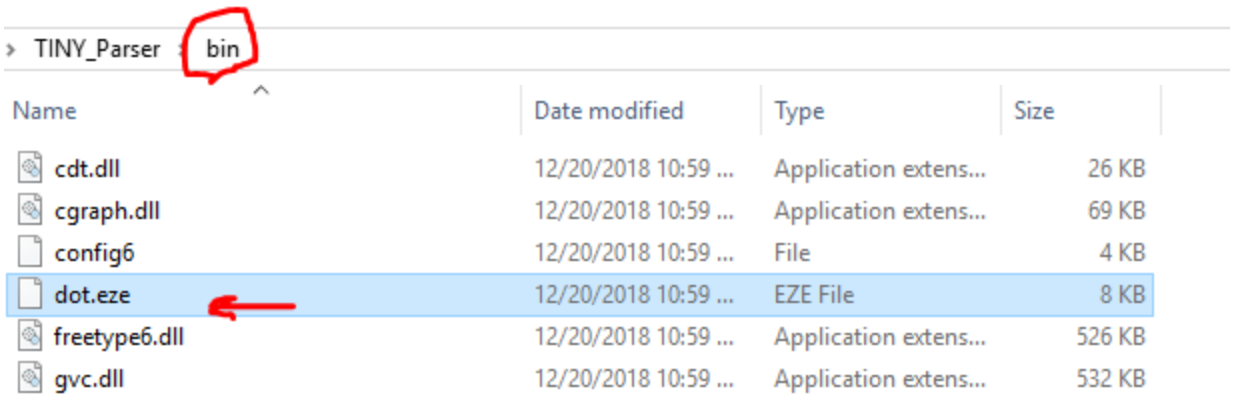
- Make sure that our test file named Tiny_program.txt exists in the same directory as the .exe file.

- Program starts with operating on the TINY code previously provided in the lecture slides.

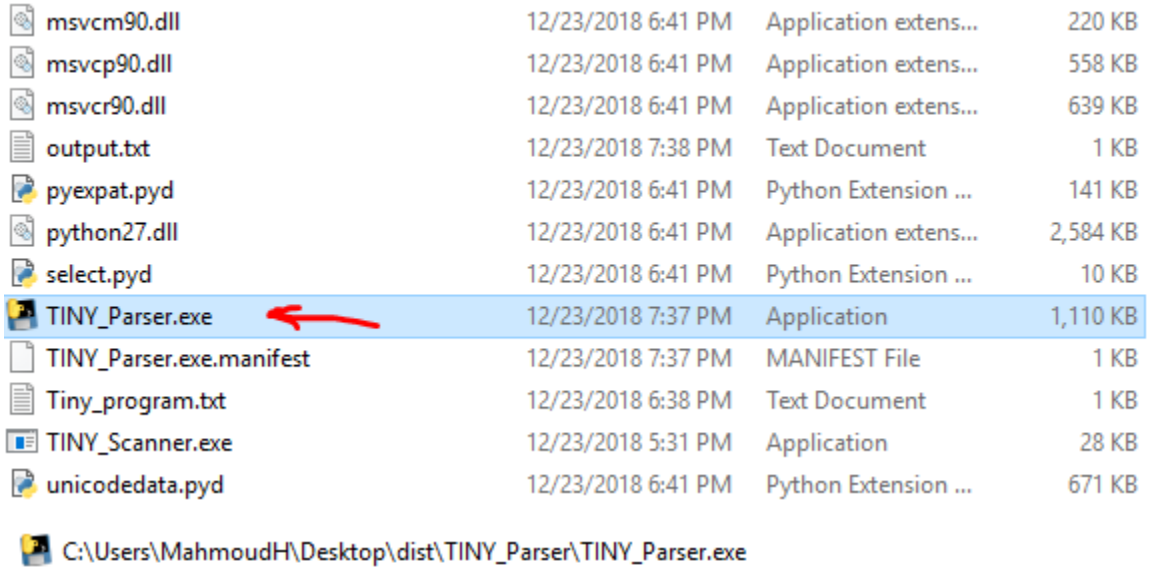
- If you want to use another test for a tiny program, edit the existing provided test file.

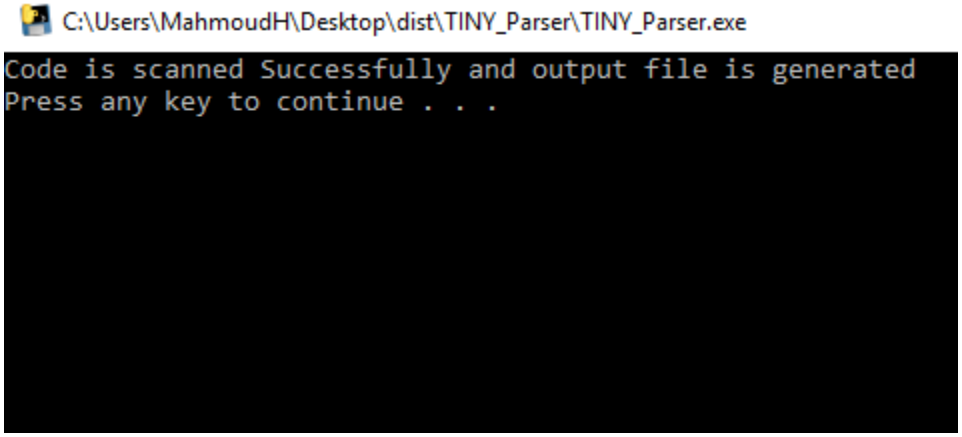
- Change 3 file names from ".eze" to ".exe" which are: TINY_Scanner.eze, TINY_Parser.eze and bin/dot.eze



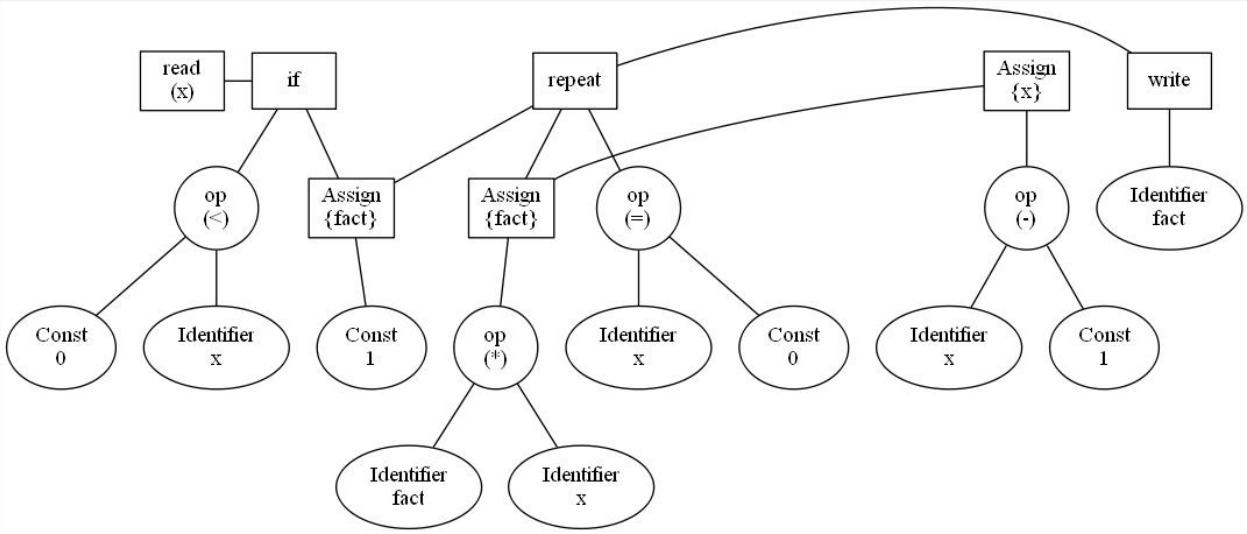


- Open TINY_Parser.exe, a cmd window is shown saying the scanning process is complete





- Press any key to continue, and output is shown:



- Snapshots from parser:

In [329]:
```
import sys
import graphviz as gh
from graphviz import Graph
import os
os.environ["PATH"] += os.pathsep+ os.getcwd() + './bin' #'C:\graphviz-2.38\bin'
```

In [330]:
```
os.system('"TINY_Scanner.exe"')
base_path = "./"
filename = "output.txt"
#path_to_file = os.path.join(base_path, filename)
path_to_file = os.path.join(base_path, filename)
f = open(path_to_file , 'r')
s = "".join(line for line in f)
#print(s)

class graph_node:
    level=-1
    child=0
    parent_id = -1
    sibling_id = -1
    L_child_id = -1
    R_child_id = -1
    extra_child = -1
```

In [331]:
```
parent_edges_list=[]
node_conter=0
```

In [332]:
```
l=s.split('\n')
```

In [333]:
```
lst_of_tokens=[]
count=0
for item in l:
    lst_of_tokens.append(item.split(','))

    
current_node = [graph_node() for i in range (len(lst_of_tokens))]
def correct_nodes(node_index):
    global current_node
    if(node_index == 0):
        return
    elif(current_node[node_index].L_child_id != -1):
        current_node[node_index].level = current_node[current_node[node_index].L_child_id].level
        current_node[current_node[node_index].L_child_id].level += 1
        current_node[node_index].parent_id = current_node[current_node[node_index].L_child_id].parent_id
        current_node[current_node[node_index].L_child_id].parent_id = node_index
        correct_nodes(current_node[node_index].L_child_id)

    if(current_node[node_index].R_child_id != -1):
        current_node[node_index].level = current_node[current_node[node_index].R_child_id].level
        current_node[current_node[node_index].R_child_id].level += 1
        current_node[node_index].parent_id = current_node[current_node[node_index].R_child_id].parent_id
        current_node[current_node[node_index].R_child_id].parent_id = node_index
        #print("right",current_node[node_index].R_child_id,"node index",node_index)
        correct_nodes(current_node[node_index].R_child_id)
```

In [334]:
```
index=0
token=""
mysiblings=[]
def stmt_sequence():
    global token
    current_index = stmt() # returned el index current node
    while(token != "until" and token != "end" and token != "else"):
        match(token) # ; ---> var := const
        current_node[current_index].sibling = stmt()
        # tuning the relationship
        # level
        current_node[current_node[current_index].sibling].level = current_node[current_index].level

        #set edge from current index to sibling
        new.edge(str(current_index),str(current_node[current_index].sibling),constraint='false')
        mysiblings.append(str(current_node[current_index].sibling))
        current_index = current_node[current_index].sibling
        if(flag==1):
            break
        ## write code here
        ##########################
        #current_index = stmt()
```

In [335]:
```
def stmt():
    if lst_of_tokens[index][0]=="if":
        current_index = IF_stmt()
    elif lst_of_tokens[index][0]=="repeat":
        current_index = repeat_stmt()
    elif lst_of_tokens[index][0]=="read":
        current_index = read_stmt()
    elif lst_of_tokens[index][0]=="write":
        current_index = write_stmt()
    else:
        current_index = assign_stmt()

    return current_index
```

In [336]:
```
flag=0
```

In [337]:
```
def match(expected_token):
    global token
    global index
    global flag

    if index == (len(lst_of_tokens) - 2):
        flag=1
        return 'ERROR'
    else:
        index = index + 1
        token=lst_of_tokens[index][0]
        return 'NO ERROR'
    #global token
    #if token==expected_token:
        #if(index!=31):
            #index=index+1
            #return 'NO ERROR'
    #else:
        #return 'ERROR'
```

In [338]:
```
from graphviz import Digraph
new=Graph('Syntax Tree')
new.attr(ordering = "out")
token=lst_of_tokens[index][0]
```

In [339]:
```
def myDraw(ind,text,sh):
    global token
    global index

    new.node(str(ind),text,shape=sh)
    match(token)
    #token=lst_of_tokens[index][0]
```

In [340]:
```
def append_in_edge_list(typ,child_no):
    global node_conter
    global parent_edges_list
    node_conter=node_conter+1
    lst=[]
    lst.append(node_conter)
    lst.append(str(index))
    lst.append(typ)
    lst.append(child_no)
    lst.append('false')
    parent_edges_list.append(lst)
```

In [341]:
```
def read_stmt():
    global token
    global current_node
    # determining the root
    if(index == 0):
        current_node[index].level = 0

    match(token)
    s="read"+"\n"+"("+token +")"
    append_in_edge_list("read",0)
    myDraw(index,s,'rectangle')
    #set_edges(current_node[index - 1].level,-1,0,-1) # index --> -1

    return (index - 1)
```

In [342]:
```
def write_stmt():
    global token
    global current_node
    # determining the root
    if(index == 0):
        current_node[index].level = 0


    current_index = index
    #match(token)
    s='write'
    id=str(index)
    append_in_edge_list("Write",1)
    myDraw(current_index,s,'rectangle')
    current_node[current_index].L_child_id = exp(current_index, current_node[current_index].level)

    # Relationship Tuning
    current_node[current_node[current_index].L_child_id].parent_id = current_index
    current_node[current_node[current_index].L_child_id].level = current_node[current_index].level - 1

    #set_edges(current_node[current_index].level-1,-1,0,id)

    return current_index
```

In [343]:
```
def IF_stmt():
    global token
    global current_node
    global parent_edges_list
    #match(token) will come back later
    s='if'
    id=index
    # detemining the root
    if(index == 0):
        current_node[index].level = 0

    # new edit

    ####################
    append_in_edge_list("if",2)
    item_no=len(parent_edges_list)-1
    fir_val=parent_edges_list[item_no][0]
    sec_val=parent_edges_list[item_no][1]
    myDraw(id,s,'rectangle')
```

```
    current_node[id].L_child_id = exp(id, current_node[id].level)
    match(token) #then

    current_node[id].R_child_id = index
    stmt_sequence()

    if token=="else":
        parent_edges_list[item_no]=[fir_val,sec_val,"if",3,'false']
        match(token) #else
        current_node[id].extra_child = index
        stmt_sequence()


    match(token) #end


    return id
```

In [344]:
```
def exp(parent_index, parent_level):
    global token
    global current_node
    current_index = 0 # just an initialization
    first_operand_index = simple_exp(parent_index, parent_level)
    if(token=="<" or token=="="):
        current_index = index
        current_node[current_index].L_child_id = first_operand_index
        ##correct_nodes(current_index)
        token=lst_of_tokens[index][0]
        id=index
        s="op\n("+token+")"
        append_in_edge_list("op",2)
        myDraw(index,s,'oval')

        #set_edges(level,id,2,-1)
        #new.edge(str(current_index),str(current_node[current_index].L_child_id))

        current_node[current_index].R_child_id = simple_exp(current_index, current_node[current_index].level)
        #set_edges(level,-1,0,-1)
        #new.edge(str(current_index),str(current_node[current_index].R_child_id))

    return current_index
```

In [345]:
```
def simple_exp(parent_index, parent_level):
    global current_node
    global token
    current_index = 0 # just an initialization
    first_operand_index = term(parent_index, parent_level)

    while(token=="+" or token=='-'):
        current_index = index
        current_node[current_index].L_child_id = first_operand_index
        correct_nodes(current_index)
        token=lst_of_tokens[index][0]
        id=index
        s="op\n("+token+")"
        append_in_edge_list("op",2)
        myDraw(index,s,'oval')
        current_node[current_index].R_child_id = term(current_index, current_node[current_index].level) #typo
        first_operand_index = current_node[current_index].L_child_id

    return current_index
```

In [346]:
```
def term(parent_index, parent_level):
    global current_node
    global token
    current_index = 0 # just an initialization
    first_operand_index = factor(parent_index, parent_level)

    while(token=="*" or token=="/"):
        current_index = index
        current_node[current_index].L_child_id = first_operand_index
        ##correct_nodes(current_index)
        token=lst_of_tokens[index][0]
        id=index
        s="op\n("+token+")"
        append_in_edge_list("op",2)
        myDraw(index,s,'oval')

        #set_edges(level,id,2,-1)
        #new.edge(str(current_index),str(current_node[current_index].L_child_id))

        current_node[current_index].R_child_id = factor(current_index, current_node[current_index].level)

        #set_edges(level,-1,0,-1)
        #new.edge(str(current_index),str(current_node[current_index].R_child_id))

        first_operand_index = current_node[current_index].L_child_id

    return current_index
```

In [347]:
```
def factor(parent_index, parent_level):
    global current_node
    global token
    current_index = 0 # just an initialization
    if token=="(":
        match(token)
        #token=lst_of_tokens[index][0]
        current_index = exp(parent_index, parent_level)
        match(token)
        #token=lst_of_tokens[index][0]

    elif lst_of_tokens[index][1].strip()=="Number":
        current_index = index

        current_node[current_index].level = parent_level + 1;
        current_node[current_index].parent_id = parent_index;

        s="Const\n"+token
        append_in_edge_list("const",0)
        myDraw(index,s,'oval')
```

In [348]:
```
def assign_stmt():
    global token
    global current_node
    current_index = 0 #just an initialization

    # determining the root
    if(index == 1 or index == 0):
        current_node[0].level = 0
        current_node[1].level = 0

    if lst_of_tokens[index][1].strip()=="Identifier":
        text=lst_of_tokens[index][0]
    if lst_of_tokens[index+1][0]==":=":
        match(token)
        current_index = index
        s="Assign"+"\n"+"{"+text+"}"
        append_in_edge_list("Assign",1)
        myDraw(index,s,'rectangle')

    current_node[current_index].L_child_id = exp(current_index, current_node[current_index].level)

    return (current_index)
```

In [349]:
```
def repeat_stmt():
    global token
    global current_node
    # determining the root
    if(index == 1 or index == 0):
        current_node[index].level = 0

    s="repeat"
    current_index = index
    append_in_edge_list("repeat",2)
    myDraw(current_index,s,'rectangle') # draw + match repeat

    current_node[current_index].L_child_id = index;

    stmt_sequence()
    match(token) #until

    current_node[current_index].R_child_id = exp(current_index, current_node[current_index].level)

    return current_index
```

In [350]:
```
stmt_sequence()
#new._repr_svg_()
```

In [351]:
```
for item in mysiblings:
    for p_item in parent_edges_list:
        if item==p_item[1]:
            p_item[4]="true"
```

In [352]:
```
i=0
for item in parent_edges_list:
    if item[2]=='op':
        temp=item
        parent_edges_list[i]=parent_edges_list[i-1]
        parent_edges_list[i-1]=temp
    i=i+1
```

In [353]:
```
def child_Draw(i):
    global parent_edges_list
    if(i!=len(parent_edges_list)-1):
        while(parent_edges_list[i][3]!=0):
            #check if sibling
            if(parent_edges_list[i+1][4]=="false"):
                new.edge(parent_edges_list[i][1],parent_edges_list[i+1][1])
                #change no of children
                parent_edges_list[i][3]=parent_edges_list[i][3]-1

            child_Draw(i+1)

        parent_edges_list.remove(parent_edges_list[i])
```

In [354]:
```
while(len(parent_edges_list)>1):
    child_Draw(0)
```

In [355]:
```
new
new.format = 'jpg'
new.render('dot/tree3.gv', view = True)
```