Project Report

on

Programming Language Project

Submitted by

Nont Arayarungsarit st124335

Present to

Phan Minh Dung

Akraradet Sinsamersuk

This report is part of the Programing Language and Compilers

Asian Institute of Technology

May 2024

Architecture of compiler and basic explanation

When someone write some codes as a input. It will pass through lexical analyzer that it will check the word of each code. Each of element of code will separate to a small unit calls token so the token is a word or syntax that it use to check the code that has the word or not when the token is read, it will pass through the syntax analyzer (parsing) to check the correctness of the grammar. If the grammar already exist, it will do some expressions or statements that we have declare before.

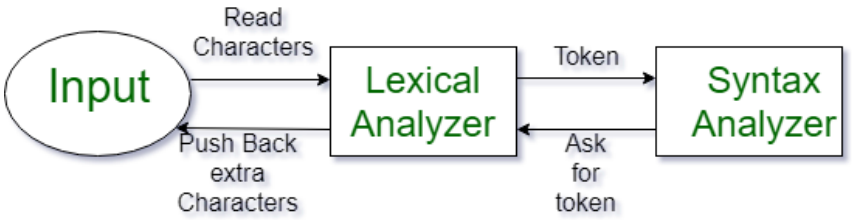

Figure 1 : the relationship between input, lexical analyzer and syntax analyzer

Basically, the code that we write, it will translate from high-level language into low level language, like assembly, it is human-readable and consists specific instructions (e.g., mov, add, jmp) along with operands (registers, memory addresses). If we want to understand the code, we need to understand architecture of computer too. By the way, In this project of compiler course, we will focus on lexical analyzer and syntax analyzer based on python instead.

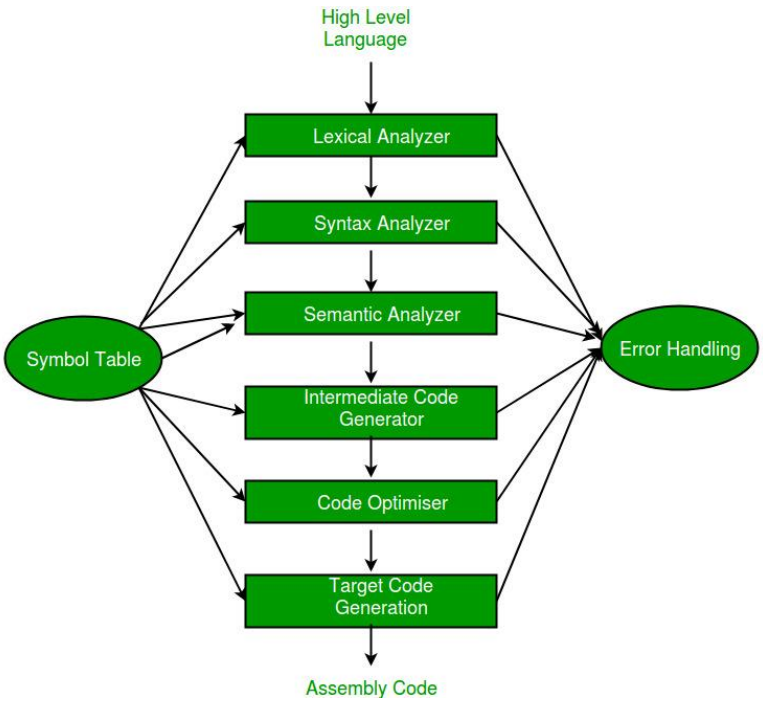

Figure 2 : the process of compiler from high-level language to assembly code

Grammar:

When writing a parser, syntax is usually specified in terms of a BNF grammar. For example, if you wanted to parse simple arithmetic expressions, you might first write an unambiguous grammar specification.

Rule 0 S' -> statement

Rule 1 statement -> PRINT (expr)

Rule 2 statement -> IF (expr) statement

Rule 3 statement -> expr

Rule 4 statement -> NAME = expr

Rule 5 expr -> NAME

#expression of while loop

Rule 6 expr -> WHILE (expr) DO expr

#expression of print

Rule 7 expr -> PRINT (expr)

#expression of string, boolean, float, number, parenthesis, uminus

Rule 8 expr -> STRING

Rule 9 expr -> BOOL

Rule 10 expr -> FLOAT

Rule 11 expr -> NUMBER

Rule 12 expr -> (expr)

Rule 13 expr -> - expr

#expression of if-then-else statement

Rule 14 expr -> IF expr THEN expr ELSE expr

Rule 15 expr -> IF expr THEN expr

 # expression of >, >=, <=, <, ==, !=

Rule 16 expr -> expr NE expr

Rule 17 expr -> expr EQ expr

Rule 18 expr -> expr LE expr

Rule 19 expr -> expr < expr

Rule 20 expr -> expr GE expr

Rule 21 expr -> expr > expr

Rule 22 expr -> expr OR expr

Rule 23 expr -> expr AND expr

 # expression of +, -, x, /

Rule 24 expr -> expr / expr

Rule 25 expr -> expr * expr

Rule 26 expr -> expr - expr

Rule 27 expr -> expr + expr

SLY uses a parsing technique known as LR-parsing or shift-reduce parsing. LR parsing is a bottom up technique that tries to recognize the right-hand-side of various grammar rules. Whenever a valid right-hand-side is found in the input, the appropriate action method is triggered and the grammar symbols on right hand side are replaced by the grammar symbol on the left-hand-side.

LR parsing is commonly implemented by shifting grammar symbols onto a stack and looking at the stack and the next input token for patterns that match one of the grammar rules.

Feature list

The project have some feature lists as follows :

1. Type checking : Assign some value then store into a variables and show the datatype of variables

```
>> x=8
Name    Value   Data Type
-----------------------------
x       8       <class 'int'>
-----------------------------

>> y=3.141
Name    Value   Data Type
-----------------------------
x       8       <class 'int'>
y       3.141   <class 'float'>
-----------------------------

>> z="Hello World!"
Name    Value   Data Type
-----------------------------
x       8       <class 'int'>
y       3.141   <class 'float'>
z       Hello World!    <class 'str'>
-----------------------------

>>
```

2. Do some simple calculations

```
>> p=x+y*1.0101/100
Name    Value   Data Type
-----------------------------
x       8       <class 'int'>
y       3.141   <class 'float'>
z       Hello World!    <class 'str'>
p       8.031727241     <class 'float'>
-----------------------------

>>
```

```
>> p
8.031727241
```

```
>> 3.141/8
0.392625
Name    Value   Data Type
-----------------------------
-----------------------------
```

3. Boolean Expressions: Equality and inequality between two arithmetic expressions.(Logic checking)

```
>> True
True
Name    Value   Data Type
------------------------------
x       8       <class 'int'>
y       3.141   <class 'float'>
z       Hello World!    <class 'str'>
p       8.031727241     <class 'float'>
------------------------------

>> False
False
Name    Value   Data Type
------------------------------
x       8       <class 'int'>
y       3.141   <class 'float'>
z       Hello World!    <class 'str'>
p       8.031727241     <class 'float'>
------------------------------

>> 1>2
False
Name    Value   Data Type
------------------------------
x       8       <class 'int'>
```

```
>> True && False OR True
True
Name    Value   Data Type
------------------------------
------------------------------

>>
```

```
>> True && False OR True == True && False
False
Name    Value   Data Type
------------------------------
------------------------------

>>
```

```
False
Name    Value   Data Type
------------------------------
------------------------------

>> True && False OR True && False != True && False OR True OR True OR True OR True && False
False
Name    Value   Data Type
------------------------------
------------------------------

>> True && False OR True && False == True && False OR True OR True OR True OR True && False
True
Name    Value   Data Type
------------------------------
------------------------------

>> True && False OR True && False == True && False && True OR True && True OR True && False
True
Name    Value   Data Type
------------------------------
------------------------------

>>
```

4. Instructions: Assignment statement, If-then-else, while-loop

```
Parser debugging for MyParser written to parser.out
>> x=5
Name    Value   Data Type
------------------------------
x       5       <class 'int'>
------------------------------

>> if x==5 then print("Hello") else print("World")
Hello
Name    Value   Data Type
------------------------------
x       5       <class 'int'>
------------------------------

>> x=2
Name    Value   Data Type
------------------------------
x       2       <class 'int'>
------------------------------

>> if x==5 then print("Hello") else print("World")
World
Name    Value   Data Type
------------------------------
x       2       <class 'int'>
------------------------------
```

```
>> if x==5 then print("Hello") else print("World")
World
Name    Value   Data Type
------------------------------
x       2       <class 'int'>
------------------------------

>> if x==2 then x=1011001
None
Name    Value   Data Type
------------------------------
x       1011001 <class 'int'>
------------------------------

>> if x==2 then x=1011001
None
Name    Value   Data Type
------------------------------
x       1011001 <class 'int'>
------------------------------

>>
```

5. Others: A print() function.

```
>> x=2
Name     Value    Data Type
------------------------------
x        2        <class 'int'>
------------------------------

>> while(x==2) do print("Dung")
Dung
Dung
Dung
Dung
Dung
Dung
Dung
Dung
```

```
>> while(True&&False) do print("Dung")
None
Name     Value    Data Type
------------------------------
x        3        <class 'int'>
------------------------------

>>
```

```
>> x=3
Name     Value    Data Type
------------------------------
x        3        <class 'int'>
------------------------------

>> while(x==2) do print("Dung")
None
Name     Value    Data Type
------------------------------
x        3        <class 'int'>
------------------------------

>>
```

False Condition

```
Parser debugging for MyParser written to parser.out
>> while(True && 3.141>=1.0101-0.011) do print("I'm weak. I don't know anything about this course.")
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
I'm weak. I don't know anything about this course.
```

Factorial and show decreasing numbers

```
>> y=5
Name    Value   Data Type
------------------------------
y       5       <class 'int'>
------------------------------

>> while(y>0) do print(y) y=y-1
5
4
3
2
1
None
Name    Value   Data Type
------------------------------
y       1       <class 'int'>
------------------------------

>>
```

```
>> f=1
Name    Value   Data Type
------------------------------
y       1       <class 'int'>
f       1       <class 'int'>
------------------------------

>> x=5
Name    Value   Data Type
------------------------------
y       1       <class 'int'>
f       1       <class 'int'>
x       5       <class 'int'>
------------------------------

>> while(x>0) do f=f*x x=x-1 print(f)
120
None
Name    Value   Data Type
------------------------------
y       1       <class 'int'>
f       120     <class 'int'>
x       0       <class 'int'>
------------------------------

>>
```

```
>> y=100
Name    Value   Data Type
------------------------------
y       100     <class 'int'>
f       362880  <class 'int'>
x       0       <class 'int'>
------------------------------

>> while(y>0) do print(y) y=y-5
100
95
90
85
80
75
70
65
60
55
50
45
40
35
30
25
20
15
10
5
None
Name    Value   Data Type
------------------------------
y       5       <class 'int'>
f       362880  <class 'int'>
x       0       <class 'int'>
------------------------------

>>
```

```
>> f=1
Name    Value   Data Type
------------------------------
y       5       <class 'int'>
f       1       <class 'int'>
x       0       <class 'int'>
------------------------------

>> x=8
Name    Value   Data Type
------------------------------
y       5       <class 'int'>
f       1       <class 'int'>
x       8       <class 'int'>
------------------------------

>> while(x>0) do f=f*x x=x-1 print(f)
40320
None
Name    Value   Data Type
------------------------------
y       5       <class 'int'>
f       40320   <class 'int'>
x       0       <class 'int'>
------------------------------

>>
```

Additional command

#Factorial

```
>> f=1

>> x=5

>> while(x>0) do f=f*x x=x-1 print(f)
```

#Show decreasing number

```
>> y=5

>> while(y>0) do print(y) y=y-1
```

How to run project

1.Clone the repository using command “git clone https://github.com/Nont18/compiler-language-project.git”

2.Once you already clone it, you can use “cd compiler-language-project”

3. type “python main.py” or click right of VScode corner to run the project locally.

4.type your command after >> to see the result.

Note : I designed this project based on shell command.

Source code

https://github.com/Nont18/compiler-language-project

References

https://sly.readthedocs.io/en/latest/sly.html

Dr. Dung's slides

https://github.com/akraradets/compiler-starter-project