The Lexer of Pascal

Using Python Language

Outline

- 1.Introduction
- 2.Test data
- 3.Conclusion

Outline

- **1.Introduction**
- 2.Test data
- 3.Conclusion

Introduction

- 1.Constructing a basic Lexer backbone using Python
- 2.Satisfying the additional requirements:
  - 2.1.Handling Hex, Oct, Bin Number
  - 2.2.String
  - 2.3.Big Number

- 1.Constructing a basic Lexer backbone using Python
- Basic Idea: String Manipulation, FSM

- 1.Constructing a basic Lexer backbone using Python
- Basic Idea: String Manipulation, FSM
- For example:

```
RegisterProperty('Owner', 'TComponent', iptRW);
```

pointer

- 1.Constructing a basic Lexer backbone using Python
- Basic Idea: String Manipulation
- For example:

Get a Token<ID, RegisterProperty>

```
RegisterProperty('Owner', 'TComponent', iptRW);
```

Get a Token<LPAREN, ‘(’>

pointer

- 1.Constructing a basic Lexer backbone using Python

- Basic Idea: String Manipulation

- For example:

RegisterProperty('Owner', 'TComponent', iptRW);

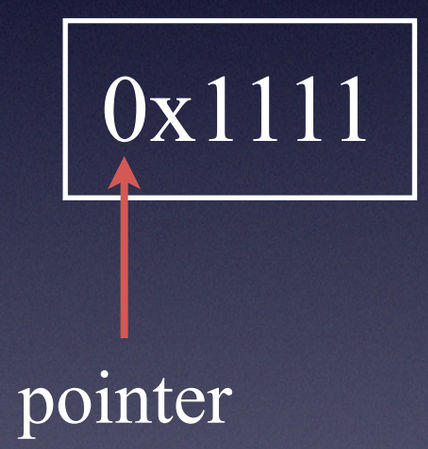
pointer

Get a Token<ID, RegisterProperty>
Get a Token<LPAREN, ‘(’>
Get a Token<String, ‘Owner’>
…
Get a Token<Semicolon, ‘;’>

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

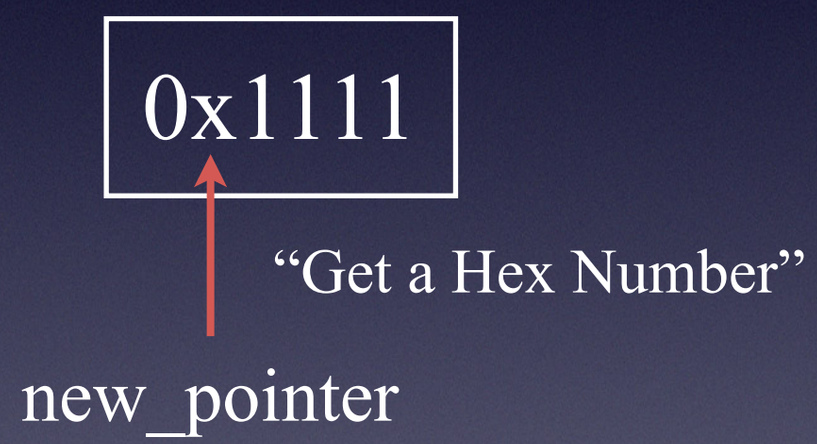
- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

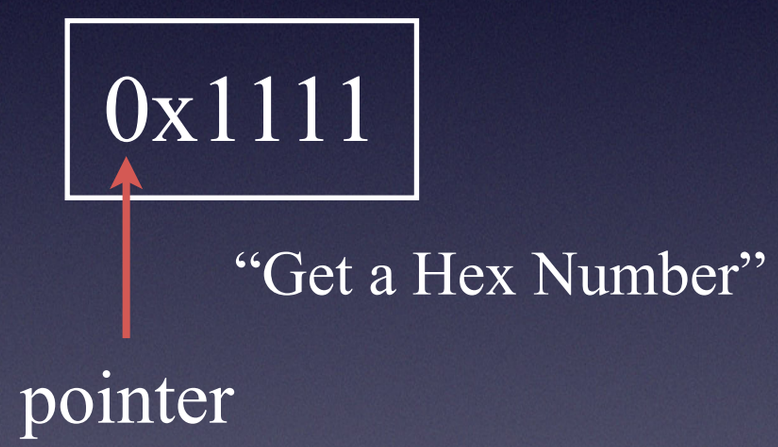
- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

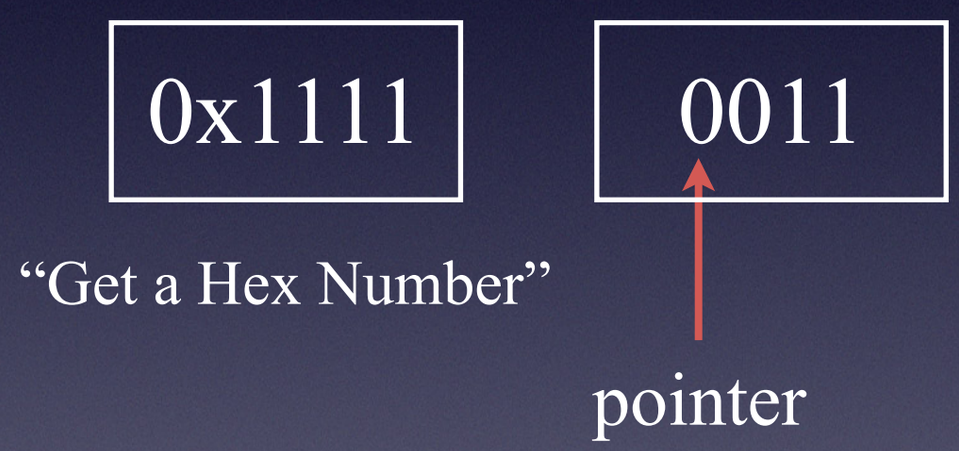
- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

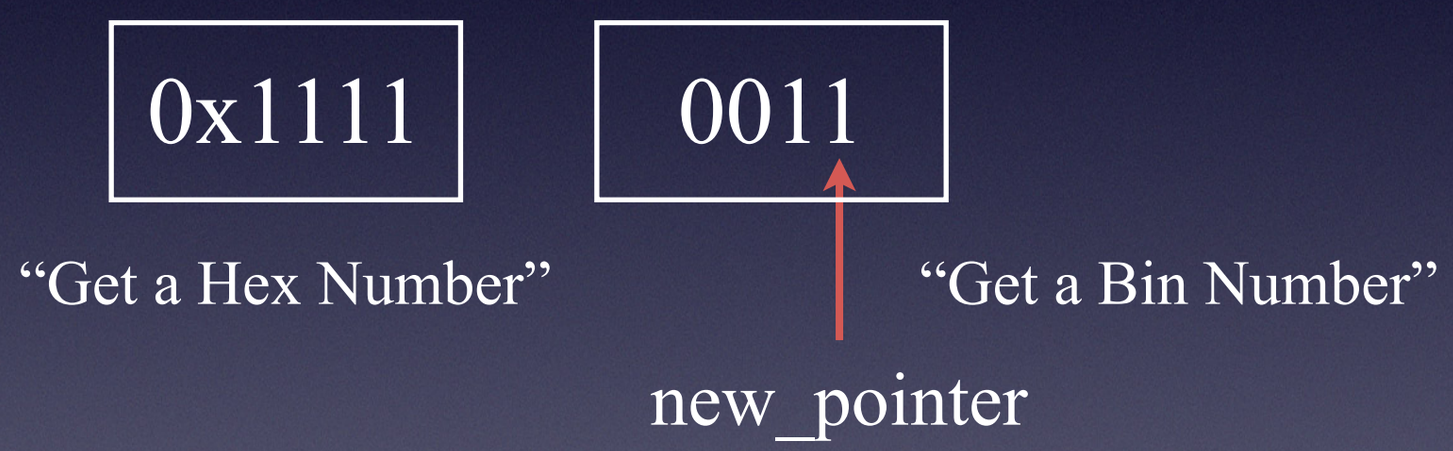
- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

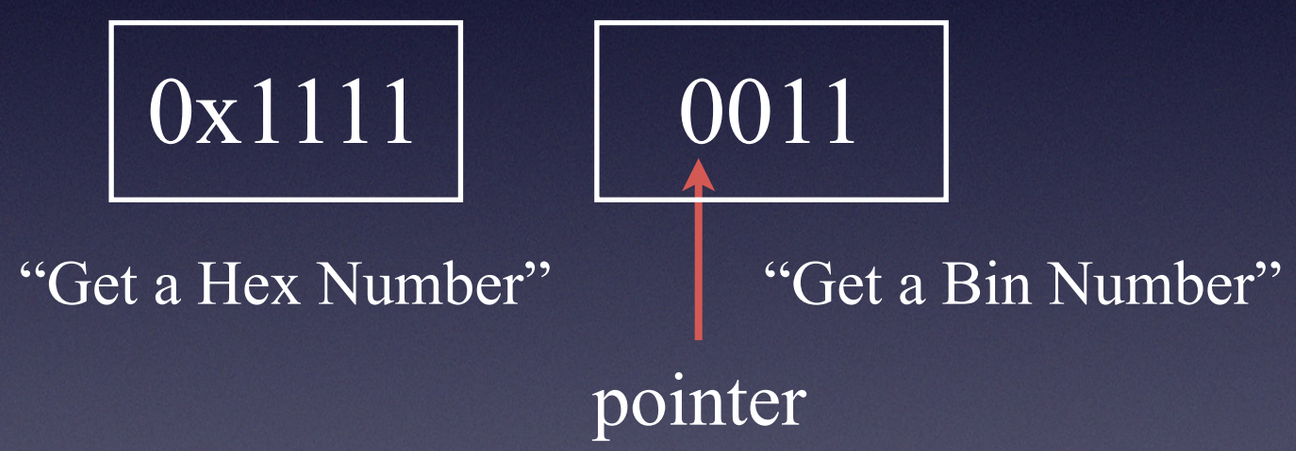
- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:



- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

0x1111	0011	00117721
“Get a Hex Number”	“Get a Bin Number”	

pointer

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

0x1111	0011	00117721
“Get a Hex Number”	“Get a Bin Number”	“Get a Oct Number!”

new_pointer

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

0x1111	0011	00117721
“Get a Hex Number”	“Get a Bin Number”	“Get a Oct Number!”

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

0x1111	0011	00117721	00a
“Get a Hex Number”	“Get a Bin Number”	“Get a Oct Number!”	**“Error!”**

new_pointer

- 2.Satisfying the additional requirements

- Handling Hex, Oct, Bin Number, using “look ahead”

- For example:

0x1111	0011	00117721	00a
“Get a Hex Number”	“Get a Bin Number”	“Get a Oct Number!”	**“Error!”**

- 2.Satisfying the additional requirements

- String, notice **<u>escape character</u>**

- For example:

"abcde\"\\FGhij opqr'st"

escape character: /

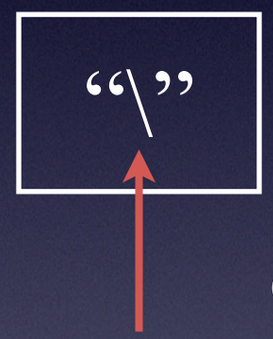
pointer

Solution: Skip?

- 2.Satisfying the additional requirements

- String, notice **<u>escape character</u>**

- Considering another example:



escape character: /

pointer

Solution: Skip?

- 2.Satisfying the additional requirements

- String, notice **escape character**

- For example:

"abcde\"\\FGhij opqr'st"

escape character: /

This is not an individual char, so we can simply skip it.

pointer

- 2.Satisfying the additional requirements

- Big Number, alerting

- For example:

$$k := 0101020102010313192102112;$$

Solution: Compare to INT Threshold,
or compare to the total number of bits

Outline

- 1.Introduction
- **2.Test data**
- 3.Conclusion

Test data

```
1   { test
2     for
3     new
4     lines
5     }
6
7   // this is a comment line
8
9   // this is a comment line
10
11  //
```

Ignore Comments

Test Case 1

Test data

```
1
2
3   a
4
5   a
6   '\n'
7   // ascil 10
8
9   {}
```

Identify '\n'

Compare ASCIL Number

Test Case 2

Test data

It is a very simple test case.

However, take a long time to resolve it.

```
1
2
```

Test Case 3

Test data

It is a very simple test case.

However, take a long time to resolve it.

(1)Analyzing normal ‘\n’ character.

Test Case 3

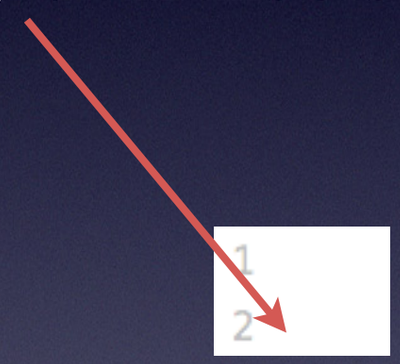
Test data

It is a very simple test case.

However, take a long time to resolve it.

(1)Analyzing normal ‘\n’ character.

(2)Analyzing the end of file, ‘\n’ in this case



Test Case 3

Test data

```
1  " this is a
2  test"
```

New line, report error!

Test Case 4

Test data

```
{ 1 }

a := 07;
b := 0xF;
c := 0011;
d := 0077;
e := 0099;
f := 00ff;
err_oct := 08;

```

Test Case 5

Test data

```
ChenMac:p1 wasdns$ ./run.py programs/test1.pas
Token(ID, 'a')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 7)

Token(SEMI, ';')

Token(ID, 'b')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 15)

Token(SEMI, ';')

Token(ID, 'c')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 3)

Token(SEMI, ';')
```

07

0xF

Test Case 5

Test data

```
1  { Program for testing Pascal Lexer }
2
3  string_1 := "hello";
4  string_2 := "hello world !";
5  string_3 := " hello world ! ";
6  string_4 := "abcde\"\\FGhij  opqr'st"; {abcde"\FGhij  opqr'st}
7  string_5 := "a\"ss\\" {a"ss\}
8
```

Test Case 6

Test data

```
Token(ASSIGN, ':=')
Token(STRING, ' hello world ! ')
Token(SEMI, ';')
Token(ID, 'string_4')
Token(ASSIGN, ':=')
Token(STRING, 'abcde"\\FGhij  opqr\'st')
Token(SEMI, ';')
Token(ID, 'string_5')
Token(ASSIGN, ':=')
Token(STRING, 'a"ss\\')
Token(EOF, None)
ChenMac:p1 wasdns$
```

Standard output

Test Case 6

Test data

```
1  { Program for testing Pascal Lexer }
2
3  string_1 := "hello";
4  string_2 := "hello world !";
5  string_3 := " hello world ! ";
6  string_4 := "abcde\"\\FGhij  opqr'st"; {abcde"\FGhij  opqr'st}
7  string_5 := "a\"ss\\" {a"ss\}
8
```

Test Case 6

Test data

```
Token(INTEGER_CONST, 99)

Token(SEMI, ';')

Token(ID, 'f')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 799)

Token(SEMI, ';')

Token(ID, 'err_oct')

Token(ASSIGN, ':=')

Error: line8, Oct Number Error
Token(INTEGER_CONST, 8)

Token(SEMI, ';')

Token(EOF, None)
```

Error output

Test Case 7

Test data

```
1  { Program for testing Pascal Lexer }
2
3  t := 3;
4  a := 0011; {2}
5  b := 0111; {8}
6  c := 0x11; {16}
7  d := 0023; {8}
8  e := 0087; {10}
9  f := 00AA; {16}
10 error := 00FZ; {error, skip this line}
11
12 long_bin_number := 00101010111010;
13 long_bin_number_with_point := 00101010111010.01010101;
14
15 long_oct_number_1 := 00102345077211;
16 long_oct_number_2 := 07121072154212234;
17 long_oct_number_3 := 00000000000117721;
18 long_oct_number_with_point := 00000000000117721.7717257;
19
20 long_dec_number_1 := 777777218217272121213455;
21 long_dec_number_2 := 0012716713182172711;
22 long_dec_number_3 := 00000000121283189;
23 long_dec_number_with_point := 123456789123456789.123456789;
24
25 long_hex_number := 0x0A000EF00ef1;
26 long_hex_number_with_point := 0x0A000EF00ef1.ffffffaeee;
```

Test Case 8

Test data

```
Token(INTEGER_CONST, 121283189)

Token(SEMI, ';')

Token(ID, 'long_dec_number_with_point')

Token(ASSIGN, ':=')

Token(REAL_CONST, 1.2345678912345678e+17)

Token(SEMI, ';')

Token(ID, 'long_hex_number')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 10995366899953)

Token(SEMI, ';')

Token(ID, 'long_hex_number_with_point')

Token(ASSIGN, ':=')

Token(INTEGER_CONST, 10995366899953)

Token(SEMI, ';')

Token(EOF, None)

ChenMac:p1 wasdns$
```

Test Case 8

Test data

- Other tests are copied from a comprehensive software, named “Pascal Scripts”.
- Open Source at https://github.com/remobjects/pascalscript.
- Along with the given test cases

```
procedure SIRegisterTPersistent(Cl: TPSPascalCompiler);
begin
  with Cl.AddClassN(cl.FindClass('TObject'), 'TPersistent') do
  begin
    RegisterMethod('procedure Assign(Source: TPersistent)');
  end;
end;

procedure SIRegisterTComponent(Cl: TPSPascalCompiler);
begin
  with Cl.AddClassN(cl.FindClass('TPersistent'), 'TComponent') do
  begin
    RegisterMethod('function FindComponent(AName: string): TComponent;');
    RegisterMethod('constructor Create(AOwner: TComponent); virtual;');

    RegisterProperty('Owner', 'TComponent', iptRW);
    RegisterMethod('procedure DestroyComponents');
    RegisterMethod('procedure Destroying');
    RegisterMethod('procedure FreeNotification(AComponent: TComponent)');
    RegisterMethod('procedure InsertComponent(AComponent: TComponent)');
    RegisterMethod('procedure RemoveComponent(AComponent: TComponent)');
    RegisterProperty('Components', 'TComponent Integer', iptr);
    RegisterProperty('ComponentCount', 'Integer', iptr);
    RegisterProperty('ComponentIndex', 'Integer', iptrw);
    RegisterProperty('ComponentState', 'Byte', iptr);
    RegisterProperty('DesignInfo', 'LongInt', iptrw);
    RegisterProperty('Name', 'string', iptrw);
    RegisterProperty('Tag', 'LongInt', iptrw);
  end;
end;
```

pascal-compiler — -bash — 80×24

```
Token(ID, 'SIRegisterTComponent')

Token(LPAREN, '(')

Token(ID, 'Cl')

Token(COLON, ':')

Token(ID, 'TPSPascalCompiler')

Token(RPAREN, ')')

Token(SEMI, ';')

Token(BEGIN, 'BEGIN')

Token(ID, 'with')

Token(ID, 'Cl')

Token(DOT, '.')

Token(ID, 'AddClassN')
```

A string named “TPSPascalCompiler”

Output of analyzing uPSC_std.pas

programs — -bash — 80×24

```
===Test47: uPSR_DB.pas===
 status: passed
===Test48: uPSR_dll.pas===
 status: passed
===Test49: uPSR_extctrls.pas===
 status: passed
===Test50: uPSR_forms.pas===
 status: passed
===Test51: uPSR_graphics.pas===
 status: passed
===Test52: uPSR_menus.pas===
 status: passed
===Test53: uPSR_std.pas===
 status: passed
===Test54: uPSR_stdctrls.pas===
 status: passed
===Test55: uPSRuntime.pas===
 status: passed
===Test56: uPSUtils.pas===
 status: passed

Congratulations: All the tests of lexer have been passed! Total: 57 tests.

ChenMac:programs wasdns$
```

Each of the tests runs our lexer program

57 Tests copied from PascalScripts

Output of unit tests

tockenizer.py UNREGISTERED

topo.py | process.py | tockenizer.py | run.py | integer_type.pas | processor.py

```python

    """Lexer: Acquiring Tokens"""

    def get_next_token(self):
        """Lexical analyzer (also known as scanner or tokenizer)
        This method is responsible for breaking a sentence
        apart into tokens. One token at a time.
        """
        while self.current_char is not None:

            # New line
            if ord(self.current_char) == ord('\n'):
                self.go_newline()
                if self.current_char == None:
                    break

            # For ignoring "//" comments, skip them when lexer analysing
            if self.current_char == '/':
                # if meet comments, skip
                if self.look_ahead(self.pos) == '/':
                    self.skip_another_comment()
                else:
                    self.advance()
                    return Token(FLOAT_DIV, '/')

                if self.current_char == None:
                    break

            # White Space
            if self.current_char.isspace():
                self.skip_whitespace()
                continue

            # Character
            if self.current_char.isalpha() or self.current_char == '_':
```

Line 121, Column 1 | Tab Size: 4 | Python

Overview of our program

Outline

- 1.Introduction
- 2.Test data
- **3.Conclusion**

Conclusion

- Mechanism: Learning by Doing(LBD)
- Driver: Interest, Technology
- Require some efforts to make it running

That’s all.

Thank you!

Group Members: 吴媛媛, 林诗尧, 陈翔