Document File for Scanner

陈睿　Chen, Rui

李亦初　Li, Yichu

王虹博　Wang, Hongbo

Document File for Scanner

0. Develop environment

Using c++ and visual studio code.

The c++ version must be higher than c++11, the following is the compile command

> g++ -std=c++11 automata.cpp list.cpp main.cpp token.cpp tools.cpp -o main

A test file is appended with the souce code. If you compile the program with the above command, you can try the test.pym file with the following command.

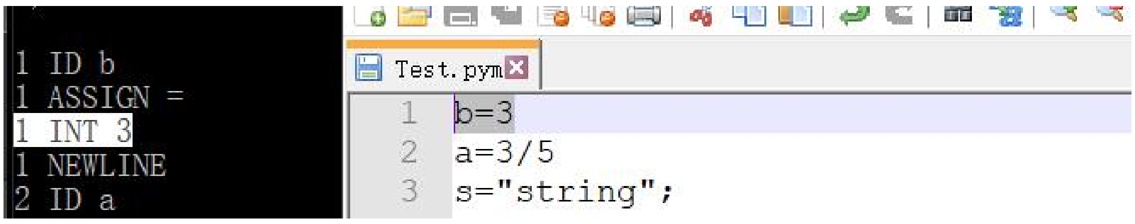
> main Test.pym

1. Features

- Comments are ignored by scanner.
- Integer

  Integers are denoted as ***INT*** token, numbers are represented by three token (two ***INT*** and one ***DIV***).

  INT

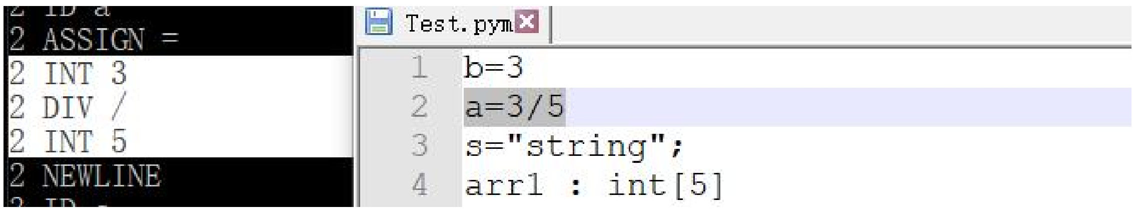


```
1 ID b
1 ASSIGN =
1 INT 3
1 NEWLINE
2 ID a
```

Test.pym

```
b=3
a=3/5
s="string";
```

  NUM



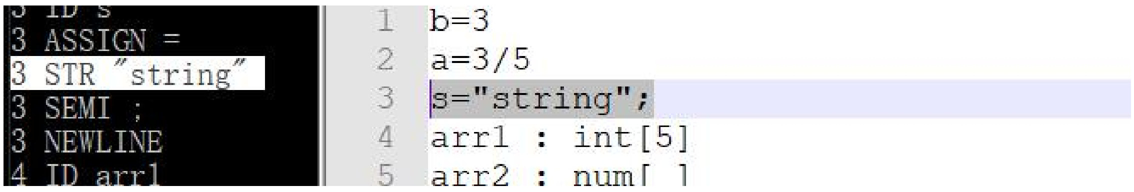
```
2 ID a
2 ASSIGN =
2 INT 3
2 DIV /
2 INT 5
2 NEWLINE
```

Test.pym

```
b=3
a=3/5
s="string";
arr1 : int[5]
```

- String

String is denoted as ***STR*** token, the content of the token will store the string value.

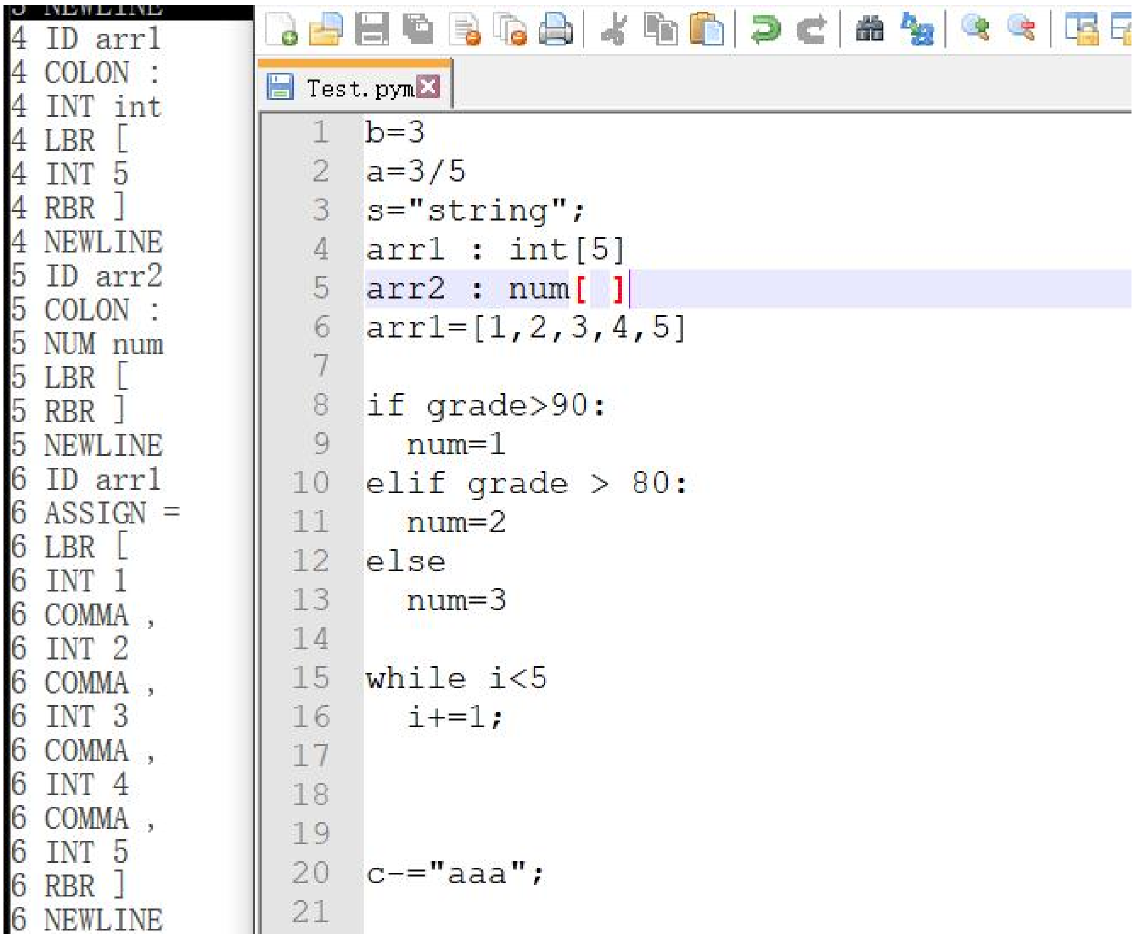


```
3 ID s
3 ASSIGN =
3 STR "string"
3 SEMI ;
3 NEWLINE
4 ID arr1
```

```
b=3
a=3/5
s="string";
arr1 : int[5]
arr2 : num[ ]
```

- Array

  Array is denoted as ***LBR*** (left bracket) and ***RBR*** (right bracket) with several elements separated by ***COMMA*** token (comma).



```
5 NEWLINE
4 ID arr1
4 COLON :
4 INT int
4 LBR [
4 INT 5
4 RBR ]
4 NEWLINE
5 ID arr2
5 COLON :
5 NUM num
5 LBR [
5 RBR ]
5 NEWLINE
6 ID arr1
6 ASSIGN =
6 LBR [
6 INT 1
6 COMMA ,
6 INT 2
6 COMMA ,
6 INT 3
6 COMMA ,
6 INT 4
6 COMMA ,
6 INT 5
6 RBR ]
6 NEWLINE
```

Test.pym

```
b=3
a=3/5
s="string";
arr1 : int[5]
arr2 : num[ ]
arr1=[1,2,3,4,5]

if grade>90:
   num=1
elif grade > 80:
   num=2
else
   num=3

while i<5
   i+=1;



c-="aaa";

```

- Address is not expressed in scanner.
- Variables

  Variables are declared by the following format (each character, keyword or identifier is a token).

```
a : int
B : num
arr1 : int[ ]
arr2 : int[ ]
arr3 : [ ]
```

- Function

  Built-in functions and user-defined functions are supported.

  For user-defined function, the grammar is :

```
def my_function1():
print("Hello from a function")
```

- Condition

  if statement and **if....elif...else** statement are supported.

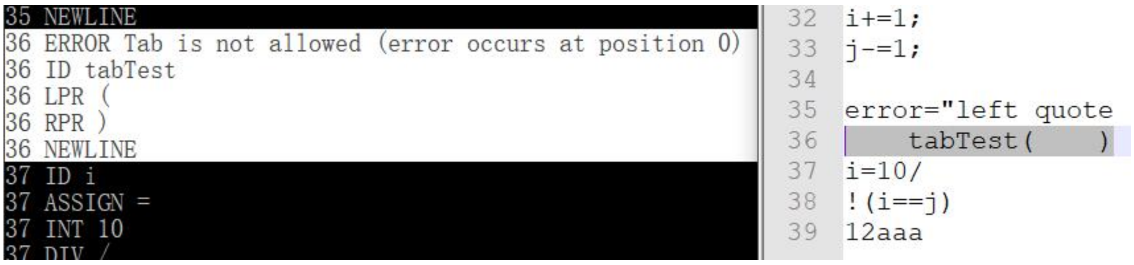
```
if grade>90:
  num=1
elif grade > 80:
  num=2
else
  num=3
```

- Loop

  Both **while** loop and **for** loop are supported by our programs.

```
while i<5
  i+=1;

for num in "123":
  i+=1
```

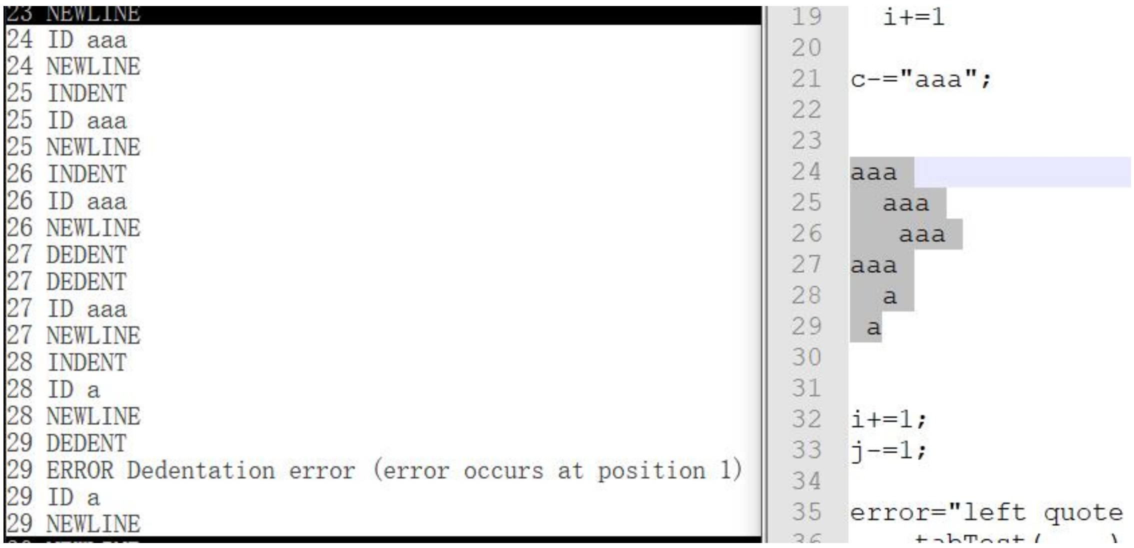
- Tab is treated as error in the leading space of a line, while it is treated as white space in other area.



```
35 NEWLINE
36 ERROR Tab is not allowed (error occurs at position 0)
36 ID tabTest
36 LPR (
36 RPR )
36 NEWLINE
37 ID i
37 ASSIGN =
37 INT 10
37 DIV /
```

```
32 i+=1;
33 j-=1;
34
35 error="left quote
36     tabTest(    )
37 i=10/
38 !(i==j)
39 12aaa
```

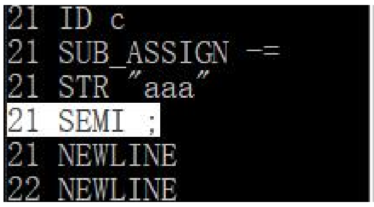
- ***INDENT*** and ***DEDENT*** token are strictly followed the indentation rules, otherwise the scanner will report errors.



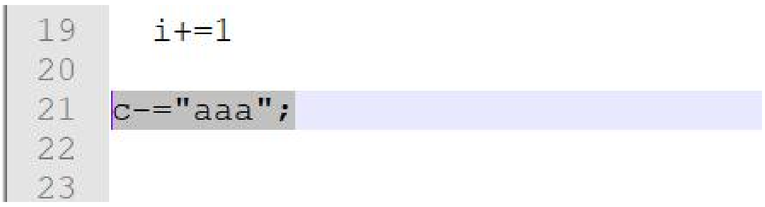
```
23 NEWLINE
24 ID aaa
24 NEWLINE
25 INDENT
25 ID aaa
25 NEWLINE
26 INDENT
26 ID aaa
26 NEWLINE
27 DEDENT
27 DEDENT
27 ID aaa
27 NEWLINE
28 INDENT
28 ID a
28 NEWLINE
29 DEDENT
29 ERROR Dedentation error (error occurs at position 1)
29 ID a
29 NEWLINE
```

```
19   i+=1
20
21 c-="aaa";
22
23
24 aaa
25   aaa
26     aaa
27 aaa
28   a
29  a
30
31
32 i+=1;
33 j-=1;
34
35 error="left quote
```

- Semicolon is allowed in scanner and separated as ***SEMI*** token.



```
21 ID c
21 SUB_ASSIGN -=
21 STR "aaa"
21 SEMI ;
21 NEWLINE
22 NEWLINE
```



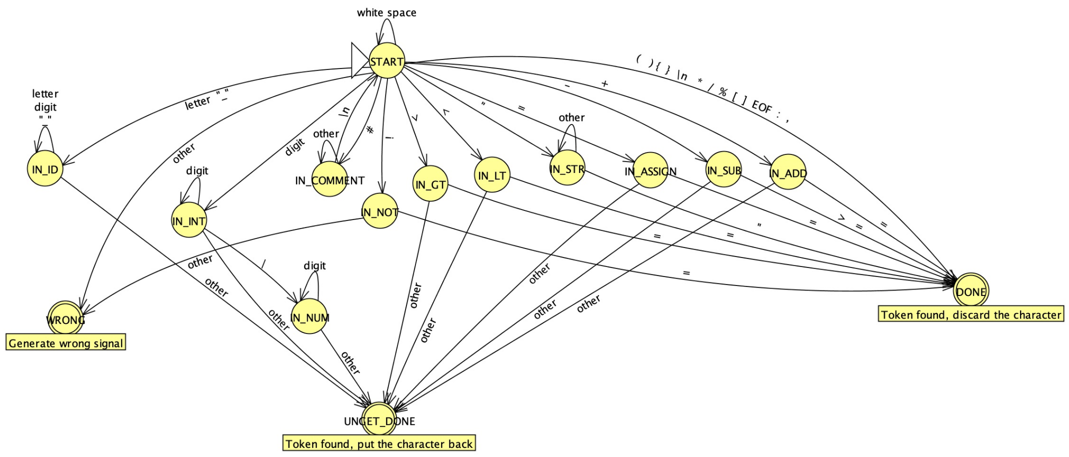
```
  i+=1

c-="aaa";


```

2. Difficulties and Solutions

- ### How to convert those strings to token one by one

  (a) Draw a DFA by JFLAP to increase the understanding of state transformation and cutting token:

  (b) Determine the type and content of token according to different states. When a token is found, return this token with its type and content, and then read afterwards.



- ### How to detect the the type of a number

  (a) When this token is in IN_INT state, when '\' occurs, it will go to IN_NUM state. IN_NUM state can only accept digit, just like IN_INT state.

  (b) It will occur problem 3.

```
else if (currentState == IN_INT) {
            if (isnum(ch)) {
                str += ch;
            }
            else if (ch == '\\') {
                str += ch;
                currentState = IN_NUM;
                numLoc = (*index)+1;
            }
            else {
                t.setType(INT);
                t.setContent(str);
                break;
            }
        }   //END OF IN_INT
```

- ## How to determine whether the token is *num* type or not, when a token is in IN_NUM state.

  (Look at our DFA diagram, it seems that digit ' \ ' can be seen as ***num*** type as well)

  (a) Record the first character after '\', detect the character. If this character is a digit, then it is a num. Otherwise, report an error.

```
else if (currentState == IN_NUM) {
    if (isnum(ch)) {
        str += ch;
    }
    else if (*index == numLoc) {    //the first character after '\'
        t.setType(ERROR);
        t.setContent(errorReport("This num is incomplete", start, numLoc-
1));
        break;
    }
    else {
        t.setType(NUM);
        t.setContent(str);
        break;
    }
}   //END OF IN_NUM
```

- ## How to implement the indentation algorithm

  Using a stack and carefully design.

```
//check indentation and dedentation, then generate token
Node* indentation(string line, int row, stack<int>& indentStack, Node*
lastNode, int* index) {
    bool tabFlag = false;
    //count the number of space and detect tab
    while (line[*index] == ' '|| line[*index]=='\t') {
        //if tab is used, report an error
        if (line[*index] == '\t') {
            lastNode = pushIntoList(lastNode, new Node(row, Token(ERROR,
errorReport("Tab is not allowed ", (*index)-1))));
            tabFlag = true;
        }
        (*index)++;
    }
    //empty line or tab is appeared, do not generate indent or dedent token
    if (*index == (line.length() - 1)||tabFlag) {
        return lastNode;
    }
    //dedentation
    if (indentStack.top() > (*index)) {
        indentStack.pop();
        lastNode = pushIntoList(lastNode, new Node(row, Token(DEDENT)));
        while (indentStack.top() > (*index)) {
            indentStack.pop();
            lastNode = pushIntoList(lastNode, new Node(row,
Token(DEDENT)));}
```

```
        //if the dedentation does not comply the previous indentation, omit it and generate an error token.
        if (indentStack.top() != (*index)){
            lastNode = pushIntoList(lastNode, new Node(row, Token(ERROR, errorReport("Dedentation error ",(*index)-1))));
            return lastNode;}
    }
    //indentation
    else if (indentStack.top() < (*index)) {
        lastNode=pushIntoList(lastNode, new Node(row, Token(INDENT)));
        indentStack.push(*index);
    }

    return lastNode;
}
```

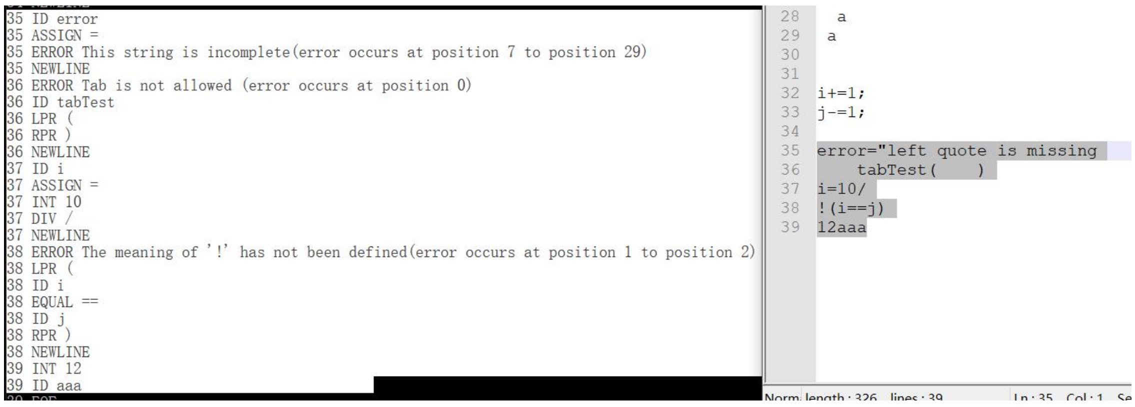
- ### How to implement the DFA algorithm

  We scan single line for a time.

```
//scan single line and generate token
Node* scan(string line, int row, Node* lastNode, int* index) {
    while ((*index) < line.length()) {
        Token t = dfa(line, index);
        lastNode=pushIntoList(lastNode, new Node(row, t));
    }
    return lastNode;
}
```

3. Bonus features

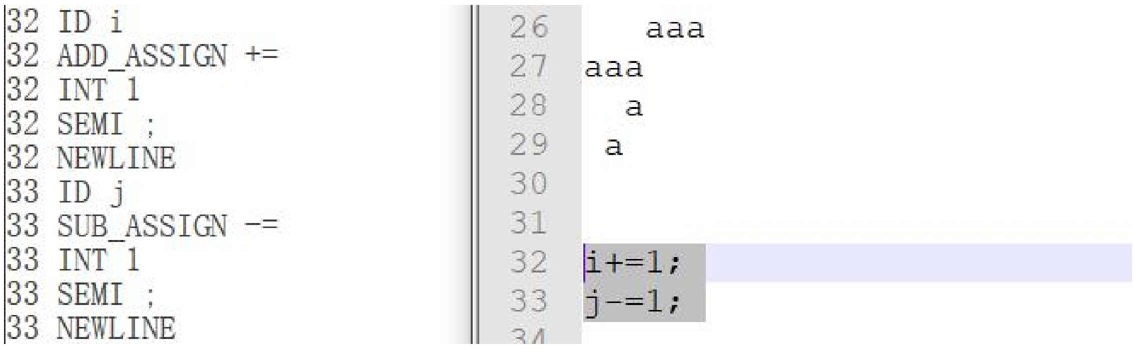
1. Scanner can detect all the errors that is defined in our scanner, and list them in the output of the scanner.
2. Scanner can also report where the errors occur and where the error ends.



```
35 ID error
35 ASSIGN =
35 ERROR This string is incomplete(error occurs at position 7 to position 29)
35 NEWLINE
36 ERROR Tab is not allowed (error occurs at position 0)
36 ID tabTest
36 LPR (
36 RPR )
36 NEWLINE
37 ID i
37 ASSIGN =
37 INT 10
37 DIV /
37 NEWLINE
38 ERROR The meaning of '!' has not been defined(error occurs at position 1 to position 2)
38 LPR (
38 ID i
38 EQUAL ==
38 ID j
38 RPR )
38 NEWLINE
39 INT 12
39 ID aaa
```

```
  a
 a


i+=1;
j-=1;

error="left quote is missing
    tabTest(     )
i=10/
!(i==j)
12aaa
```

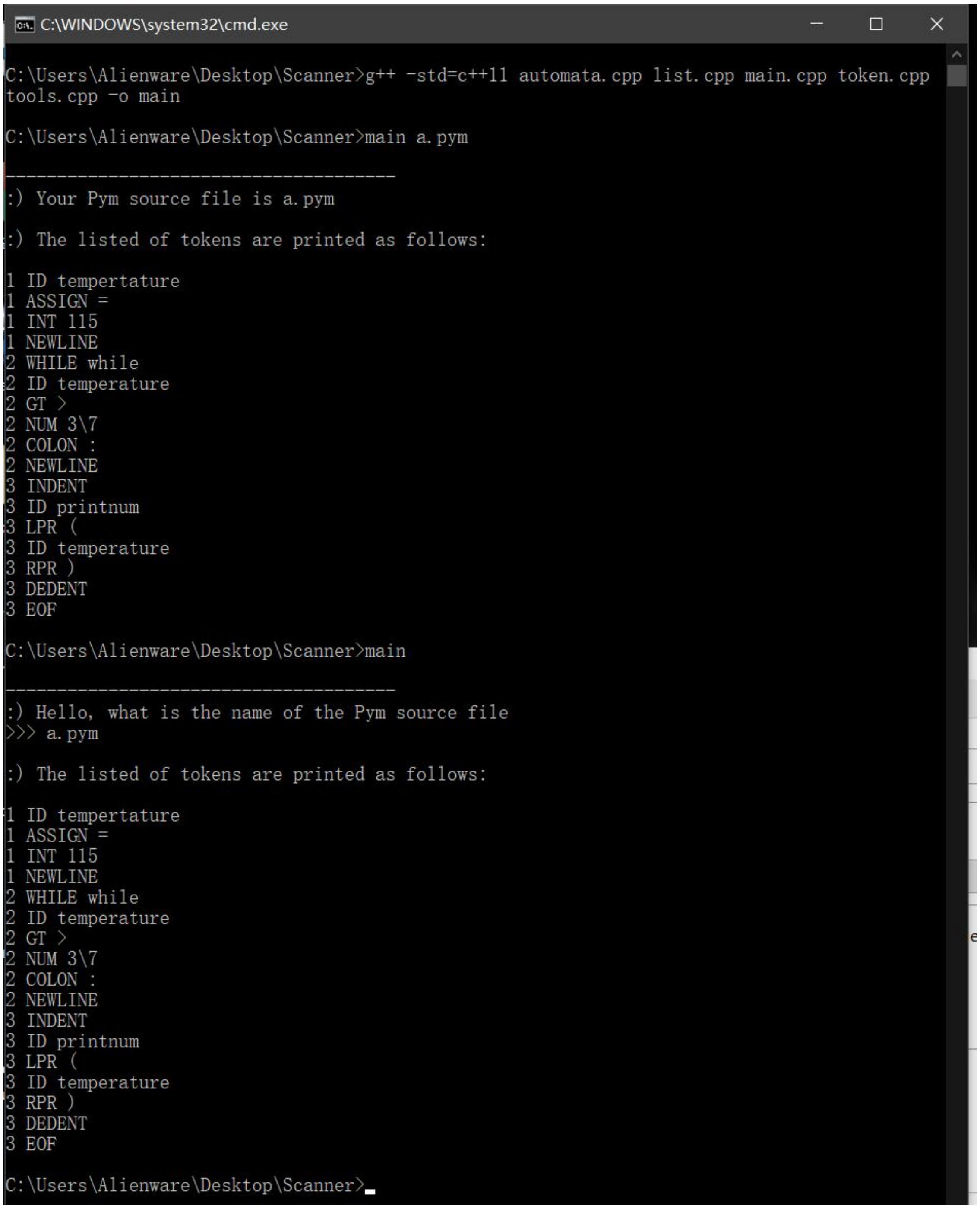
3. We added operators -= and += in our Pym scanner.



```
32 ID i
32 ADD_ASSIGN +=
32 INT 1
32 SEMI ;
32 NEWLINE
33 ID j
33 SUB_ASSIGN -=
33 INT 1
33 SEMI ;
33 NEWLINE
```

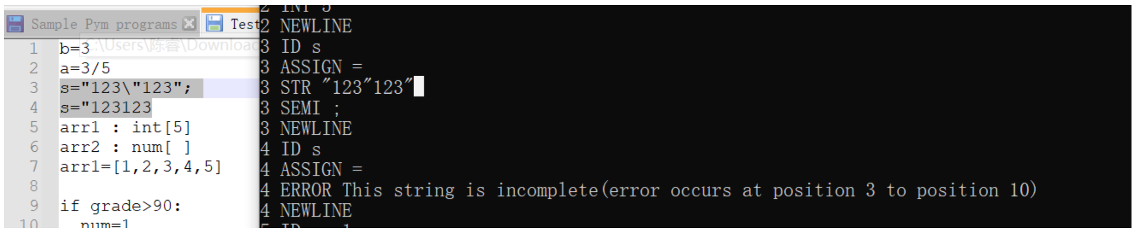
```
   aaa
aaa
  a
 a


i+=1;
j-=1;
```

4. Let program accept the source file name both as command-line argument and in-program input.



```
C:\Users\Alienware\Desktop\Scanner>g++ -std=c++11 automata.cpp list.cpp main.cpp token.cpp
tools.cpp -o main

C:\Users\Alienware\Desktop\Scanner>main a.pym

--------------------------------------
:) Your Pym source file is a.pym

:) The listed of tokens are printed as follows:

1 ID tempertature
1 ASSIGN =
1 INT 115
1 NEWLINE
2 WHILE while
2 ID temperature
2 GT >
2 NUM 3\7
2 COLON :
2 NEWLINE
3 INDENT
3 ID printnum
3 LPR (
3 ID temperature
3 RPR )
3 DEDENT
3 EOF

C:\Users\Alienware\Desktop\Scanner>main

--------------------------------------
:) Hello, what is the name of the Pym source file
>>> a.pym

:) The listed of tokens are printed as follows:

1 ID tempertature
1 ASSIGN =
1 INT 115
1 NEWLINE
2 WHILE while
2 ID temperature
2 GT >
2 NUM 3\7
2 COLON :
2 NEWLINE
3 INDENT
3 ID printnum
3 LPR (
3 ID temperature
3 RPR )
3 DEDENT
3 EOF

C:\Users\Alienware\Desktop\Scanner>
```

5. When the program encounter escape character, it will escape the next character, e.g. in a string "123\"123" , instead of reporting an error, our scanner will tranfer this string to "123"123",



```
b=3
a=3/5
s="123\"123";
s="123123
arr1 : int[5]
arr2 : num[ ]
arr1=[1,2,3,4,5]

if grade>90:
  num=1
```

```
2 NEWLINE
3 ID s
3 ASSIGN =
3 STR "123"123"
3 SEMI ;
3 NEWLINE
4 ID s
4 ASSIGN =
4 ERROR This string is incomplete(error occurs at position 3 to position 10)
4 NEWLINE
```

4. Division of Labor

- Chen, Rui
    1. Design the structure of the scanner, including classes, class member functions and tool functions.
    2. Implement the indentation algorithm and the body of the program.
    3. Design the output format and the error report format.
    4. Let program accept the source file name both as command-line argument and in-program input.
    5. Organize and merge this document file, of which the information is provided by three of us.
- Li, YiChu
    1. Design the DFA and draw the DFA diagram with JFLAP
    2. Implement the codes in dfa(), which separates characters into tokens.
    3. Add the bonus token and error token.
- Wang, HongBo
    1. Name all the token and list them as a table.
    2. Test the program and find out bugs.

5. Appendix

- Reserved Keywords

  int, num, str, if, else, elif

  while, for, def, return, and

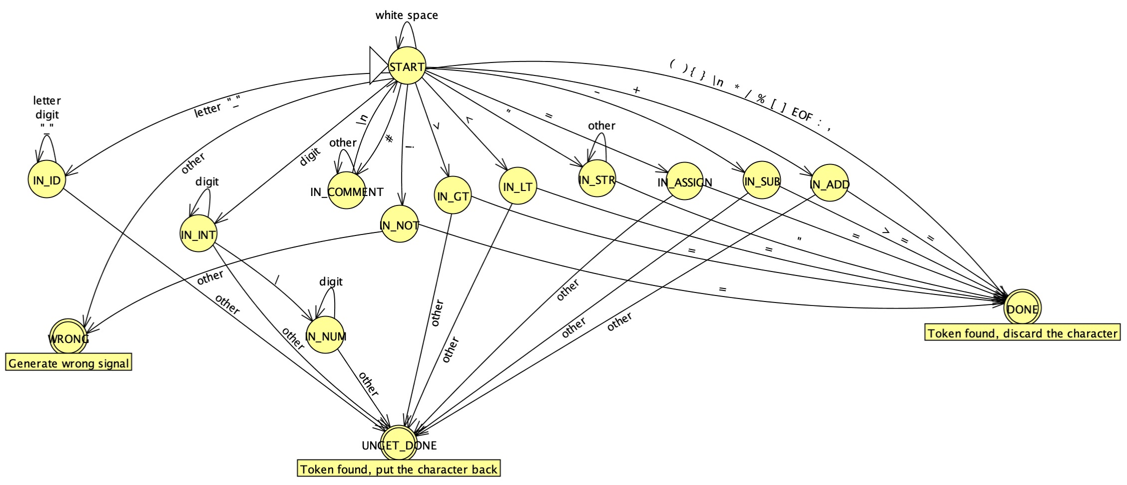
  or, not, in

- Token Name List

token	Full name	simplify
EOF	End Of File	EOF
+	addition	ADD
-	subtraction	SUB
*	multiplication	MUL
/	division	DIV
%	mod	MOD
<	less than	LT
<=	less than or equal to	LTE
>	greater than	GT
>=	greater than or equal to	GTE
=	assign	ASSIGN
==	equal	EQUAL
!=	not equal	NOT_EQUAL
,	comma	COMMA
(	left parentheses	LPR
)	right parentheses	RPR
[	left bracket	LBR
]	right bracket	RBR
{	left curly brace	LCUR
}	right curly brace	RCUR
(space)	DEDENT/INDENT	DEDENT/INDENT
(\n)	NEWLINE	NEWLINE
;	semicolon	SEMI
:	colon	COLON
-=	SUB_ASSIGN	SUB_ASSIGN
+=	ADD_ASSIGN	ADD_ASSIGN
->	return type	RETURN_TYPE
if	if	IF
else	else	ELSE
elif	else if	ELIF

token	Full name	simplify
while	while	WHILE
def	define	DEF
return	return	RETURN
and	and	AND
or	or	OR
not	not	NOT
int	integer	INT
num	number	NUM
str	string	STR
id	identifier	ID
in	keyword in	IN

- DFA Diagram



white space
START
letter
digit
"_"
letter "_"
IN_ID
other
digit
IN_INT
\n
other
#
IN_COMMENT
!
IN_NOT
>
IN_GT
<
IN_LT
"
IN_STR
=
IN_ASSIGN
-
IN_SUB
+
IN_ADD
( ){ } \n * / % [ ] EOF : ,
other
/
IN_NUM
=
"
>
WRONG
Generate wrong signal
UNGET_DONE
Token found, put the character back
DONE
Token found, discard the character