Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

Факультет Прикладної математики
Кафедра спеціалізованих комп’ютерних систем

Лабораторна робота №1

з дисципліни “Основи проектування трансляторів”

“РОЗРОБКА ЛЕКСИЧНОГО АНАЛІЗАТОРА”

Варіант 1

Виконав:

студент групи КВ-72

Бербега В.О.

Перевірив:

_____

Київ 2020

Постановка задачі:

Розробити програму лексичного аналізатора (ЛА) для підмножини мови програмування SIGNAL.

Програма має забезпечувати наступне (якщо це передбачається граматикою варіанту):

- згортання ідентифікаторів;
- згортання ключових слів;
- згортання цілих десяткових констант;
- згортання дійсних десяткових констант;
- згортання строкових констант, якщо вони визначені в заданій мові;

Також у всіх варіантах необхідно забезпечити:

- видалення коментарів, заданих у вигляді (*<текст коментарю>*)

Для кодування лексем необхідно використовувати числові діапазони, вказані в Таблиці 1.

Таблиця 1. Діапазони кодування лексем

Вид лексеми	**Числовий діапазон**
Односимвольні роздільники та знаки операцій (: / ; + тощо)	0 – 255 (тобто коди ASCII)
Багатосимвольні роздільники (:= <= <= тощо)	301 – 400
Ключові слова (BEGIN, END, FOR тощо)	401 – 500
Константи	501 – 1000
Ідентифікатори	1001 – . . .

Входом ЛА має бути наступне:

- вихідна програма, написана підмножиною мови SIGNAL відповідно до варіанту;
- таблиця кодів ASCII з атрибутами для визначення токенів;
- таблиця багато символьних роздільників;
- таблиця ключових слів;
- таблиця констант, в яку, при необхідності, попередньо можуть бути занесені стандартні константи;
- таблиця ідентифікаторів, в яку, при необхідності, попередньо занесені наперед визначені ідентифікатори.

Виходом ЛА має бути наступним:

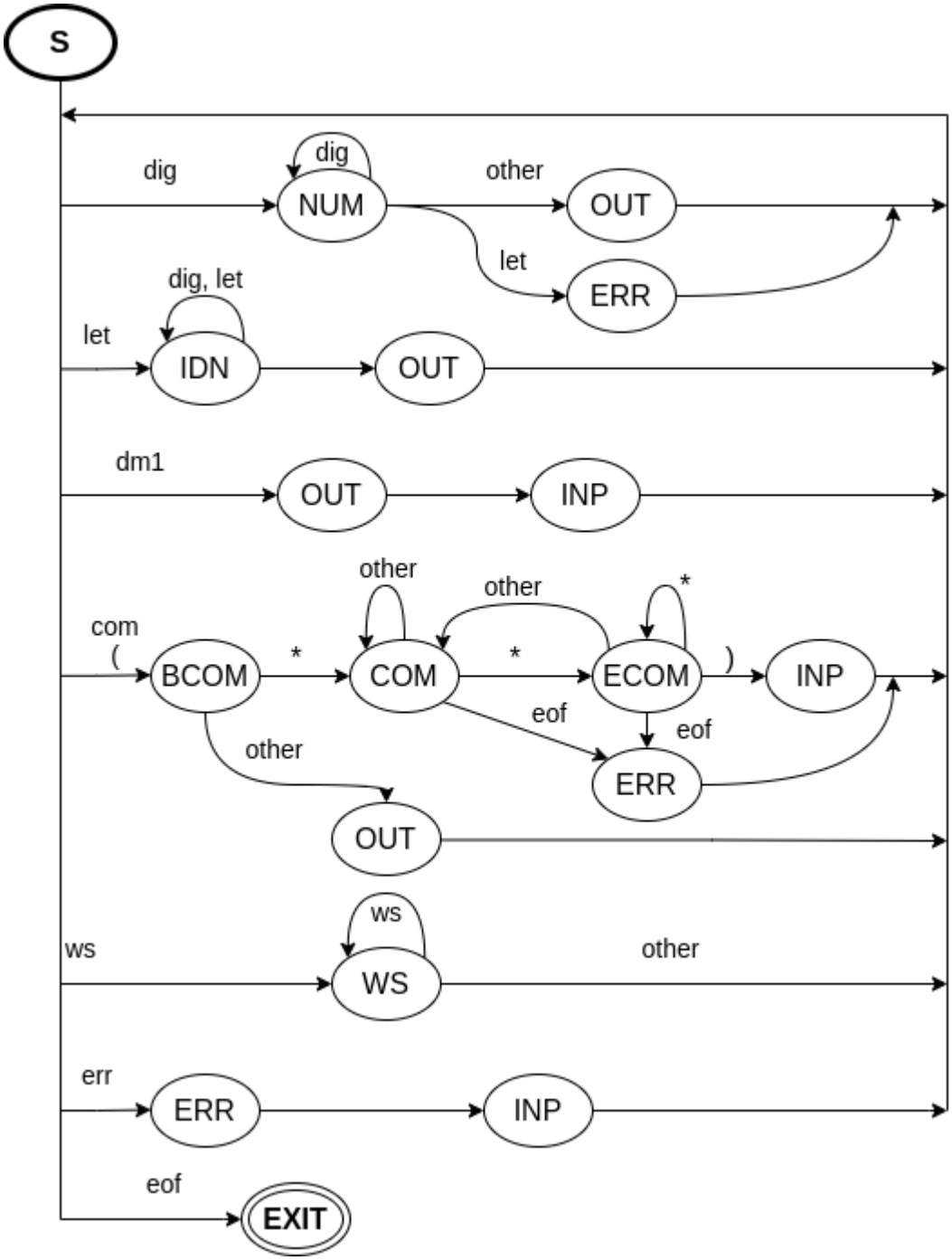
- закодований рядок лексем з інформацією про їх розташування у вихідній програмі (номер рядка, номер колонки);
- таблиця констант, що сформована для конкретної програми і яка містить значення та тип констант;
- таблиця ідентифікаторів, що сформована для конкретної програми.

Індивідуальне завдання

Варіант 1

```
1. <signal-program> --> <program>
2. <program> --> PROGRAM <procedure-identifier> ;

              <block>. |
              PROCEDURE <procedure-identifier>
              <parameters-list> ; <block> ;

3. <block> --> <declarations> BEGIN <statements-list> END

4. <declarations> --> <label-declarations>

5. <label-declarations> --> LABEL <unsigned-integer> <labels-list>; | <empty>

6. <labels-list> --> , <unsigned-integer> <labels-list> | <empty>

7. <parameters-list> --> ( <declarations-list> ) | <empty>

8. <declarations-list> --> <empty>
9. <statements-list> --> <empty>
10. <procedure-identifier> --> <identifier>
11. <identifier> --> <letter><string>
12. <string> --> <letter><string> |<digit><string> |  <empty>

13. <unsigned-integer> --> <digit><digits-string>
14. <digits-string> --> <digit><digits-string> |<empty>



15. <digit> --> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |9

16. <letter> --> A | B | C | D | ... | Z
```

Граф автомату



Текст програми:

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>

using namespace std;


int keyword_search(string tmp);
int constant_search(int constant);
int constant_tab(int constant);
int identifier_search(string tmp);
int identifier_tab(string tmp);



struct Symbol
{
int attr;
char value;
} symbol;

struct Lexem
{
int code;
int row;
int column;
} lex;

struct Lexer_error
{
int type;
int row;
int column;
char sym;
} lex_error;

int _const = 501; // const begin value
int _identif = 1001; // identifier begin value
int attributes[256];

// vector<Lexem> lexem_table;
map<int, int> const_tab;
map<string, int> identif_tab;
map<string, int> keyword_tab = {{"PROGRAM", 401}, {"PROCEDURE", 402}, {"BEGIN", 403},
{"END", 404}, {"LABEL", 405}};
```

```
Symbol get_symbol(ifstream &fin)
{
Symbol temp_sym;
fin.get(temp_sym.value);
temp_sym.attr = attributes[int(temp_sym.value)];
return temp_sym;
}

int main()
{
int constant, _constant, keyword, identif;
int column_num = 1, row_num = 1;
int i, l = 0, k = 0;
string temp_symb = "";
lex_error.type = 0;
string temp;
string *arr_const = new string[256];
int *code_const = new int[256];
string *arr_ident = new string[256];
int *code_ident = new int[256];

for (i = 0; i <= 255; ++i) { attributes[i] = 6; }


attributes[32] = 0; // space symbol
for (i = 8; i <= 13; i++) { attributes[i] = 0; }
// digits
for (i = 48; i <= 57; i++) { attributes[i] = 1; }
// letters
for (i = 65; i <= 90; i++) { attributes[i] = 2; }


attributes[44] = 3; // ,
attributes[46] = 3; // .
attributes[58] = 3; // :
attributes[59] = 3; // ;


attributes[40] = 5; // (
attributes[41] = 3; // )

string test;
cout << "Enter folder (TrueTest(1 or 2) or FalseTest(1 or 2)): " << endl;
cin >> test;

ifstream fin("Test/" + test + "/input.sig");
ofstream fout1("Test/" + test + "/generated.txt");
ofstream fout2("Test/" + test + "/expected.txt");
if (fin.peek() == std::ifstream::traits_type::eof())
{
cout << "Empty file or no file" << endl;
return 0;
```

```
}

symbol = get_symbol(fin);
while (!fin.eof())
{
temp = "";
lex.code = 0;
switch (symbol.attr)
{
case 0: // space symbols
while ((!fin.eof()) && (symbol.attr == 0))
{
if (symbol.value == int('\n'))
{
row_num++;
column_num = 0;
}
symbol = get_symbol(fin);
column_num++;
}
break;

case 1: // digits
lex.row = row_num;
lex.column = column_num;
while ((!fin.eof()) && (symbol.attr == 1))
{
temp += symbol.value;
symbol = get_symbol(fin);
column_num++;
}
constant = stoi(temp);
_constant = constant_search(constant);
if (_constant != -1)
{
lex.code = _constant;
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
}
else
{
lex.code = constant_tab(constant);
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
arr_const[k] = temp;
code_const[k] = lex.code;
```

```
k += 1;
}
break;

case 2: // letters
lex.row = row_num;
lex.column = column_num;
while ((!fin.eof()) && ((symbol.attr == 2) || (symbol.attr == 1)))
{
temp += symbol.value;
symbol = get_symbol(fin);
column_num++;
}
keyword = keyword_search(temp);
if (keyword != -1)
{
lex.code = keyword;
if(temp.length() > 5)
{
cout << "|" << temp << "\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column << "
\t|\n";
fout1 << "|" << temp << "\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout2 << lex.code << " ";
}
else
{
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
}
}
else
{
identif = identifier_search(temp);
if (identif != -1)
{
lex.code = identif;
if(temp.length() > 2)
{
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
}
else
{
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
```

```
fout1 << "|" << temp << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
}
}
else
{
lex.code = identifier_tab(temp);
if(temp.length() > 2)
{
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
arr_ident[l] = temp;
code_ident[l] = lex.code;
l += 1;
}
else
{
cout << "|" << temp << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column <<
" \t|\n";
fout1 << "|" << temp << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " << lex.column
<< " \t|\n";
fout2 << lex.code << " ";
arr_ident[l] = temp;
code_ident[l] = lex.code;
l += 1;
}
}
}
break;

case 3: // single char delimiters
if(symbol.value == int('('))
{
temp_symb = '(';
symbol = get_symbol(fin);
lex_error.column = column_num;
if (symbol.value == int('*'))
{
if (fin.eof())
{
lex_error.type = 50;
lex_error.row = row_num;
}
else
{
symbol = get_symbol(fin);
column_num++;
while (symbol.value != int(')'))
```

```
{
while ((!fin.eof()) && (symbol.value != int('*')))
{
temp += symbol.value;
symbol = get_symbol(fin);
column_num++;
}
if (fin.eof())
{
lex_error.type = 50;
lex_error.row = row_num;
symbol.value = int('+');
break;
}
else
{
symbol = get_symbol(fin);
column_num++;
}
}

if (!fin.eof())
{
symbol = get_symbol(fin);
column_num++;
}
}
}
else
{
lex.row = row_num;
lex.column = column_num;
lex.code = attributes[int('(')];
cout << "|" << temp_symb << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
fout1 << "|" << temp_symb << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
column_num++;
}
break;
}
else
{
lex.row = row_num;
lex.column = column_num;
lex.code = attributes[symbol.value];
cout << "|" << symbol.value << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
fout1 << "|" << symbol.value << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
symbol = get_symbol(fin);
column_num++;
```

```
break;
}

case 4:
cout << "Case 4" << endl;
break;

case 5:
if (fin.eof())
{
lex.row = row_num;
lex.column = column_num;
lex.code = symbol.attr;
}
else
{
if(symbol.value == int('('))
{
temp_symb = '(';
symbol = get_symbol(fin);
lex_error.column = column_num;
if (symbol.value == int('*'))
{
if (fin.eof())
{
lex_error.type = 50;
lex_error.row = row_num;
}
else
{
symbol = get_symbol(fin);
column_num++;
while (symbol.value != int(')'))
{
while ((!fin.eof()) && (symbol.value != int('*')))
{
temp += symbol.value;
symbol = get_symbol(fin);
column_num++;
}
if (fin.eof())
{
lex_error.type = 50;
lex_error.row = row_num;
symbol.value = int('+');
break;
}
else
{
symbol = get_symbol(fin);
column_num++;
}
```

```
}

if (!fin.eof())
{
symbol = get_symbol(fin);
column_num++;
}
}
}
else
{
lex.row = row_num;
lex.column = column_num;
lex.code = attributes[int('(')];
cout << "|" << temp_symb << "\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
fout1 << "|" << temp_symb << "\t\t\t| " << lex.code << " \t| " << lex.row << " \t| " <<
lex.column << " \t|\n";
column_num++;
}
break;
}
}
break;

case 6:
lex_error.row = row_num;
lex_error.column = column_num;
lex_error.sym = symbol.value;
lex_error.type = 60;
}
if (lex_error.type != 0)
{
cout << "Lexer Error: ";
fout1 << "Lexer Error: ";
if (lex_error.type == 50)
{
cout << "unclosed comment, " << "row " << lex_error.row << ", column " <<
lex_error.column << endl;
fout1 << "unclosed comment, " << "row " << lex_error.row << ", column " <<
lex_error.column << endl;
break;
}
else
{
if (lex_error.type == 60)
{
cout << "Illegal symbol: " << lex_error.sym << ", " << "row " << lex_error.row << ", column "
<< lex_error.column << endl;
fout1 << "Illegal symbol: " << lex_error.sym << ", " << "row " << lex_error.row << ", column
" << lex_error.column << endl;
break;
```

```
}
}
}
}


cout << endl << "Constant table:" << endl;
fout1 << endl << "Constant table:" << endl;
for(int i = 0; i < k; i++)
{
cout << arr_const[i] << " --- " << code_const[i] << endl;
fout1 << arr_const[i] << " --- " << code_const[i] << endl;
}
cout << endl << "Identifier table:" << endl;
fout1 << endl << "Identifier table:" << endl;
for(int i = 0; i < l; i++)
{
cout << arr_ident[i] << " --- " << code_ident[i] << endl;
fout1 << arr_ident[i] << " --- " << code_ident[i] << endl;
}
delete[] arr_const;
delete[] arr_ident;
delete[] code_const;
delete[] code_ident;
fin.close();
fout1.close();
fout2.close();
cout << endl;
return 0;
}


int keyword_search(string tmp)
{
map<string, int>::iterator it;
it = keyword_tab.find(tmp);
if (it != keyword_tab.end())
{
return it->second;
}
else
{
return -1;
}
}

int identifier_tab(string tmp)
{
int value = identif_tab.size() + _identif;
identif_tab.insert(make_pair(tmp, value));
return value;
}
```

```
int identifier_search(string tmp)
{
map<string, int>::iterator it;
it = identif_tab.find(tmp);
if (it != identif_tab.end())
{
return it->second;
}
else
{
return -1;
}
}

int constant_tab(int constant)
{
int value = const_tab.size() + _const;
const_tab.insert(make_pair(constant, value));
return value;
}

int constant_search(int constant)
{
map<int, int>::iterator it;
it = const_tab.find(constant);
if (it != const_tab.end())
{
return it->second;
}
else
{
return -1;
}
}
```

Тести

Truetest1

```
PROGRAM SCS;
    PROCEDURE NAVI;
    A1;
    BEGIN
        A2, B2;
    END;
    LABEL 12, 14;
    (JW, KRIMZ)
```

```
vavox@vavox-Lenovo-Y50-70:~/Univ/OPT/Lab1$ make run
g++ -c lexer.cpp
g++ lexer.o -o lexer
./lexer
Enter folder (TrueTest(1 or 2) or FalseTest(1 or 2)):
TrueTest1
|PROGRAM     | 401  | 1    | 1    |
|SCS         | 1001 | 1    | 9    |
|;           | 3    | 1    | 12   |
|PROCEDURE   | 402  | 2    | 5    |
|NAVI        | 1002 | 2    | 15   |
|;           | 3    | 2    | 19   |
|A1          | 1003 | 3    | 5    |
|;           | 3    | 3    | 7    |
|BEGIN       | 403  | 4    | 5    |
|A2          | 1004 | 5    | 9    |
|,           | 3    | 5    | 11   |
|B2          | 1005 | 5    | 13   |
|;           | 3    | 5    | 15   |
|END         | 404  | 6    | 5    |
|;           | 3    | 6    | 8    |
|LABEL       | 405  | 7    | 5    |
|12          | 501  | 7    | 11   |
|,           | 3    | 7    | 13   |
|14          | 502  | 7    | 15   |
|;           | 3    | 7    | 17   |
|(           | 5    | 8    | 5    |
|JW          | 1006 | 8    | 6    |
|,           | 3    | 8    | 8    |
|KRIMZ       | 1007 | 8    | 10   |
|)           | 3    | 8    | 15   |

Constant table:
12 --- 501
14 --- 502

Identifier table:
SCS --- 1001
NAVI --- 1002
A1 --- 1003
A2 --- 1004
B2 --- 1005
JW --- 1006
KRIMZ --- 1007
```

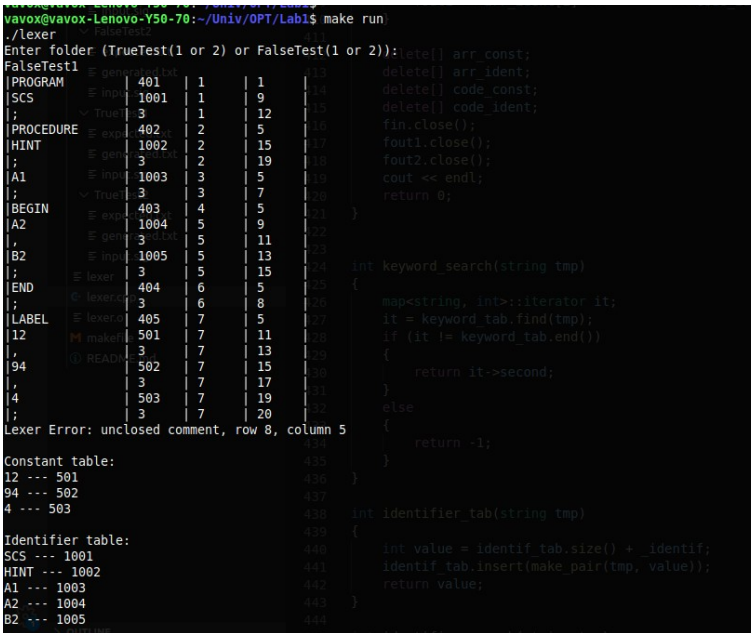
Truetest2

```
PROGRAM SCS;
    PROCEDURE HINT;
    A1;
    BEGIN(*BegiNN%$&*@#$$$#

    gghhjj

    *)
        A2, B2;
    END;
    LABEL 12, 14, 78;
```

```
vavox@vavox-Lenovo-Y50-70:~/Univ/OPT/Lab1$ make run
./lexer
Enter folder (TrueTest(1 or 2) or FalseTest(1 or 2)):
TrueTest2
|PROGRAM     | 401  | 1    | 1    |
|SCS         | 1001 | 1    | 9    |
|;           | 3    | 1    | 12   |
|PROCEDURE   | 402  | 2    | 5    |
|HINT        | 1002 | 2    | 15   |
|;           | 3    | 2    | 19   |
|A1          | 1003 | 3    | 5    |
|;           | 3    | 3    | 7    |
|BEGIN       | 403  | 4    | 5    |
|A2          | 1004 | 5    | 9    |
|,           | 3    | 5    | 11   |
|B2          | 1005 | 5    | 13   |
|;           | 3    | 5    | 15   |
|END         | 404  | 6    | 5    |
|;           | 3    | 6    | 8    |
|LABEL       | 405  | 7    | 5    |
|12          | 501  | 7    | 11   |
|,           | 3    | 7    | 13   |
|14          | 502  | 7    | 15   |
|,           | 3    | 7    | 17   |
|78          | 503  | 7    | 19   |
|;           | 3    | 7    | 21   |

Constant table:
12 --- 501
14 --- 502
78 --- 503

Identifier table:
SCS --- 1001
HINT --- 1002
A1 --- 1003
A2 --- 1004
B2 --- 1005
```

Fasle test1

```
PROGRAM SCS;
    PROCEDURE HINT;
    A1;
    BEGIN
        A2, B2;
    END;
    LABEL 12, 94, 4;
    (*JW, KRIMZ)
```



False test2

```
PROGRAM SCS;
    PROCEDURE HINT;
    A1;
    BEGIN
        A2, B2;
    END;
    LABEL %12, 44;

    (*JW, KRIMZ*)
```