Lab3 - Scanner Documentation

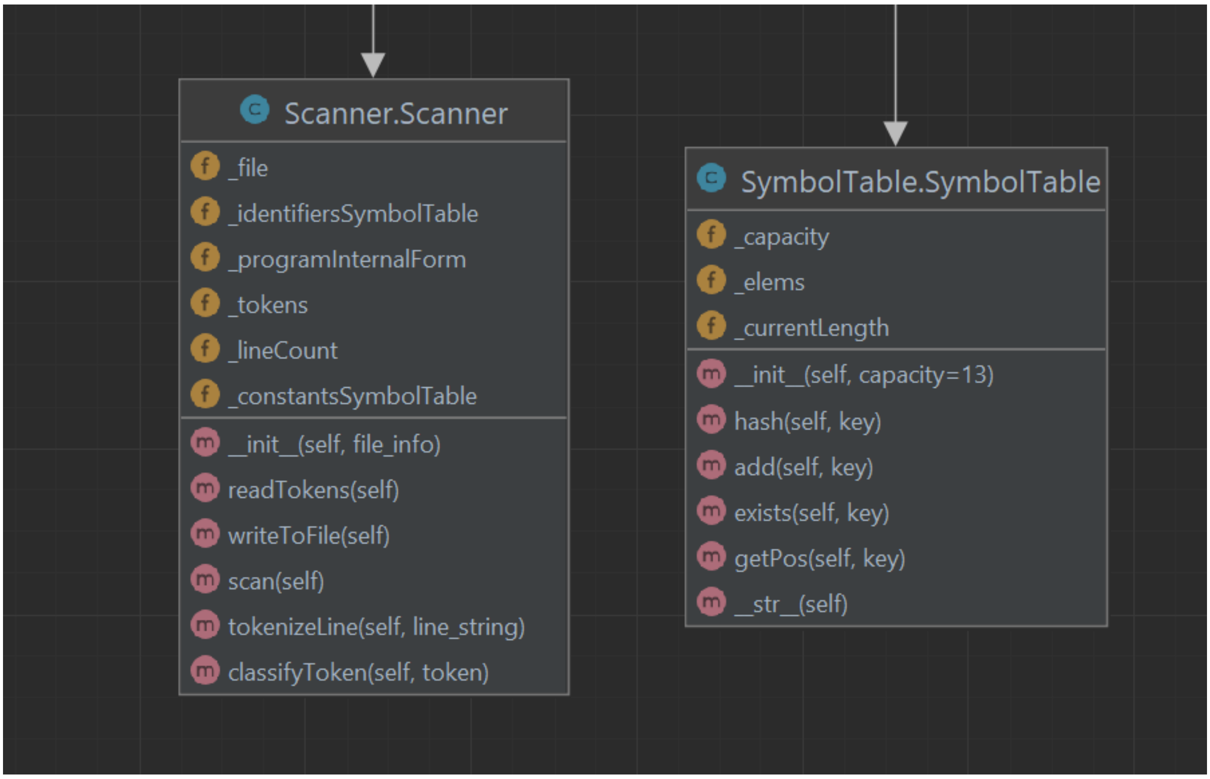
Github links

Previous lab: Symbol table → https://github.com/DiaconuAna/Formal-Languages-and-Compiler-Design/tree/main/Lab2 - Symbol Table

Current lab: Scanner → https://github.com/DiaconuAna/Formal-Languages-and-Compiler-Design/tree/main/Lab3 - Scanner

---

Class diagrams for Scanner and Symbol Table



Scanner class atributes

- `file`: instance of the file in which the toy language program is written
- `identifiersSymbolTable`: one of the two required symbol tables - for the identifiers
- `constantsSymbolTable`: one of the two required symbol tables - for the constants
- `programInternalForm`: a list of pairs (token/id/ct, number) corresponding to the program internal form

- `tokens`: list of program tokens read from tokens.in
- `lineCount`: counter to keep track of the line we are currently scanning in the program

Methods

▼ `readTokens()`

in: -

out: -

preconditions: the file `tokens.in` exists and contains the toy language's corresponding tokens

postconditions: the `tokens` attribute of the class is populated with the tokens read from the file

▼ `writeToFile`

in: -

out: -

preconditions: class attributes needed are `programInternalForm` and the two symbol tables (identifiers and constants)

postconditions: every pair from `programInternalForm` is written on a separate line in `PIF.out` + same for the elements stored in the symbol table

▼ `scan`

The scanning algorithm for the lecture is implemented here, with a little tweak for the two symbol tables. The text corresponding to the toy language is split by lines, each line being split in tokens using the `tokenizeLine` method. We try to classify each element as a reserved word/ keyword, an identifier or a constant. If it cannot be classified into one of these categories, the program ends in a lexical error specifying the line and the said token. Otherwise, program tokens are directly added to the PIF as a pair of the form `(token, 0)`, identifiers are added into the `identifiersSymbolTable` as pairs of the form `(id, (bucket_index, position_in_bucket))` and for constants we have `(ct, (bucket_index, position_in_bucket))`

in: -

out: -

preconditions: The file instance points to an existing file containing the text of the toy language.

postconditions: `writeToFile()` function is called with the final versions of the `programInternalForm`, `identifiersSymbolTable1` and `constantsSymbolTable`

▼ `tokenizeLine(line_string)`

In the `tokenizeLine` method we split the tokens from each line after removing whitespaces, tabs and newlines. We perform the look-ahead here for complex tokens corresponding to our toy language, such as `end_if`, `<-, >=, <=`. We also look for a potential lexical error as == is not considered a token in our program.

in: `line_string` - the string representing a line of the toy language program we read from the file

out: array of each element from the file corresponding to a potential token (program token, identifier or constant)

preconditions: `lineCount` attribute is initialized beforehand

postconditions: array of tokens from the line

▼ `classifyToken(token)`

in: token - string representing a token which is not a reserved word

out:

- 0 - token cannot be classified as an identifier nor as a constant
- 1 - token is an identifier
- 2 - token is a string constant
- 3 - token is a char constant
- 4 - token is an integer constant

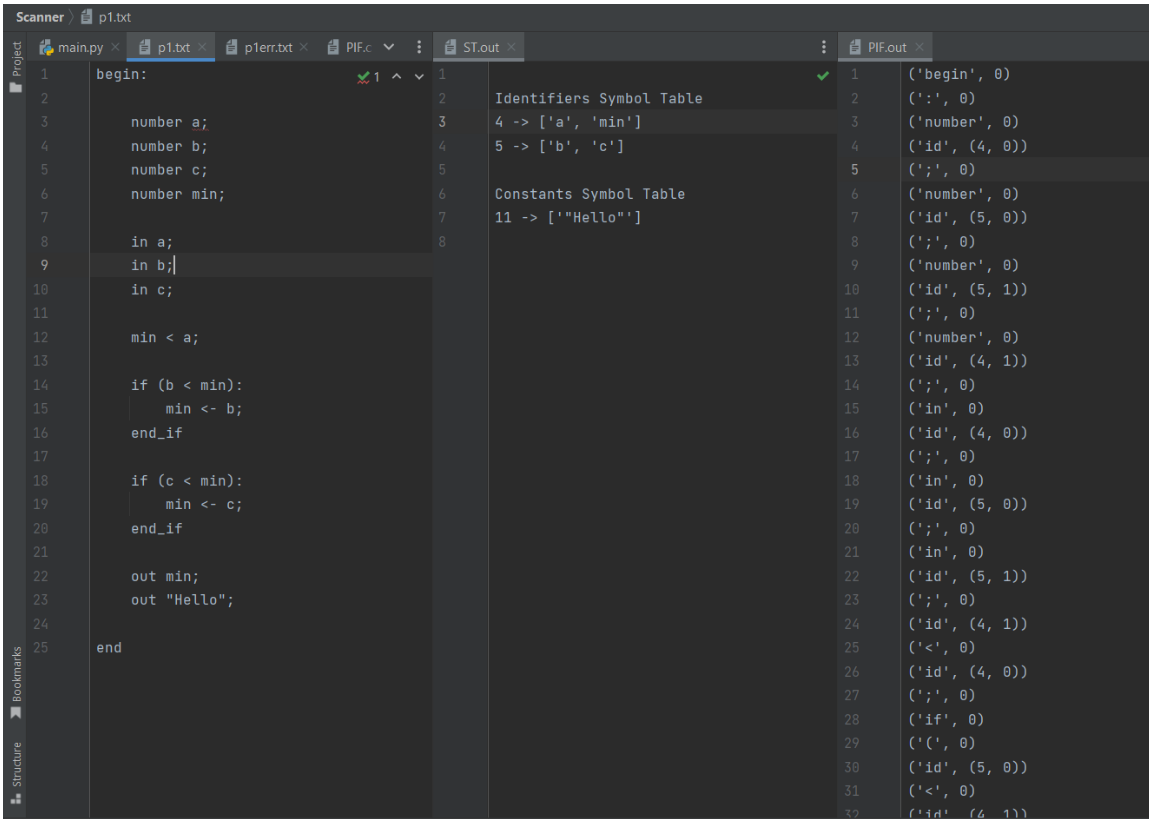
preconditions: token is a string

postconditions: token is classified using regular expression as an identifier, constant or none of the above

---

Examples

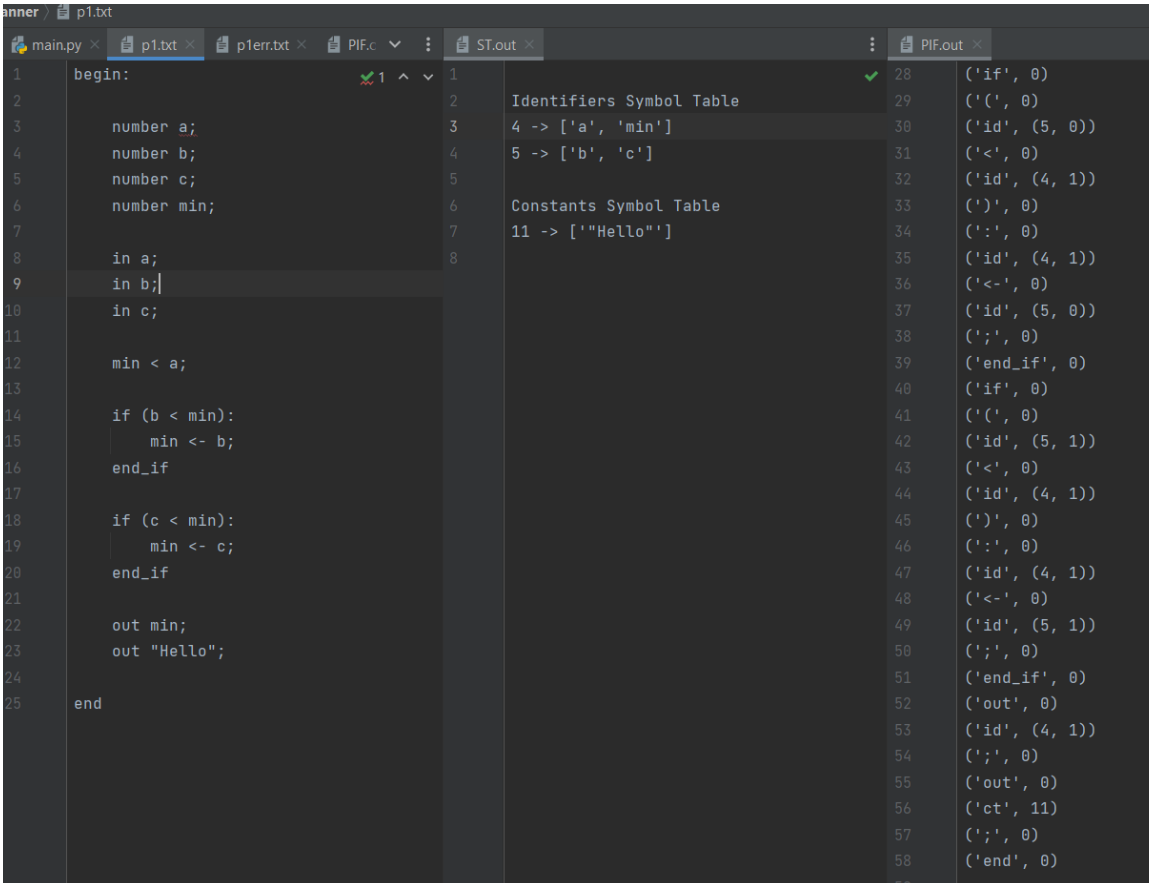
▼ p1.txt



```
begin:

    number a;
    number b;
    number c;
    number min;

    in a;
    in b;
    in c;

    min < a;

    if (b < min):
        min <- b;
    end_if

    if (c < min):
        min <- c;
    end_if

    out min;
    out "Hello";

end
```

```

Identifiers Symbol Table
4 -> ['a', 'min']
5 -> ['b', 'c']

Constants Symbol Table
11 -> ['"Hello"']

```

```
('begin', 0)
(':', 0)
('number', 0)
('id', (4, 0))
(';', 0)
('number', 0)
('id', (5, 0))
(';', 0)
('number', 0)
('id', (5, 1))
(';', 0)
('number', 0)
('id', (4, 1))
(';', 0)
('in', 0)
('id', (4, 0))
(';', 0)
('in', 0)
('id', (5, 0))
(';', 0)
('in', 0)
('id', (5, 1))
(';', 0)
('id', (4, 1))
('<', 0)
('id', (4, 0))
(';', 0)
('if', 0)
('(', 0)
('id', (5, 0))
('<', 0)
('id', (4, 1))
```

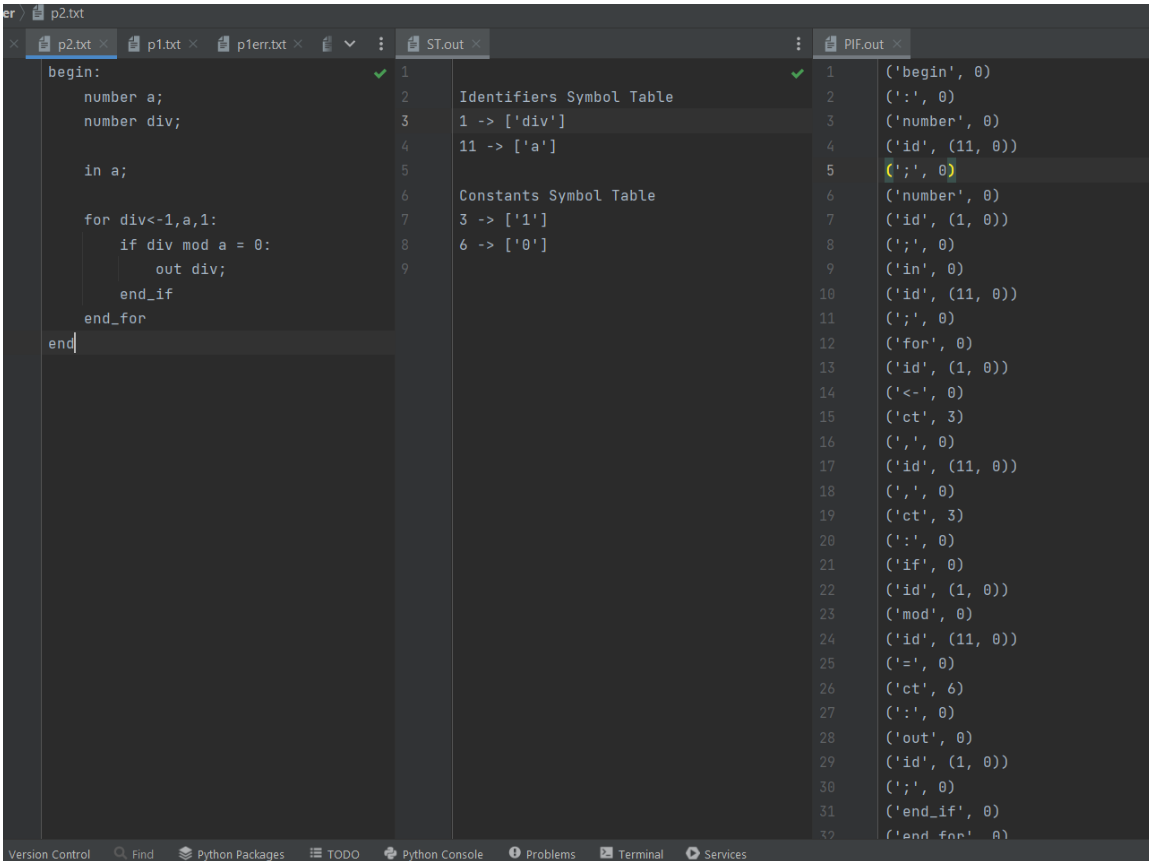


```
begin:

    number a;
    number b;
    number c;
    number min;

    in a;
    in b;
    in c;

    min < a;

    if (b < min):
        min <- b;
    end_if

    if (c < min):
        min <- c;
    end_if

    out min;
    out "Hello";

end
```

```

Identifiers Symbol Table
4 -> ['a', 'min']
5 -> ['b', 'c']

Constants Symbol Table
11 -> ['"Hello"']

```

```
('if', 0)
('(', 0)
('id', (5, 0))
('<', 0)
('id', (4, 1))
(')', 0)
(':', 0)
('id', (4, 1))
('<-', 0)
('id', (5, 0))
(';', 0)
('end_if', 0)
('if', 0)
('(', 0)
('id', (5, 1))
('<', 0)
('id', (4, 1))
(')', 0)
(':', 0)
('id', (4, 1))
('<-', 0)
('id', (5, 1))
(';', 0)
('end_if', 0)
('out', 0)
('id', (4, 1))
(';', 0)
('out', 0)
('ct', 11)
(';', 0)
('end', 0)
```

▼ p2.txt

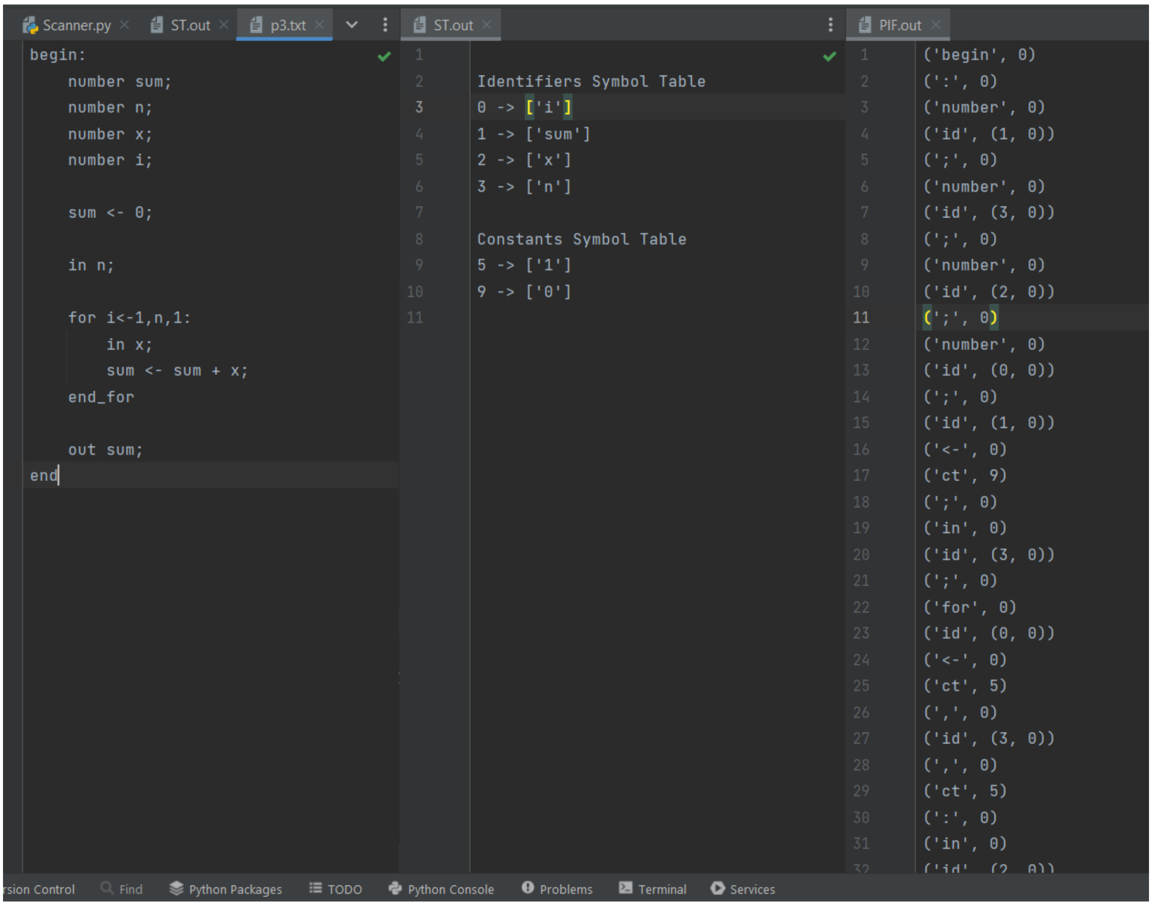


```
begin:
    number a;
    number div;

    in a;

    for div<-1,a,1:
        if div mod a = 0:
            out div;
        end_if
    end_for
end
```

```

Identifiers Symbol Table
1 -> ['div']
11 -> ['a']

Constants Symbol Table
3 -> ['1']
6 -> ['0']
```

```
('begin', 0)
(':', 0)
('number', 0)
('id', (11, 0))
(';', 0)
('number', 0)
('id', (1, 0))
(';', 0)
('in', 0)
('id', (11, 0))
(';', 0)
('for', 0)
('id', (1, 0))
('<-', 0)
('ct', 3)
(',', 0)
('id', (11, 0))
(',', 0)
('ct', 3)
(':', 0)
('if', 0)
('id', (1, 0))
('mod', 0)
('id', (11, 0))
('=', 0)
('ct', 6)
(':', 0)
('out', 0)
('id', (1, 0))
(';', 0)
('end_if', 0)
('end_for', 0)
```

▼ p3.txt

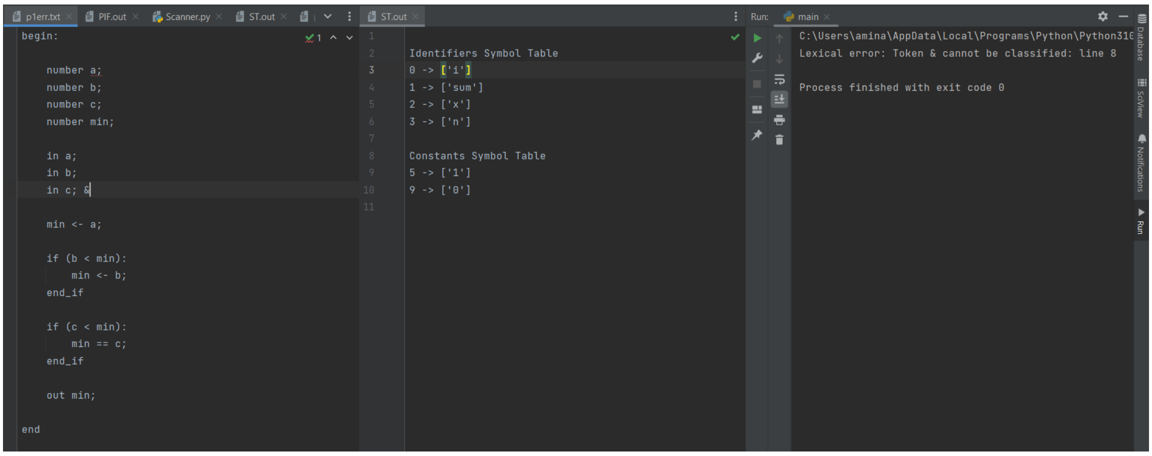


```
begin:
    number sum;
    number n;
    number x;
    number i;

    sum <- 0;

    in n;

    for i<-1,n,1:
        in x;
        sum <- sum + x;
    end_for

    out sum;
end
```

```

Identifiers Symbol Table
0 -> ['i']
1 -> ['sum']
2 -> ['x']
3 -> ['n']

Constants Symbol Table
5 -> ['1']
9 -> ['0']
```

```
('begin', 0)
(':', 0)
('number', 0)
('id', (1, 0))
(';', 0)
('number', 0)
('id', (3, 0))
(';', 0)
('number', 0)
('id', (2, 0))
(';', 0)
('number', 0)
('id', (0, 0))
(';', 0)
('id', (1, 0))
('<-', 0)
('ct', 9)
(';', 0)
('in', 0)
('id', (3, 0))
(';', 0)
('for', 0)
('id', (0, 0))
('<-', 0)
('ct', 5)
(',', 0)
('id', (3, 0))
(',', 0)
('ct', 5)
(':', 0)
('in', 0)
('id', (2, 0))
```

▼ p1_err.txt



```
begin:

    number a;
    number b;
    number c;
    number min;

    in a;
    in b;
    in c; &

    min <- a;

    if (b < min):
        min <- b;
    end_if

    if (c < min):
        min == c;
    end_if

    out min;

end
```

```

Identifiers Symbol Table
0 -> ['i']
1 -> ['sum']
2 -> ['x']
3 -> ['n']

Constants Symbol Table
5 -> ['1']
9 -> ['0']
```

```
C:\Users\amina\AppData\Local\Programs\Python\Python310
Lexical error: Token & cannot be classified: line 8

Process finished with exit code 0
```