

INE 5426

Trabalho 2: Analizador Sintático



A sua ConvCC-2023-1 possui recursão à esquerda?

A nossa ConvCC-2023-1 gerada não possui recursões à esquerd.
A gramatica completa sera mostrada mais a frente.

A sua ConvCC-2023-1 está fatorada à esquerda?

FUNCLIST → *FUNCDEF FUNCLIST* | *FUNCDEF*



```
3  FUNCLIST -> FUNCDEF FUNCLISTAUX
4
5  FUNCLISTAUX -> FUNCLIST | &
```

PARAMLISTCALL → (*ident*, *PARAMLISTCALL* | *ident*)?



PARAMLISTCALL -> *ident* , *PARAMLISTCALL* | *ident* | &



```
36  PARAMLISTCALL -> ident PARAMLISTCALLAUX | &
37
38  PARAMLISTCALLAUX -> , PARAMLISTCALL | &
```

A sua ConvCC-2023-1 está fatorada à esquerda?

PARAMLIST → ((*int* | *float* | *string*) *ident*, *PARAMLIST* |
(*int* | *float* | *string*) *ident*)?



PARAMLIST -> DATATYPE ident PARAMLIST | DATATYPE ident | &



PARAMLIST -> DATATYPE ident PARAMLISTAUX | &
PARAMLISTAUX -> , PARAMLIST | &

VARDECL → (*int* | *float* | *string*) *ident* ([*int_constant*])*



VARDECL -> DATATYPE ident OPT_VECTOR

OPT_VECTOR -> [*int_constant*] OPT_VECTOR | &

A sua ConvCC-2023-1 está fatorada à esquerda?

STATELIST → *STATEMENT* (*STATELIST*)?



`STATELIST -> STATEMENT STATELIST | STATEMENT | &`



`STATELIST -> STATEMENT OPT_STATELIST`

`OPT_STATELIST -> STATELIST | &`

LVALUE → *ident*([*NUMEXPRESSION*])*



`LVALUE -> ident OPT_ALLOC_NUMEXP`

`OPT_ALLOC_NUMEXP -> [NUMEXPRESSION] OPT_ALLOC_NUMEXP | &`

Retiramos recursões à esquerda e fatoramos...

Modificando um pouco a gramática original e colocando em LL (1), temos a seguinte gramática

A ConvCC-2023-1 modificada para LL(1):

PROGRAM -> STATEMENT | FUNCLIST | &

FUNCLIST -> FUNCDEF FUNCLISTAUX

FUNCLISTAUX -> FUNCLIST | &

FUNCDEF -> def ident (PARAMLIST) { STATELIST }

PARAMLIST -> DATATYPE ident PARAMLISTAUX | &

PARAMLISTAUX -> , PARAMLIST | &

DATATYPE -> int | float | string

**STATEMENT -> VARDECL ; | ATRIBSTAT ; | PRINTSTAT ; | READSTAT ; | RETURNSTAT ; |
IFSTAT | FORSTAT | { STATELIST } | break ; | ;**

VARDECL -> DATATYPE ident OPT_VECTOR

OPT_VECTOR -> [int_constant] OPT_VECTOR | &

ATRIBSTAT -> LVALUE = ATRIBSTAT_RIGHT

ATRIBSTAT_RIGHT -> FUNCCALL_OR_EXPRESSION | ALLOCEXPRESSION

**FUNCCALL_OR_EXPRESSION-> + FACTOR REC_UNARYEXPR REC_PLUS_MINUS_TERM
OPT_REL_OP_NUM_EXPR
| - FACTOR REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| int_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| float_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| string_constant REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| return_null REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| (NUMEXPRESSION) REC_UNARYEXPR REC_PLUS_MINUS_TERM OPT_REL_OP_NUM_EXPR
| ident FOLLOW_IDENT**

**FOLLOW_IDENT-> OPT_ALLOC_NUMEXP REC_UNARYEXPR REC_PLUS_MINUS_TERM
OPT_REL_OP_NUM_EXPR | (PARAMLISTCALL)**

PARAMLISTCALL -> ident PARAMLISTCALLAUX | &

PARAMLISTCALLAUX -> , PARAMLISTCALL | &

PRINTSTAT -> print EXPRESSION

READSTAT -> read LVALUE

RETURNSTAT -> return

IFSTAT -> if (EXPRESSION) { STATELIST } OPT_ELSE

OPT_ELSE -> else { STATELIST } | &

FORSTAT -> for (ATRIBSTAT ; EXPRESSION ; ATRIBSTAT) STATEMENT

STATELIST -> STATEMENT OPT_STATELIST

OPT_STATELIST -> STATELIST | &

ALLOCEXPRESSION -> new DATATYPE [NUMEXPRESSION] OPT_ALLOC_NUMEXP

OPT_ALLOC_NUMEXP -> [NUMEXPRESSION] OPT_ALLOC_NUMEXP | &

EXPRESSION -> NUMEXPRESSION OPT_REL_OP_NUM_EXPR

OPT_REL_OP_NUM_EXPR -> REL_OP NUMEXPRESSION | &

REL_OP -> < | > | <= | >= | == | /=

NUMEXPRESSION -> TERM REC_PLUS_MINUS_TERM

REC_PLUS_MINUS_TERM -> PLUS_OR_MINUS TERM REC_PLUS_MINUS_TERM | &

PLUS_OR_MINUS -> + | -

TERM -> UNARYEXPR REC_UNARYEXPR

REC_UNARYEXPR -> UNARYEXPR_OP TERM | &


UNARYEXPR_OP -> * | / | %

UNARYEXPR -> PLUS_OR_MINUS FACTOR | FACTOR

**FACTOR -> int_constant
| float_constant
| string_constant
| return_null
| LVALUE
| (NUMEXPRESSION)**

LVALUE -> ident OPT_ALLOC_NUMEXP

Para ConvCC-2023-1 estar em LL(1):

- Não deve ser recursiva à esquerda ;  Já mostramos que ela não é recursiva à esquerda
- A regra que deve ser escolhida ao desenvolver um não-terminal deve ser determinada por esse não-terminal e pelo (no máximo) próximo token na entrada.

Para demonstrar a segunda regra....

Precisamos satisfazer 3 condições

Condição A

Para as produções associadas a cada não-terminal x
 $x \rightarrow \alpha_1 pr(r_1) \mid \dots \mid \alpha_n pr(r_n)$
 $\text{First}(\alpha_i) \cap \text{First}(\alpha_j)$ é vazio sempre que $i \neq j$. Ou seja, os corpos das produções têm de ter primeiros conjuntos disjuntos.

Condição B

Para cada não-terminal x e o conjunto de produção associado ao não-terminal
 $x \rightarrow \alpha_1 pr(r_1) \mid \dots \mid \alpha_n pr(r_n)$
no máximo um α_i é anulável.

Condição C

Se x é um não-terminal anulável, então $\text{follow}(x)$ é disjunto de $\text{first}(x)$.

Condição A

Para as produções associadas a cada não-terminal x
 $x \rightarrow \alpha_1 pr(r_1) \mid \dots \mid \alpha_n pr(r_n)$
 $First(\alpha_i) \cap First(\alpha_j)$ é vazio sempre que $i \neq j$. Ou seja, os corpos das produções têm de ter primeiros conjuntos disjuntos.

Condição B

Para cada não-terminal x e o conjunto de produção associado ao não-terminal
 $x \rightarrow \alpha_1 pr(r_1) \mid \dots \mid \alpha_n pr(r_n)$
no máximo um α_i é anulável.

#	Expressão	Predição
1	PROGRAM \rightarrow STATEMENT	{, break, ;, int, float, string, print, return, for, ident, if, read
2	PROGRAM \rightarrow FUNCLIST	def
3	PROGRAM \rightarrow EPISILON	EPISILON
4	FUNCLIST \rightarrow FUNCDEF FUNCLISTAUX	def
5	FUNCLISTAUX \rightarrow FUNCLIST	def
6	FUNCLISTAUX \rightarrow EPISILON	EPISILON
7	FUNCDEF \rightarrow def ident { PARAMLIST } { STATELIST }	def
8	PARAMLIST \rightarrow DATATYPE ident PARAMLISTAUX	int, float, string
9	PARAMLIST \rightarrow EPISILON	EPISILON
10	PARAMLISTAUX \rightarrow PARAMLIST	

Condição C

Se x é um não-terminal anulável, então $\text{follow}(x)$ é disjunto de $\text{first}(x)$.

- PROGRAM;
- FUNCLISTAUX;
- PARAMLIST;
- PARAMLISTAUX;
- OPT_VECTOR;
- FOLLOW_IDENT;
- PARAMLISTCALL;
- OPT_ELSE;
- OPT_STATELIST;
- OPT_REL_OP_NUM_EXPR;
- REC_PLUS_MINUS_TERM;
- REC_UNARYEXPR

$\text{FIRST}(\text{PROGRAM}) = \{\text{int, float, string, ident, print, read, return, if, for, break, '{', '}', \text{def, \&}}\}$

$\text{FIRST}(\text{FUNCLISTAUX}) = \{\text{def, \&}\}$

$\text{FIRST}(\text{PARAMLIST}) = \{\text{int, float, string, \&}\}$

$\text{FIRST}(\text{PARAMLISTAUX}) = \{'', \&\}$

$\text{FIRST}(\text{OPT_VECTOR}) = \{'[', \&\}$

$\text{FOLLOW}(\text{PROGRAM}) = \{\$ \}$

$\text{FOLLOW}(\text{FUNCLISTAUX}) = \{\$ \}$

$\text{FOLLOW}(\text{PARAMLIST}) = \{')'\}$

$\text{FOLLOW}(\text{PARAMLISTAUX}) = \{')'\}$

$\text{FOLLOW}(\text{OPT_VECTOR}) = \{';'\}$

“

1. Não ser recursiva à esquerda
 2. Estar fatorada
 3. Para todo $A \in N$ tal que $A \xRightarrow{*} \varepsilon$, $\text{First}(A) \cap \text{Follow}(A) = \emptyset$
- .

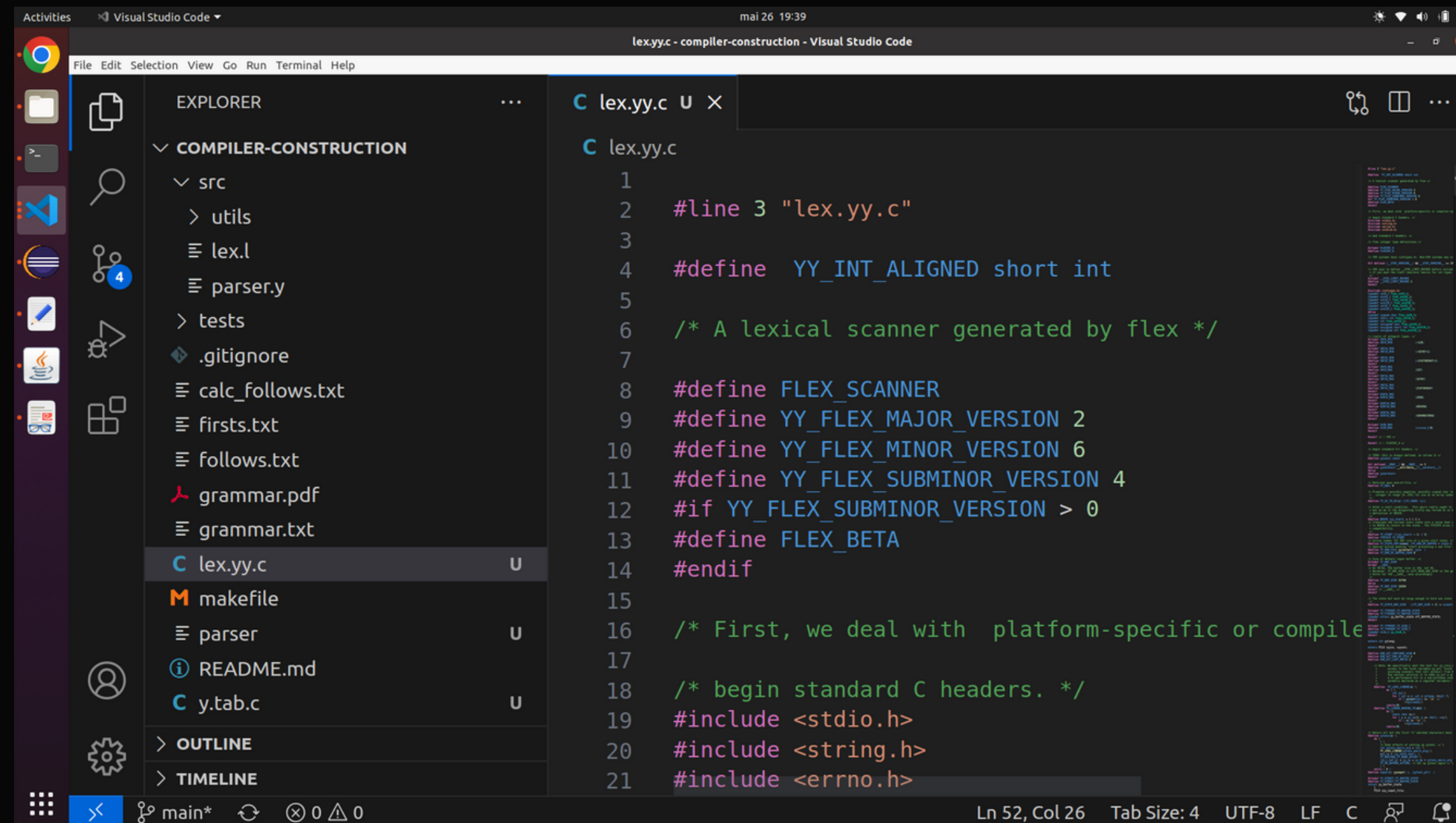
É LL (1)

Para desenvolver o trabalho utilizamos o YACC:

Yacc (Yet Another Compiler-Compiler) é uma ferramenta para construção de compiladores e interpretadores. Com base em uma especificação de gramática, ele gera um analisador sintático LALR(1) capaz de analisar várias linguagens. O analisador divide a entrada em uma árvore de análise sintática, usando automatos de pilha e tabelas de parsing. Sua saída é um arquivo em C/C++, que pode ser combinado com um analisador léxico (Lex/Flex) para criar um compilador completo. O Yacc simplifica o desenvolvimento de compiladores, fornecendo uma solução eficiente para análise sintática.

Qual é a entrada da ferramenta YACC?

- YACC recebe como entrada um arquivo gerado pelo LEX, que contém as regras de análise léxica:

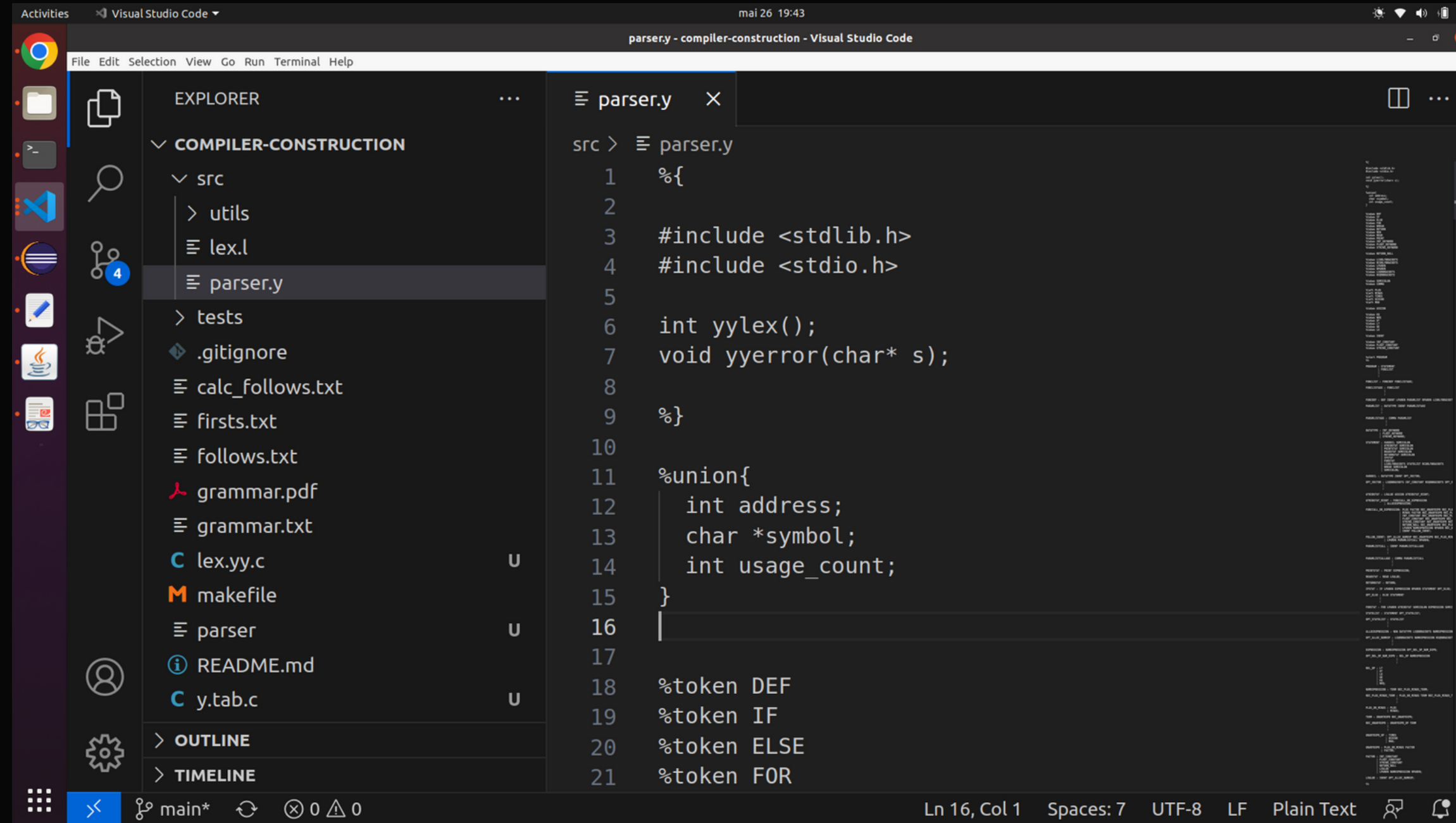


The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'COMPILER-CONSTRUCTION'. The file 'lex.yy.c' is selected and highlighted. The main editor area shows the content of 'lex.yy.c', which is a C file generated by flex. The code includes standard C headers and defines various macros for the lexical scanner.

```
lex.yy.c
1
2  #line 3 "lex.yy.c"
3
4  #define YY_INT_ALIGNED short int
5
6  /* A lexical scanner generated by flex */
7
8  #define FLEX_SCANNER
9  #define YY_FLEX_MAJOR_VERSION 2
10 #define YY_FLEX_MINOR_VERSION 6
11 #define YY_FLEX_SUBMINOR_VERSION 4
12 #if YY_FLEX_SUBMINOR_VERSION > 0
13 #define FLEX_BETA
14 #endif
15
16 /* First, we deal with platform-specific or compiler-specific issues */
17
18 /* begin standard C headers. */
19 #include <stdio.h>
20 #include <string.h>
21 #include <errno.h>
```

Qual é a entrada da ferramenta YACC?

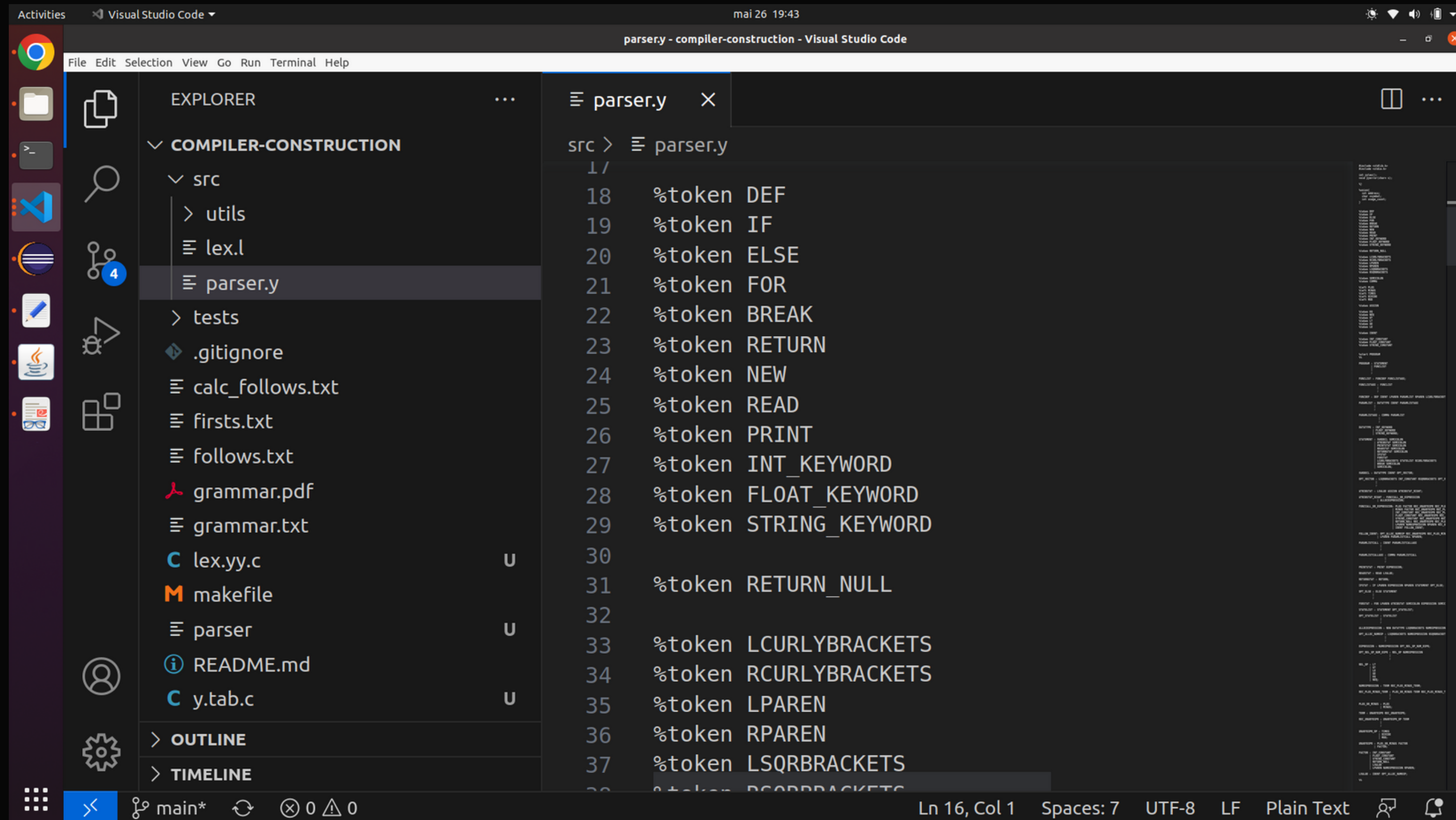
- YACC recebe como entrada um arquivo de especificação que descreve a gramática da linguagem a ser processada



The screenshot shows the Visual Studio Code interface. In the EXPLORER sidebar on the left, the file `parser.y` is selected under the `src` directory. The main editor window displays the content of `parser.y`, which is a YACC grammar specification. The code includes standard headers, function declarations, a union for non-terminals, and token definitions.

```
src > parser.y
1  %{
2
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  int yylex();
7  void yyerror(char* s);
8
9  %}
10
11 %union{
12     int address;
13     char *symbol;
14     int usage_count;
15 }
16
17
18 %token DEF
19 %token IF
20 %token ELSE
21 %token FOR
```

Qual é a entrada da ferramenta YACC?



The screenshot shows the Visual Studio Code interface with the 'COMPILER-CONSTRUCTION' project open. The Explorer sidebar on the left lists files in the 'src' directory, including 'parser.y'. The main editor window displays the content of 'parser.y', which is a YACC grammar file. The file defines tokens and non-terminals for a compiler construction project.

```
17
18 %token DEF
19 %token IF
20 %token ELSE
21 %token FOR
22 %token BREAK
23 %token RETURN
24 %token NEW
25 %token READ
26 %token PRINT
27 %token INT_KEYWORD
28 %token FLOAT_KEYWORD
29 %token STRING_KEYWORD
30
31 %token RETURN_NULL
32
33 %token LCURLYBRACKETS
34 %token RCURLYBRACKETS
35 %token LPAREN
36 %token RPAREN
37 %token LSQBRACKETS
38 %token RSQBRACKETS
```

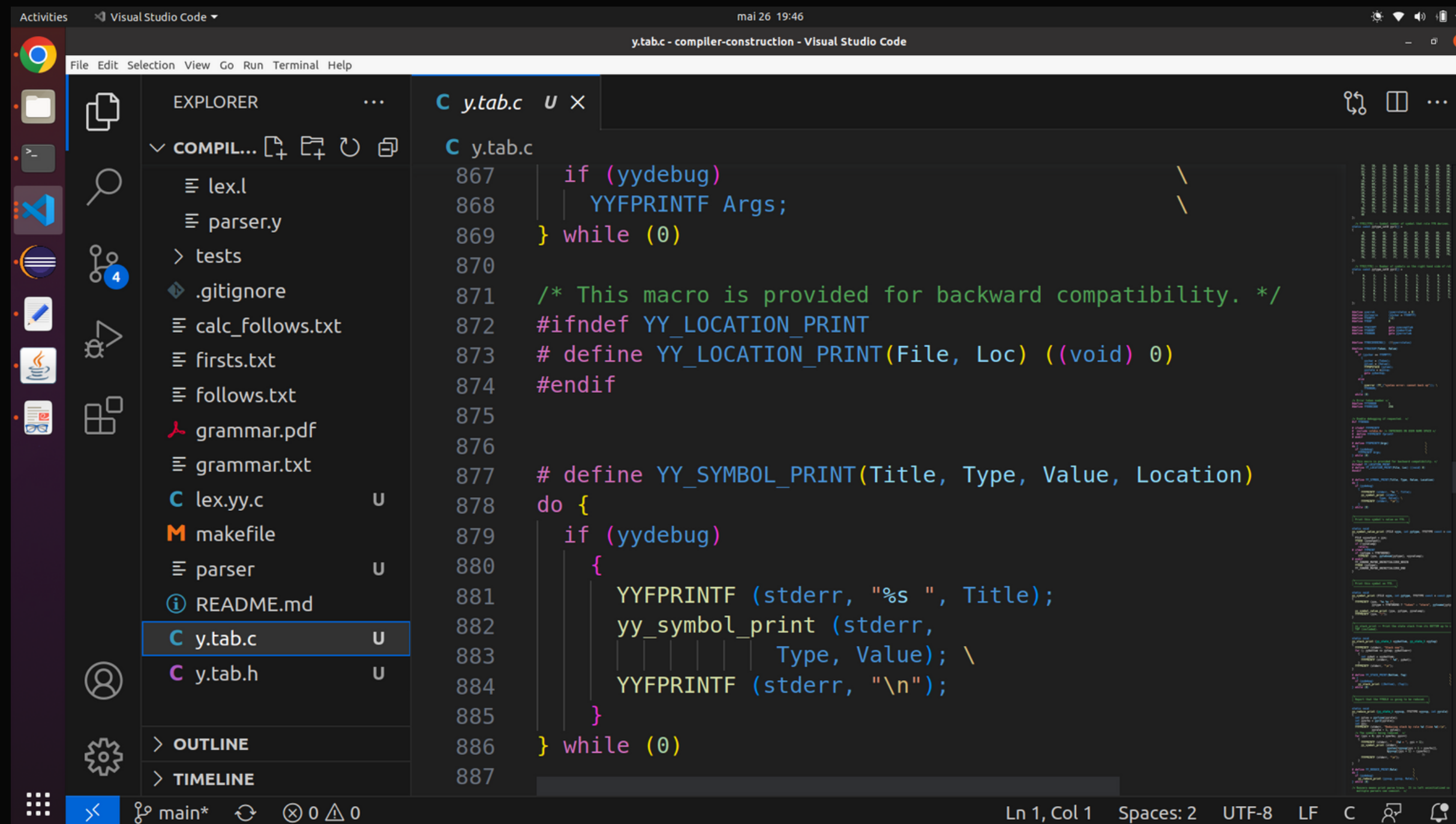

Qual é a entrada da ferramenta YACC?

```
src > parser.y
65 %start PROGRAM
66 %%
67
68 PROGRAM : STATEMENT
69         | FUNCLIST
70         ;
71
72
73 FUNCLIST : FUNCDEF FUNCLISTAUX;
74
75 FUNCLISTAUX : FUNCLIST
76             |
77             ;
78
79 FUNCDEF : DEF IDENT LPAREN PARAMLIST RPAREN LCURLYBRA
80
81 PARAMLIST : DATATYPE IDENT PARAMLISTAUX
82           |
83           ;
84
85 PARAMLISTAUX : COMMA PARAMLISTAUX
```

Ln 16, Col 1 Spaces: 7 UTF-8 LF Plain Text

Qual é a saída da ferramenta YACC?

- Saída do YACC: um analisador sintático em C ou C++



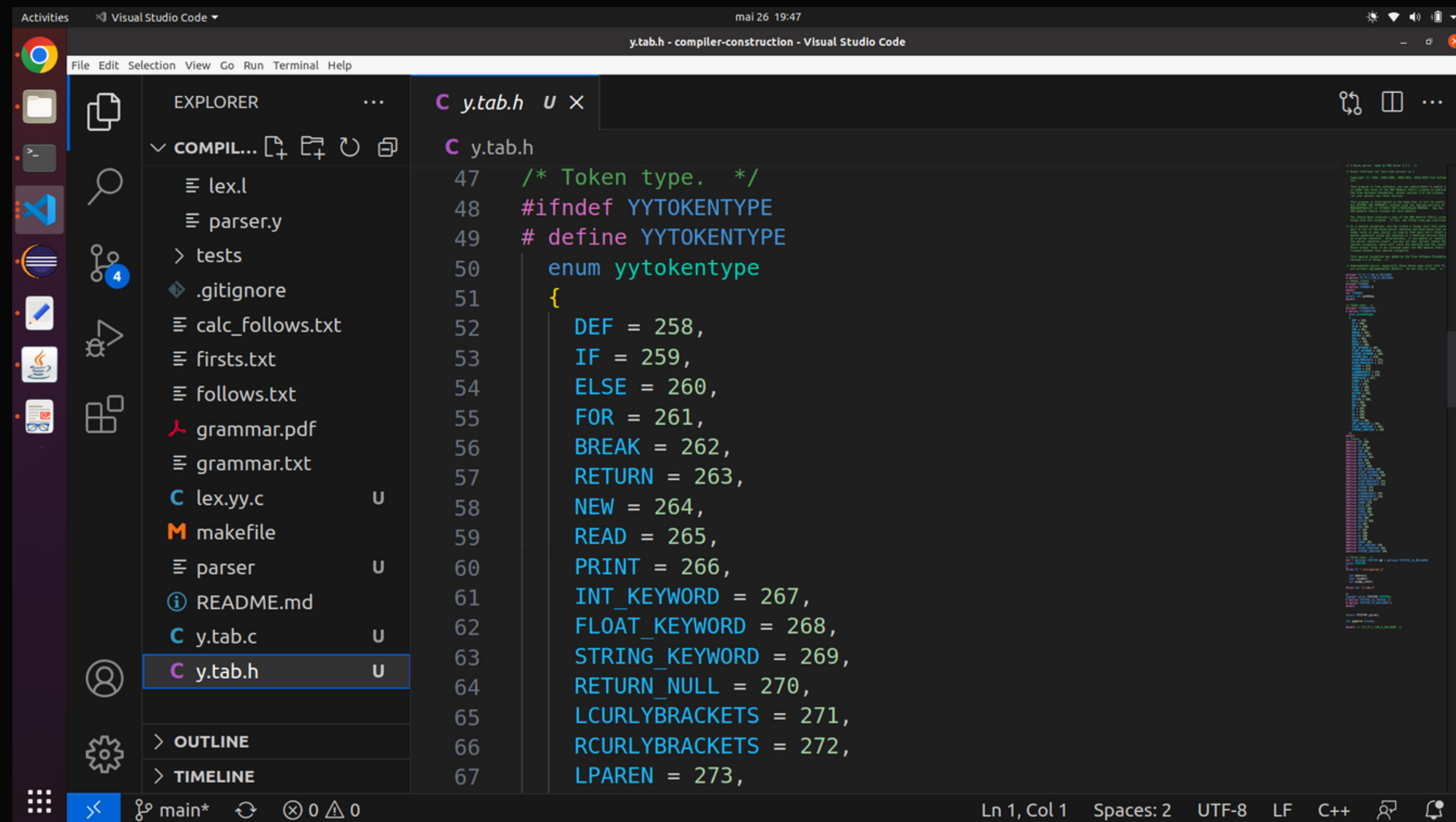
The screenshot shows the Visual Studio Code editor with the file `y.tab.c` open. The file contains the C code generated by YACC. The code includes a `while (0)` loop for debugging, a macro for backward compatibility, and a `YY_SYMBOL_PRINT` macro for printing symbols. The code is as follows:

```
867     if (yydebug) \
868         YYFPRINTF Args; \
869     } while (0)
870
871     /* This macro is provided for backward compatibility. */
872     #ifndef YY_LOCATION_PRINT
873     # define YY_LOCATION_PRINT(File, Loc) ((void) 0)
874     #endif
875
876     # define YY_SYMBOL_PRINT(Title, Type, Value, Location)
877     do {
878         if (yydebug)
879         {
880             YYFPRINTF (stderr, "%s ", Title);
881             yy_symbol_print (stderr,
882                             Type, Value); \
883             YYFPRINTF (stderr, "\n");
884         }
885     } while (0)
886
887
```

The Explorer view on the left shows the project structure, including files like `lex.l`, `parser.y`, `tests`, `.gitignore`, `calc_follows.txt`, `firsts.txt`, `follows.txt`, `grammar.pdf`, `grammar.txt`, `lex.yy.c`, `makefile`, `parser`, `README.md`, `y.tab.c`, and `y.tab.h`. The `y.tab.c` file is selected.

Qual é a saída da ferramenta YACC?

- YACC recebe como saída: juntamente com arquivos de cabeçalho.



The screenshot shows the Visual Studio Code interface. In the Explorer sidebar on the left, the file `y.tab.h` is highlighted under the `COMPIL...` folder. The main editor window displays the content of `y.tab.h`, which is a C header file generated by YACC. The code defines a token type enum and lists various keywords and symbols with their corresponding values.

```
47  /* Token type. */
48  #ifndef YYTOKENTYPE
49  # define YYTOKENTYPE
50  enum yytokentype
51  {
52      DEF = 258,
53      IF = 259,
54      ELSE = 260,
55      FOR = 261,
56      BREAK = 262,
57      RETURN = 263,
58      NEW = 264,
59      READ = 265,
60      PRINT = 266,
61      INT_KEYWORD = 267,
62      FLOAT_KEYWORD = 268,
63      STRING_KEYWORD = 269,
64      RETURN_NULL = 270,
65      LCURLYBRACKETS = 271,
66      RCURLYBRACKETS = 272,
67      LPAREN = 273,
```


Conclusões e Resumo

1. CC-2023-1 está na forma BNF. Coloque-a na forma convencional de gramática. Chame tal gramática de ConvCC-2023-1.
2. A sua ConvCC-2023-1 possui recursão à esquerda? Justifique detalhadamente sua resposta. Se ela tiver recursão à esquerda, então remova tal recursão.
3. A sua ConvCC-2023-1 está fatorada à esquerda? Justifique detalhadamente sua resposta. Se ela não estiver fatorada à esquerda, então fatore.
4. Faça ConvCC-2023-1 ser uma gramática em LL(1). É permitido adicionar novos terminais na gramática, se achar necessário. Depois disso, mostre que ConvCC-2023-1 está em LL(1) (você pode usar o Teorema ou a tabela de reconhecimento sintático vistos em videoaula).
6. se usou ferramenta, uma descrição da entrada exigida pela ferramenta e da saída dada por ela.