**Tema 1: Analizator lexical**

Documentatie

Analizatorul lexical este implementat pentru un subset al limbajului de programare C++. Minilimbajul contine atat tipuri de date simple (int, double), cat si compuse (int[]). Instructiunile premise sunt cele de atribuire, intrare/iesire, selectie (conditionata) si de ciclare.

## Clasificarea atomilor

1. ***Identificatori***

ID ::= litera (litera|cifra|”\_”)\*

litera ::= “A”|”B”|…|”Z”|”a”|”b”|…|”z”

cifra ::= ”0”|“1”|”2”|“3”|”4”|”5”|”6”|”7”|”8”|”9”

1. ***Constante***

CONST\_STRING ::= “”” {cifra | litera | caracter\_special} “””

cifra ::= ”0”|“1”|”2”|“3”|”4”|”5”|”6”|”7”|”8”|”9”

litera ::= “a”|”b”|”c”| … | “z” |”A”|”B”|…|”Z”

caracter\_special ::= “~”|”!”|”@”|”#”|”$”|”%”|”^”|”&”|” “ |”\*“|”(“|”)”|”\_”|”-”|”+”|”=”|

“`”|”[”|”{“|”]”|”}”|”\”|”|”|”;”|”:”|”’”|””””|”,”|”<”|”.”|”>”|”?”|”/”

CONST\_POZITIV ::= cifra\_nenula (cifra\_nenula|”0”)\* [ “,” (cifra\_nenula|”0”)\* cifra\_nenula ]

CONST ::= “0” | ([“+”|”-”] cifra\_nenula (cifra\_nenula|”0”)\* [ “,” (cifra\_nenula|”0”)\* cifra\_nenula ])

cifra\_nenula ::= “1”|”2”|“3”|”4”|”5”|”6”|”7”|”8”|”9”

1. ***Cuvinte rezervate***

“int”, “double”, “void”, “main”, “cout”, “cin”, “while”, “if”, “else”, “M\_PI”, “endl”

1. ***Operatori***

“+”, “-”, “\*”, “<<”, “>>”, “=”, “!=”, “> ”, ”<”, “<=”, “>=”, “==”, “[”, “]”

1. ***Separatori***

“(”, “)”, “{”, “}”, “,”, “;”, “ ”, “ “ ”

## Specificarea minilimbajului de programare folosind EBNF

**program** ::= “int” “main” “(” “)” “{” [lista\_decl] [lista\_instr] “}”

**lista\_decl** ::= dec “;” {dec “;”}

**dec** ::= tip lista\_def

**tip** ::= (“int” | “double”) [ “[” CONST\_POZITIV|ID “]” ]

**lista\_def** ::= definire {“,” definire}

**definire** ::= ID [ “=” CONST ]

**lista\_instr** ::=instr {“;” instr}

**instr** ::= atribuire | instr\_if | instr\_loop | intrare | iesire

**atribuire** ::= ID “=” variabila { operator\_ aritm variabila}

**variabila** ::= ID | CONST | variabila\_compusa

**variabila\_compusa** ::= ID “[” (CONST\_POZITIV | ID) “]”

**operator\_aritm** ::= “\*”|”-”|”+”

**instr\_if** ::= “if” ”(” conditie “)” “{” lista\_instr “}” [“else” “{” lista\_instr “}” ]

**conditie** ::= variabila operator\_rel variabila

**operator\_rel** ::= “!=”|“==”|“> ”|”<”|“<=”|“>=”

**instr\_loop** ::= “while” “(” conditie “)” “{” lista\_instr ”}”

**intrare** ::= “cout” scriere {scriere}

**scriere** ::= “<<” ( variabila | “endl” | CONST\_STRING )

**iesire** ::= “cin” “>>” ID

# Analizatorul lexical

Algoritmul corespunzator analizei lexicale functioneaza in urmatorul mod: dupa ce un atom a fost detectat se incearca clasificarea lui prin 3 instructiuni *if*. Pentru a nu se face o clasificare gresita a unui atom, mai intai se testeaza apartenenta sa la elementele limbajului: cuvant rezervat, operator, separator. Pe ramura *else* se verifica daca atomul este identificator. In caz contrar, se intra din nou pe ramura *else* si se verifica daca cuvantul gasit este constanta. Daca nu este nici constanta, atunci analizatorul a identificat o eroare lexicala.

Odata clasificat, atomul este scris in fisier cu ajutorul functiei *generareFip(cod: int, codTs: int, outputFile: string)*. Pentru cuvintele cheie, separatori si operatori, codul din tabela de simboluri este implicit 0. Pentru identificatori si constante se va cauta codul corespunzator in tabelele de simboluri (tabelele sunt separate pentru constante si identificatori). Daca atomul nu exista in tabela, atunci se va adauga o noua intregistarea.

Cuvintele cheie, separatorii si operatorii se vor citi din fisiere. Pentru identificatori si constante se folosesc expresii regulare. Identificatorii nu pot avea mai mult de 8 caractere.