Exercitiul 1: Specificarea minilimbajului de programare

1. Tipuri de date simple si tip de date definit de utilizator
2. Integer - tip de date pentru numere intregi(int)
3. Float - tip de date pentru numere reale(float)
4. Char - tip de date pentru caractere (char)
5. Struct - tip de date definit de utilizator. Acesta permite gruparea mai multor tipuri de date simple sau structuri de date ( struct circle { float r;};
6. String - tip de date pentru șiruri de caractere (string)

2. Instructiuni

1. de atribuire: se realizeaza prin intermediului operatorului de atribuire “=”
2. de intrare/iesire: se realizeaza prin intermediul :

* Functiei de citire: cin >> nume\_variabila;
* Functie de afisare: cout<<nume\_varaibila;
* Functia de returnare: return nume\_variabilă; (ex: return a;) - se folosește pentru a returna o valoare dintr-o funcție

1. de selectie (conditionala) : se realizeaza prin intermediul:

if (condiție) { // bloc de instrucțiuni } else { // bloc de instrucțiuni }

4. de ciclare :

* Se realizează prin intermediul: while (condiție) { // bloc de instrucțiuni }
* Se mai realizeaza si prin intermediul : do { // bloc de instrucțiuni } while (condiție); (ex: do { cout << a; } while (a > 5);)
* Se mai realizeaza si prin intermediul : for (initializare; conditie; actualizare) {

// bloc de instrucțiuni }

3. Restrictii

* Identificatori:
  + Pot conține litere, cifre și underscore (\_), dar nu pot începe cu o cifră.
  + Sunt case sensitive.
  + Nu pot fi cuvinte cheie.
* Comentarii:
  + Sunt permise comentariile pe o singură linie, începând cu “//”.
  + Sunt permise comentariile pe mai multe linii, între “/\*” și “\*/”.
* Tipul de date Struct:
  + Se definește prin intermediul cuvântului cheie struct.
  + Membrii structurii sunt accesați cu operatorul . (ex: a.b).
  + Atribuirea membrilor se face prin intermediul operatorului “=” (ex: a.b = 5;).
* Pot returna decat in functii de tip int sau float
* Pot avea un singur return in functie

1. Simboluri:

* Operatori:
  1. Aritmetici: +, -, \*, /, %
  2. De comparație: <, >, <=, >=, ==, !=
  3. De atribuire: =
  4. De separare: ,, ;
* Delimitatori: (, ), {, }, [, ], ;
* Cuvinte cheie:
  1. int
  2. float
  3. char
  4. struct
  5. string
  6. if
  7. while
  8. cin >>
  9. cout <<
  10. else
  11. return
  12. do

#### 5 Identificatori:

* Definiție:
  + identifier = letter (letter | digit | \_)\*
  + letter = a | b | ... | z | A | B | ... | Z
  + digit = 0 | 1 | ... | 9

FORMA BNF :

<program> ::= <antet> ? <functie>

<antet> ::= <includeri\_biblioteci> <lista\_declarare>

< includeri\_biblioteci > ::= “#include” <biblioteca> < includeri\_biblioteci > | eps;

<declartion\_list> ::= <declaration> < lista\_declarare > | eps;

<biblioteca> ::= “<iostream>”

<functie> ::= <functie\_antet> <body>

< functie\_antet> ::= <type> <EID> ( <lista\_parametrii>? )

<type> ::= int | float | struct | void

<lista\_parametrii> ::= <declaratie> , < lista\_parametrii> | <declaratie>

<declaratie> ::= <type> ID ; | “struct” ID { <declaratie>\* } ;

<body> ::= { <instructiuni\_compuse> }

<instructiuni\_compuse> ::= <instructiune> <instructiuni\_compuse> | <instructiune>

<instructiune> ::= <declaratie> ; | <assignment> ; | <instructiuni\_input> ; | <instructiuni\_output> ;

| <while\_loop> | <if\_statement> |<optional\_return>; | <do\_while\_loop>;

<assignment> ::= <EID> = <expresii\_aritmetice> ;

<instructiuni\_input> ::= cin >> <EID> ;

<instructiuni\_output> ::= cout << <expresii\_aritmetice> ; | cout << endl ;

<while\_loop> ::= while ( <conditie> ) <body>

<do\_while\_loop> ::= do <body> while ( <condition> ) ;

<if\_statement> ::= if ( <conditie> ) <body> ( else <body> )?

<conditie> ::= <expresii\_aritmetice> <operatori\_relationali> <expresii\_aritmetice>

| <expresii\_aritmetice>

<expresii\_aritmetice> ::= <expresii\_aritmetice> <operator\_aritmetic> <expresii\_aritmetice>

| <EID> | CONST

<operator\_aritmetic> ::= + | - | \* | / | %  
<operatori\_relationali> ::= != | == | < | > | <= | >=  
<EID> ::= ID| ID "." ID  
<const\_int> ::= <zero> | ( + | - ) ? <nr\_valid>

<nr\_valid> ::= <non-zero> <digit>\*

<zero> ::= “0”

<non-zero> ::= "1" | ... | "9"

<const\_float> ::= <const\_int> "." <digit> \*  
<const\_string> ::= "\"" { character } "\"" ;  
<digit>::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

<letter>::= "a" | "b" | ... | "z" | "A" | "B" | ... | "Z" ;  
<character>::= <letter> | <digit> | " " | "!" | "#" ;

ID ::= ( <digit> | "\_")\*  
CONST ::= <const\_int> | <const\_float> | <const\_string>

* ***Tabel de Codificare(TC)***

|  |  |
| --- | --- |
| Atom | ID |
| ID | 0 |
| CONST | 1 |
| ( | 2 |
| ) | 3 |
| , | 4 |
| int | 5 |
| float | 6 |
| char | 7 |
| main | 8 |
| struct | 9 |
| string | 10 |
| { | 11 |
| } | 12 |
| = | 13 |
| + | 14 |
| - | 15 |
| \* | 16 |
| / | 17 |
| % | 18 |
| < | 19 |
| > | 20 |
| <= | 21 |
| >= | 22 |
| == | 23 |
| != | 24 |
| && | 25 |
| || | 26 |
| ! | 27 |
| ; | 28 |
| cin | 29 |
| cout | 30 |
| if | 31 |
| else | 32 |
| while | 33 |
| << | 34 |
| >> | 35 |
| . | 36 |
| return | 37 |
| void | 38 |
| null | 39 |
| #include | 40 |
| do | 41 |

4. Implementarea analizatorului lexical

* Date de intrare: fisier text cu un program scris in MLP
* Date de iesire: fisier text cu tabelele :
  + FIP - forma interna a programului sursa
  + TS - tabela de simboluri
* Programul semnaleaza erori lexicale care apar. Restrictii suplimentare:

1. TS: a. unica pt identificatori si constante b. separat pentru identificatori si constante

2. Organizare TS: a. Tabel ordonat lexicografic b. Tabel arbore binar de cautare (ordine lexicografica) c. Tabel de dispersie (hash) Se mai cere:

* implementarea structuriilor de date cerute pentru TS;
* salvare FIP and TS in fisiere text;Pozitia in TS (din fisierul corespunzator FIP) va fi numarul liniei din fisierul TS in care este stocata informatia referitoare la acel atom lexical.