Tipuri de date:  
- int : numere intregi  
- double : numere reale  
- siruri : sir de caractere  
  
  
Instructiuni:  
- atribuire: id = expr;   
- intrare / iesire: citire(id) | afisare(expr)   
- selectie:  
 daca (conditie) {  
 // instructiuni  
 }  
 altfel\_daca (conditie) {  
 // instructiuni  
 }  
 altfel {  
 // instructiuni  
 }  
- ciclare:  
 atata\_timp\_cat (conditie) {  
 // instructiuni  
 }  
 continua\_pentru (id = expr\_initial; conditie; id = id + pas) {  
 // instructiuni  
 }  
- reguli specifice:  
 -> se pot declara mai multe variabile pe aceeasi linie, separate prin virgula, dar fiecare initializata individual  
 -> identificatori: litere, cifre, \_, trebuie sa inceapa cu cifre, maxim 200 caractere  
 -> constante numerice: pot fi pozitive sau negative, reale sau intregi  
 -> constante sir: trebuie inchise intre ghilimele ""  
   
Analizatorul lexical:  
 Simboluri:  
 Operatori:  
 - aritmetici: +, -, \*, /, %  
 - atribuire: =  
 - egalitate | inegalitate: ==, !=  
 - relationali: <, <=, >, >=  
 - secventa: ,  
 Separatori: "{}", ";", "()"  
 Cuvinte cheie:  
 - int = int  
 - double = double  
 - siruri = string  
 - afisare = Console.WriteLine  
 - citire = Console.ReadLine  
 - daca = if  
 - altfel\_daca = if else  
 - altfel = else  
 - atata\_timp\_cat = while  
 - continua\_pentru = for  
  
 Identificatori:  
 - litere = "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"  
 - cifre = "0" | "1" | ... | "9"  
 - underscore = "\_"  
 - identifier = (cifre) { litere | cifre | underscore }{0,199}

Tabelu de coduri:  
 ATOM : COD  
   
 ID : 0  
 CONST\_NUM : 1  
 CONST\_SIR :2  
 int : 3  
 double : 4  
 siruri : 5  
 daca :6  
 altfel\_daca :7  
 altfel : 8  
 atata\_timp\_cat : 9  
 continua\_pentru : 10  
 citire : 11  
 afisare : 12  
 ( : 13  
 ) : 14  
 { : 15  
 } : 16  
 ; : 17  
 , : 18  
 = : 19  
 + : 20  
 - : 21  
 \* : 22  
 / : 23  
 % : 24  
 == : 25  
 != : 26  
 < : 27  
 > : 28  
 <= : 29  
 >= : 30  
 start : 31  
 end :32

BNF  
   
 <program> ::= <antet\_program> <functie>  
 <antet\_program> ::= "start"  
  
 <functie> ::= <antet\_functie> <corp>  
 <antet\_functie> ::= <tip> ID (<lista\_declaratii>) | <tip> ID ()  
 <lista\_declaratii> ::= <declarare\_functie> , <lista\_declaratii> | <declarare\_functie>  
 <declarare\_functie> ::= <tip> ID  
  
 <corp> ::= { <instr\_compusa> }  
 <instr\_compusa> ::= <instr> <instr\_compusa> | <instr>  
 <instr> ::= <declarare> | <atribuire> | <instr\_citire> | <instr\_afisare> | <instr\_if> | <instr\_while> | <instr\_for>  
  
 <declarare> ::= <tip> ID ";" <newlineuri> | <tip> <atribuire> <newlineuri>  
 <newlineuri> ::= <newline> <newlineuri> | E  
 <newline> ::= "\n"  
  
 <atribuire> ::= ID = <expr\_aritmetica> ; | ID += <expr\_aritmetica> ;  
  
 <instr\_citire> ::= citire(ID) ;  
 <instr\_afisare> ::= afisare(<expr\_aritmetica>) ;  
  
 <expr\_aritmetica> ::= <expr\_aritmetica> <operator\_aritmetic> <expr\_aritmetica> | ID | CONST  
 <operator\_aritmetic> ::= + | - | \* | / | %  
  
 <instr\_while> ::= atata\_timp\_cat(<conditie>) <corp>  
 <instr\_for> ::= continua\_pentru(ID = <expr\_aritmetica>; <conditie>; ID = ID + pas) <corp>  
  
 <conditie> ::= <expr\_aritmetica> <operator\_relational> <expr\_aritmetica> | <expr\_aritmetica>  
 <operator\_relational> ::= == | != | < | > | <= | >=  
  
 <instr\_if> ::= daca(<conditie>) <corp>  
 | altfel\_daca(<conditie>) <corp>  
 | altfel <corp>  
  
 ID ::= ^[0-9][0-9A-Za-z\_]{0,199}$   
 CONST ::= <const\_numeric> | <const\_sir>  
 <const\_numeric> ::= ^[+-]?[0-9]+(\.[0-9]+)?$   
 <const\_sir> ::= ^".\*"$   
   
  
Aria și perimetrul cercului  
   
start

{

double 1raza;

citire(1raza);

double 2pi = 3.14;

double 3A = 2pi \* 1raza \* 1raza;

double 4P = 2 \* 2pi \* 1raza;

afisare(3A);

afisare(" ");

afisare(4P);

}

end

CMMDC a două numere  
  
start

{

int 1a;

int 2b;

citire(1a);

citire(2b);

int 3rest = -1;

atata\_timp\_cat(2b != 0)

{

3rest = 1a % 2b;

1a = 2b;

2b = 3rest;

}

afisare(1a);

}

end

Suma a N numere  
  
start

{

int 1sum = 0;

int 2n;

int 3a;

citire(2n);

atata\_timp\_cat(2n > 0)

{

citire(3a);

1sum += 3a;

2n = 2n - 1;

}

afisare(1sum);

}

End

Erori reale în MLP și limbaj original

start

{

int 1x // eroare: lipseste ";"

1x = 67;

tip\_nou 2y; // eroare: tip\_nou nu exista

citire(3z);

}

End

Erori MLP, nu în limbaj original

start

{

int 1x, 2y; // eroare MLP: trebuie pe linii separate

citire(1x);

citire(2y);

afisare(1x \* 2y \* 2y);

return 0; // eroare MLP: return interzis

}

End

Ilinca

* Nu are else if
* Nu are for
* Nu are do while
* Nu are char uri
* Nu poate face a+=1 sau a++