# Language Specification:

1. Language Definition:

1.1 Alphabet:

1.1.a. Upper **(A-Z)** and lower-case letters **(a-z)** of the English alphabet

b. Underline character **'\_'**;

c. Decimal digits **(0-9)**;

Lexic:

a. Special symbols, representing:

- operators:

arithmetic: **+, -, \*, /, %**

assignment: **=**

bitwise logic: **~, &, |, ^**

bitwise shifts: **<<, >>**

boolean logic: **!, &&, ||**

conditional evaluation: **? :**

equality testing**: ==, !=**

increment and decrement: **++, --**

order relations: **<, <=, >, >=**

sequencing: **,**

- separators **[ ] { } ; :**

- reserved words:

break -> **stăi**

case -> **trabă**

char -> **cuvânt scurt**

const -> **variabilă fixă**

continue -> **trecem peste**

default -> **numa așa**

do -> **fă**

double -> **cuvânt dublu**

else -> **celălalt adevăr**

float -> **cuvânt cu puncte**

for -> **pentru**

printf -> **no arată-mi**

if -> **adevăr**

inline -> **liniuță**

int -> **cuvânt cu numere**

long -> **cuvânt cu numere lungi**

return -> **dă-napoi**

short -> **cuvânt și mai scurt**

sizeof -> **cât îi de mare?**

switch -> **înainte de trabă**

typedef -> **merge și așa**

void -> **nimic**

while -> **să margă**

b. Identifiers

- a sequence of letters and digits, such that the first character is a letter; the rule is:

identifier = ["\_"]letter[{letter | digit}]

letter = "A" | "B" | . ..| "Z" | "a" | "b" | ... | "z"

non\_zero\_digit = "1" |...| "9"

zero\_digit = "0"

digit = zero\_digit | non\_zero\_digit

sign = ["+" | "-"]

comma = “,”

c. Constants

1.integer:

integer = zero\_digit | sign non\_zero\_digit { digit }

2.character:

character = 'letter'|'digit'

3.double\_ float:

double\_float = sign | non\_zero\_digit {digit} [comma digit {digit}]

2.2 Syntax:

The words - predefined tokens are specified between " and ":

a) Sintactical rules:

program = “stmtlist”

stmtlist = stmt;{stmt;}

stmt = decl | simplestmt | structstmt

decl = type identifier

type = primarytypes | arraytype

primarytypes = “int” | “long” | “float” | "double" | "short" | "char"

arraytype = primarytipes ”[“ nr “]”

cmpstmt = “{“ stmtlist ”}”

simplestmt = assignment | outstmt

assignment = identifier “=” expression

expression = [!] (term | expression operation expression | “(” expression operation expression “)”)

operation = “+” | “-” | “\*” | “/” | “%” | “^” | "&" |

term = identifier | nr | char{char}

outstmt = "printf" "(" identifier ")" ";"

structstmt = ifstmt | whilestmt | forstmt | switchstmt | cmpstmt

ifstmt = “if” “(“ condition “)” stmt {(else ifstmt | else stmt)}

whilestmt = “while” condition compstmt

forstmt = “for” “(“ assignment; condition; identifier (increment | decrement) “)” stmtlist

switchstmt = "switch" "(" condition ")" casestmt{casestmt} "default" ":" stmtlist

casestmt = "case" ":" stmtlist "break;"

condition = expression relation expression

b) lexical rules:

identifier = letter | letter{letter}{digit}

letter = "A" | "B" |...| "Z" | "a" | "b" | ... | "z"

digit = "0" | "1" |...| "9"

relation = "<" | "<=" | "==" | "!=" | ">=" | ">" | "&&" | "||"   
  
Codification table:

|  |  |
| --- | --- |
| Token type | code |
| identifier | 0 |
| constant | 1 |
| [ | 2 |
| ] | 3 |
| { | 4 |
| } | 5 |
| ; | 6 |
| : | 7 |
| , | 9 |
| < | 10 |
| <= | 11 |
| > | 12 |
| >= | 13 |
| ++ | 14 |
| -- | 15 |
| == | 16 |
| != | 17 |
| ? : | 18 |
| ! | 19 |
| && | 20 |
| || | 21 |
| << | 22 |
| >> | 23 |
| ~ | 24 |
| & | 25 |
| | | 26 |
| ^ | 27 |
| = | 28 |
| + | 29 |
| - | 30 |
| \* | 31 |
| % | 32 |
| / | 33 |
| break | 34 |
| case | 35 |
| char | 36 |
| const | 37 |
| continue | 38 |
| default | 39 |
| do | 40 |
| double | 41 |
| else | 42 |
| float | 45 |
| for | 46 |
| printf | 47 |
| if | 48 |
| int | 49 |
| long | 50 |
| return | 51 |
| short | 52 |
| sizeof | 54 |
| static | 55 |
| switch | 57 |
| typedef | 58 |
| void | 60 |
| while | 61 |
| inline | 62 |