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 -> opreste

case -> caz

char -> caracter

const -> constanta

continue -> continua

default -> implicit

do -> fa

double -> dublu

else -> altfel

float -> pluteste

for -> pentru

printf -> printeaza

if -> daca

int -> numar

long -> lung

return -> intoarce

short -> scurt

sizeof -> marime

switch -> intrerupator

typedef -> definire

void -> vid

while -> cattimp

inline -> inlinie

b.identifiers

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

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

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

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

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

zero\_digit = "0"

sign = ["+" | "-"]

comma = “,”

c.constants

1.integer:

integer = zero\_digit | sign digit {(zero\_digit | digit)}

2.character:

character = 'letter'|'digit'

3.double\_ float:

double\_float = sign digit {digit} [comma digit {digit}]

2.2 Syntax:

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

a) Sintactical rules:

program = “cmpstmt”

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 stmt}

condition = expression RELATION expression

whilestmt = “while” condition compstmt

forstmt = “for” “(“ assignment; condition; “)” stmt

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

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

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 |
| ; | 8 |
|  | 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 |
| , | 34 |
| break | 35 |
| case | 36 |
| char | 37 |
| const | 38 |
| continue | 39 |
| default | 40 |
| do | 41 |
| double | 42 |
| else | 43 |
| float | 44 |
| for | 45 |
| printf | 46 |
| if | 47 |
| int | 48 |
| long | 49 |
| return | 50 |
| short | 51 |
| sizeof | 52 |
| static | 53 |
| switch | 54 |
| typedef | 55 |
| void | 56 |
| while | 57 |
| inline | 58 |