

Compilers Assignment 2

Name- Ayush singh

Roll no.- 150101013

Name – Bishwendra choudhary

Roll no.- 150101017

Name – Aman Agarwal

Roll no. – 150101005

Name – Soumik roy

Roll no. – 150101074

Terminals The set of terminals consists of six types of elements:

- 1. alphabet = a / ... / z / A / ... / Z***
- 2. number = 0 / ... / 9***
- 3. arithmetic op = (/) / + / - / * / /***
- 4. relational op = < / > / = / !***
- 5. logical op = & / /***
- 6. whitespace (tabs or line breaks)***

All lexemes of the grammar consist of combinations of these terminals. Hence, tokens of the grammar can be defined using these 4 sets.

Tokens The token classes are defined as follows:

1. KEYWORD = static | int | bool | break | return | print | read | if | else | while
2. ID = alphabet (alphabet | number)*
3. NUMCONST = (number)+
4. BOOLCONST = true | false
5. OPERATOR = arithmetic op | relational op | logical op
| {<=, >=, !=, ==}

6. DELIMITER = { // , ; , { , }

7. FUNCTION = ID (ID) *)

Grammar

Production Rules

type specifier \rightarrow *int* / *bool*

function dec \rightarrow *type specifier* *ID* (*parameters*) *statement*

parameters \rightarrow *parameter list* / *epsilon*

parameter list \rightarrow *parameter list* , *parameter type list* / *parameter type list*

parameter type list \rightarrow *type specifier* *ID*

statement \rightarrow *expression_stmt*

/compound_stmt
/constant_stmt
/conditional_stmt
/iteration_stmt
/jump_stmt
/return

expression_stmt -> expression;
/;

compound stmt -> { stmt list }
stmt list -> stmt list statement | epsilon

iteration_stmt -> while (expression) statement
/do statement while (expression);
/for(; ;) statement
/for(; ; expression) statement
/for(; expression;) statement
/for(; expression; expression)
statement
/for(expression; ;) statement

local_declaration_list -> declaration

/local_declaration_list declaration

/epison

declaration -> Precessor ID = Construct_Processor

/ID

/Memory ID = Construct_Memory

/ID

/Link ID = Construct_Link

/ID

/Job ID = Construct_Job

/ID

/Cluster ID = Construct_Cluster

/ID

/Global_Scheduler ID = Construct_Global_Scheduler

/Local_Scheduler ID = Construct_Local_Scheduler

/int ID = ID

/float ID = ID

/string ID = ID_STRING

/int ID[] = {arg_list}

/float ID[] = {arg_list}

/Processor ID[] = {CONSTRUCTOR_LIST}

/ Local_Scheduler ID[] = {CONSTRUCTOR_LIST}
/ Cluster ID[] = {CONSTRUCTOR_LIST}
/ Job ID[] = {CONSTRUCTOR_LIST}
/ Memory ID[] = {CONSTRUCTOR_LIST}
/ Link ID[] = {CONSTRUCTOR_LIST}
/ int ID[INT]
/ float ID[INT]
/ Processor ID[INT]
/ Cluster ID[INT]
/ Job ID[INT]
/ Memory ID[INT]
/ Link ID[INT]
/ Local_Scheduler ID[INT]

arg_list -> arg_list, expression
/ expression

expression -> ID = simple_expr
/ ID += simple_expr
/ ID -= simple_expr
*/ ID *= simple_expr*

/ID/= simple_expr

/simple_expr

simple_expr -> (simple_expr / and_expr / or_expr)

/and_expr

/or_expr

//ID(Parameter2)

//ID.ID(Parameter2)

Parameter2 -> Parameter2, ID = ID / epsilon / ID = rel_exp / ID

= [float_list] / ID = 'ID' / ID = "ID" / rel_exp / ID

float_list -> float_list, float / epsilon / float_list, ID

and_expr -> and_expr && unary_rel_expr

/unary_rel_expr

or_expr -> or_expr // unary_rel_expr

/unary_rel_expr

unary_rel_expr -> ! unary_rel_expr

/rel_expr

rel_expr -> sum_expr relop sum_expr

/sum_expr

relop \rightarrow *relational_op*
 / {<=,>=,==,!=}

sum_expr \rightarrow *sum_expr sumop term*
 /term

sumop \rightarrow *+*
 /-

term \rightarrow *term mulop unary_expr*
 /unary_expr

mulop \rightarrow ***
 //
 /%

unary_expr \rightarrow *unaryop unary_expr*
 /factor

factor \rightarrow *ID*

/ (expression)

/ call

/ constant

call -> ID (args)

args -> arg_list

/ e

constant -> NUMCONSTANT

/ true

/ false

conditional_expression -> or_expr / or_expr ? expression :

conditional_expression