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 specif ier → int | bool***

***function dec → type specif ier ID (parameters) statement***

***parameters → parameter list|epsilon***

***parameter list → parameter list , parameter type list| parameter type list***

***parameter type list → type specif ier ID***

***statement -> 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***

***| for ( expression ; ; expression ) statement***

***| for ( expression ; expression ; ) statement***

***| for ( expression ; expression ; expression ) statement***

***jump\_stmt -> continue ;***

***| break ;***

***| return ;***

***| return expression ;***

***conditional\_stmt -> if ( expression ) statement***

***| if ( expression ) statement else statement***

***| switch ( expression ) statement***

***constant\_stmt -> int***

***| float***

***| char***

***| id***

***`***

***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|epison | 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 -> relational\_op***

***| {<=,>=,==,!=}***

***sum\_expr -> sum\_expr sumop term***

***| term***

***sumop -> +***

***| -***

***term -> term mulop unary\_expr***

***| unary\_expr***

***mulop -> \****

***| /***

***| %***

***unary\_expr -> unaryop unary\_expr***

***| factor***

***factor -> ID***

***| (expression)***

***| call***

***| constant***

***call -> ID (args)***

***args -> arg\_list***

***| e***

***constant -> NUMCONSTANT***

***| true***

***| false***

***conditional\_expression -> or\_expr | or\_expr ? expression : conditional\_expression***