Group No.:- 35

Members:-

1. Laksh Singla - 2017A7PS0082P
2. Kunal Mohta - 2017A7PS0148P
3. Suyash Raj - 2017A7PS0191P
4. Shubham Saxena - 2017A7PS0302P

Function description:-

- **make_node(Label, child1, child2, ...) :-** Create and return a new AST node with label **Label** and pointers to children **child1, child2, ...**
- **make_linked_list() :-** Create and return pointer to the head of a newly created empty linked list.
- **insert_at_begin(node, linked_list) :-** Add **node** at the beginning of **linked_list**. Returns a pointer to the head of the modified linked list.
- **make_leaf(Label, lexeme) :-** Create a new AST leaf node with label **Label** and lexeme value **lexeme**.

AST rules:-

No.	Grammar rule	AST rules
1	<program> -> <moduleDeclarations> <otherModules>$_1$ <driverModule> <otherModules>$_2$	<program>.addr = make_node('<program>', <moduleDeclarations>.addr, <otherModules>$_1$.addr, <driverModule>.addr, <otherModules>$_2$.addr) free(<moduleDeclarations>, <otherModules>$_1$, <driverModule>, <otherModules>$_2$)
2	<moduleDeclarations>$_1$ -> <moduleDeclaration> <moduleDeclarations>$_2$	<moduleDeclarations>$_1$.addr = insert_at_begin(<moduleDeclaration>.addr, <moduleDeclarations>$_2$.addr) free(<moduleDeclaration>, <moduleDeclarations>$_2$)
3	<moduleDeclarations> -> EPS	<moduleDeclarations>.addr = make_linked_list() free(EPS)

4	<moduleDeclaration> -> DECLARE MODULE ID SEMICOL	ID.addr = make_leaf('ID', ID.lexval)
		<moduleDeclaration>.addr = make_node('<moduleDeclaration>', ID.addr)
		free(DECLARE, MODULE, ID, SEMICOL)

5	<otherModules>$_1$ -> <module> <otherModules>$_2$	<otherModules>$_1$.addr = insert_at_begin(<module>.addr, <otherModules>$_2$.addr) free(<module>, <otherModules>$_2$)

6	<otherModules> -> EPS	<otherModules>.addr = make_linked_list() free(EPS)
7	<driverModule> -> DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>	<driverModule>.addr = make_node('<driverModule>', <moduleDef>.addr) free(DRIVERDEF, DRIVER, PROGRAM, DRIVERENDDEF, <moduleDef>)

8	<module> -> DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef>	ID.addr = make_leaf('ID', ID.lexval)
		<module>.addr = make_node('<module>', ID.addr, <input_plist>.addr, <ret>.addr, <moduleDef>.addr)
		free(DEF, MODULE, ID, ENDDEF, TAKES, INPUT, SQBO, <input_plist>, SQBC, SEMICOL, <ret>, <moduleDef>)

9	<ret> -> RETURNS SQBO <output_plist> SQBC SEMICOL	<ret>.addr = <output_plist>.addr free(RETURNS, SQBO, <output_plist>, SQBC, SEMICOL)
10	<ret> -> EPS	<ret>.addr = NULL free(EPS)

11	<input_plist> -> ID COLON <dataType> <input_plist2>	ID.addr = make_leaf('ID', ID.lexval)
		list_node = make_node('input_plist_node', ID.addr, <dataType>.addr)
		<input_plist>.addr = insert_at_begin(list_node, <input_plist2>.list)
		free(ID, COLON, <dataType>, <input_plist2>)

12	<input_plist2>$_1$ -> COMMA ID COLON <dataType> <input_plist2>$_2$	ID.addr = make_leaf('ID', ID.lexval)
		list_node = make_node('input_plist_node', ID.addr, <dataType>.addr)
		<input_plist2>$_1$.list = insert_at_begin(list_node, <input_plist2>$_2$.list)
		free(COMMA, ID, COLON, <dataType>, <input_plist2>$_2$)

13	<input_plist2> -> EPS	<input_plist2>.list = make_linked_list() free(EPS)

14	<output_plist> -> ID COLON <type> <output_plist2>	ID.addr = make_leaf('ID', ID.lexval)
		list_node = make_node('output_plist_node', ID.addr, <type>.addr)
		<output_plist>.addr = insert_at_begin(list_node, <output_plist2>.list)
		free(ID, COLON, <type>, <output_plist2>)

15	<output_plist2>$_1$ -> COMMA ID COLON <type> <output_plist2>$_2$	ID.addr = make_leaf('ID', ID.lexval)
		list_node = make_node('output_plist_node', ID.addr, <type>.addr)
		<output_plist2>$_1$.list = insert_at_begin(list_node, <output_plist2>$_2$.list)
		free(COMMA, ID, COLON, <type>, <output_plist2>$_2$)

16	<output_plist2> -> EPS	<output_plist2>.list = make_linked_list() free(EPS)
17	<dataType> -> INTEGER	<dataType>.addr = make_leaf('INTEGER', INTEGER.lexval) free(INTEGER)
18	<dataType> -> REAL	<dataType>.addr = make_leaf('REAL', REAL.lexval) free(REAL)
19	<dataType> -> BOOLEAN	<dataType>.addr = make_leaf('BOOLEAN', BOOLEAN.lexval) free(BOOLEAN)

20	<dataType> -> ARRAY SQBO <range_arrays> SQBC OF <type>	ARRAY.addr = make_leaf('ARRAY', ARRAY.lexval)
		<dataType>.addr = make_node('<dataType>', ARRAY.addr, <range_arrays>.addr, <type>.addr)
		free(ARRAY, SQBO, <range_arrays>, SQBC, OF, <type>)

21	<range_arrays> -> <index_nt>$_1$ RANGEOP <index_nt>$_2$	<range_arrays>.addr = make_node('<range_arrays>', <index_nt>$_1$.addr, <index_nt>$_2$.addr) free(<index_nt>$_1$, RANGEOP, <index_nt>$_2$)
22	<type> -> INTEGER	<type>.addr = make_leaf('INTEGER', INTEGER.lexval) free(INTEGER)
23	<type> -> REAL	<type>.addr = make_leaf('REAL', REAL.lexval) free(REAL)
24	<type> -> BOOLEAN	<type>.addr = make_leaf('BOOLEAN', BOOLEAN.lexval) free(BOOLEAN)
25	<moduleDef> -> START <statements> END	<moduleDef>.addr = <statements>.addr free(START, <statements>, END)
26	<statements> -> <statement> <statements>	<statements>.addr = insert_at_begin(<statement>.addr, <statements>.addr) free(<statement>, <statements>)
27	<statements> -> EPS	<statements>.addr = make_linked_list() free(EPS)
28	<statement> -> <ioStmt>	<statement>.addr = <ioStmt>.addr free(<ioStmt>)
29	<statement> -> <simpleStmt>	<statement>.addr = <simpleStmt>.addr free(<simpleStmt>)
30	<statement> -> <declareStmt>	<statement>.addr = <declareStmt>.addr free(<declareStmt>)
31	<statement> -> <condionalStmt>	<statement>.addr = <condionalStmt>.addr free(<condionalStmt>)
32	<statement> -> <iterativeStmt>	<statement>.addr = <iterativeStmt>.addr

		free(<iterativeStmt>)

33	<ioStmt> -> GET_VALUE BO ID BC SEMICOL	ID.addr = make_leaf('ID', ID.lexval)
		<ioStmt>.addr = make_node('input_stmt', ID.addr)
		free(GET_VALUE, BO, ID, BC, SEMICOL)

34	<ioStmt> -> PRINT BO <var> BC SEMICOL	<ioStmt>.addr = make_node('output_stmt', <var>.addr) free(PRINT, BO, <var>, BC, SEMICOL)
35	<boolConstt> -> TRUE	<boolConstt>.addr = make_leaf('TRUE', TRUE.lexval) free(TRUE)
36	<boolConstt> -> FALSE	<boolConstt>.addr = make_leaf('FALSE, FALSE.lexval) free(FALSE)

37	<var_id_num> -> ID <whichId>	ID.addr = make_leaf('ID', ID.lexval)
		If (<whichId>.addr == NULL**) {**
		    <var_id_num>.addr = ID.addr
		}
		else {
		    <var_id_num>.addr = make_node('<var_id_num>', ID.addr, <whichId>.addr)
		}
		free(ID, <whichId>)

38	<var_id_num> -> NUM	<var_id_num>.addr = make_leaf('NUM', NUM.lexval) free(NUM)
39	<var_id_num> -> RNUM	<var_id_num>.addr = make_leaf('RNUM'. RNUM.lexval) free(RNUM)
40	<var> -> <var_id_num>	<var>.addr = <var_id_num>.addr free(<var_id_num>)
41	<var> -> <boolConstt>	<var>.addr = <boolConstt>.addr free(<boolConstt>)
42	<whichId> -> SQBO <index_nt> SQBC	<whichId>.addr = <index_nt>.addr free(SQBO, <index_nt>, SQBC)
43	<whichId> -> EPS	<whichId>.addr = NULL free(EPS)
44	<simpleStmt> -> <assignmentStmt>	<simpleStmt>.addr = <assignmentStmt>.addr free(<assignmentStmt>)
45	<simpleStmt> -> <moduleReuseStmt>	<simpleStmt>.addr = <moduleReuseStmt>.addr free(<moduleReuseStmt>)

46	<assignmentStmt> -> ID <whichStmt>	ID.addr = make_leaf('ID', ID.addr)
		<assignmentStmt>.addr = make_node('<assignmentStmt>', ID.addr, <whichStmt>.addr)
		free(ID, <whichStmt>)

47	<whichStmt> -> <lvalueIDStmt>	<whichStmt>.addr = <lvalueIDStmt>.addr free(<lvalueIDStmt>)
48	<whichStmt> -> <lvalueARRStmt>	<whichStmt>.addr = <lvalueARRStmt>.addr free(<lvalueARRStmt>)
49	<lvalueIDStmt> -> ASSIGNOP <expression> SEMICOL	<lvalueIDStmt>.addr = make_node('<lvalueIDStmt>', <expression>.addr) free(ASSIGNOP, <expression>, SEMICOL)
50	<lvalueARRStmt> -> SQBO <index_nt> SQBC ASSIGNOP <expression> SEMICOL	<lvalueARRStmt>.addr = make_node('<lvalueARRStmt>', <index_nt>.addr, <expression>.addr) free(SQBO, <index_nt>, SQBC, ASSIGNOP, <expression>, SEMICOL)
51	<index_nt> -> NUM	<index_nt>.addr = make_leaf('NUM', NUM.lexval) free(NUM)
52	<index_nt> -> ID	<index_nt>.addr = make_leaf('ID', ID.lexval) free(ID)

53	<moduleReuseStmt> -> <optional> USE MODULE ID WITH PARAMETERS <idList> SEMICOL	ID.addr = make_leaf('ID', ID.lexval)
		<moduleReuseStmt>.addr = make_node('<moduleReuseStmt>', <optional>.addr, ID.addr, <idList>.addr)
		free(<optional>, USE, MODULE, ID, WITH, PARAMETERS, <idList>, SEMICOL)

54	<optional> -> SQBO <idList> SQBC ASSIGNOP	<optional>.addr = <idList>.addr free(SQBO, <idList>, SQBC, ASSIGNOP)
55	<optional> -> EPS	<optional>.addr = NULL free(EPS)

56	<idList> -> ID <idList2>	ID.addr = make_leaf('ID', ID.lexval)
		list_head = insert_at_begin(ID.addr, <idList2>.list)
		<idList>.addr = make_node('<idList>', list_head)
		free(ID, <idList2>)

57	<idList2>$_1$ -> COMMA ID <idList2>$_2$	ID.addr = make_leaf('ID', ID.lexval)
		<idList2>$_1$.list = insert_at_begin(ID.addr, <idList2>$_2$.list)
		free(COMMA, ID, <idList2>$_2$)

58	<idList2> -> EPS	<idList2>.list = make_linked_list() free(EPS)
59	<expression> -> <arithmeticOrBooleanExpr>	<expression>.addr = <arithmeticOrBooleanExpr>.addr free(<arithmeticOrBooleanExpr>)
60	<expression> -> <unary_nt>	<expression>.addr = <unary_nt>.addr free(<unary_nt>)
61	<unary_nt> -> <unary_op> <new_NT>	<unary_nt>.addr = make_node('<unary_nt>',

		<unary_op>.addr, <new_NT>.addr) free(<unary_op>, <new_NT>)
62	<new_NT> -> BO <arithmeticExpr> BC	<new_NT>.addr = <arithmeticExpr>.addr free(BO, <arithmeticExpr>, BC)
63	<new_NT> -> <var_id_num>	<new_NT>.addr = <var_id_num>.addr
64	<unary_op> -> PLUS	<unary_op>.addr = make_leaf('PLUS', PLUS.lexval) free(PLUS)
65	<unary_op> -> MINUS	<unary_op>.addr = make_leaf('MINUS', MINUS.lexval) free(MINUS)
66	<arithmeticOrBooleanExpr> -> <AnyTerm> <arithmeticOrBooleanExpr2>	<arithmeticOrBooleanExpr2>.inh_addr = <AnyTerm>.addr <arithmeticOrBooleanExpr>.addr = <arithmeticOrBooleanExpr2>.addr free(<AnyTerm>, <arithmeticOrBooleanExpr2>)
67	<arithmeticOrBooleanExpr2>$_1$ -> <logicalOp> <AnyTerm> <arithmeticOrBooleanExpr2>$_2$	<arithmeticOrBooleanExpr2>$_2$.inh_addr = make_node(<logicalOp>.lexval, <arithmeticOrBooleanExpr2>$_1$.inh_addr, <AnyTerm>.addr) <arithmeticOrBooleanExpr2>$_1$.addr = <arithmeticOrBooleanExpr2>$_2$.addr free(<logicalOp>, <AnyTerm>, <arithmeticOrBooleanExpr2>$_2$)
68	<arithmeticOrBooleanExpr2> -> EPS	<arithmeticOrBooleanExpr2>.addr = <arithmeticOrBooleanExpr2>.inh_addr free(EPS)
69	<AnyTerm> -> <arithmeticExpr> <AnyTerm2>	<AnyTerm2>.inh_addr = <arithmeticExpr>.addr <AnyTerm>.addr = <AnyTerm2>.addr free(<arithmeticExpr>, <AnyTerm2>)
70	<AnyTerm> -> <boolConstt>	<AnyTerm>.addr = <boolConstt>.addr free(<boolConstt>)
71	<AnyTerm2> -> <relationalOp> <arithmeticExpr>	<AnyTerm2>.addr = make_node(<relationalOp>.lexval, <AnyTerm2>.inh_addr, <arithmeticExpr>.addr) free(<relationalOp>, <arithmeticExpr>)
72	<AnyTerm2> -> EPS	<AnyTerm2>.addr = <AnyTerm2>.inh_addr free(EPS)
73	<arithmeticExpr> -> <term> <arithmeticExpr2>	<arithmeticExpr2>.inh_addr = <term>.addr <arithmeticExpr>.addr = <arithmeticExpr2>.addr free(<term>, <arithmeticExpr2>)
74	<arithmeticExpr2>$_1$ -> <op1> <term> <arithmeticExpr2>$_2$	<arithmeticExpr2>$_2$.inh_addr = make_node(<op1>.lexval, <arithmeticExpr2>$_1$.inh_addr, <term>.addr) <arithmeticExpr2>$_1$.addr = <arithmeticExpr2>$_2$.addr free(<op1>, <term>, <arithmeticExpr2>$_2$)

75	<arithmeticExpr2> -> EPS	<arithmeticExpr2>.addr = <arithmeticExpr2>.inh_addr free(EPS)
76	<term> -> <factor> <term2>	<term2>.inh_addr = <factor>.addr <term>.addr = <term2>.addr free(<factor>, <term2>)
77	<term2>$_{1}$ -> <op2> <factor> <term2>$_{2}$	<term2>$_{2}$.inh_addr = make_node(<op2>.lexval, <term2>$_{1}$.inh_addr, <factor>.addr) <term2>$_{1}$.addr = <term2>$_{2}$.addr free(<op2>, <factor, <term2>$_{2}$)
78	<term2> -> EPS	<term2>.addr = <term2>.inh_addr free(EPS)
79	<factor> -> BO <arithmeticOrBooleanExpr> BC	<factor>.addr = <arithmeticOrBooleanExpr>.addr free(BO, <arithmeticOrBooleanExpr>, BC)
80	<factor> -> <var_id_num>	<factor>.addr = <var_id_num>.addr free(<var_id_num>)
81	<op1> -> PLUS	<op1>.addr = make_leaf('PLUS', PLUS.lexval) free(PLUS)
82	<op1> -> MINUS	<op1>.addr = make_leaf('MINUS', MINUS.lexval) free(MINUS)
83	<op2> -> MUL	<op2>.addr = make_leaf('MUL', MUL.lexval) free(MUL)
84	<op2> -> DIV	<op2>.addr = make_leaf('DIV', DIV.lexval) free(DIV)
85	<logicalOp> -> AND	<logicalOp>.addr = make_leaf('AND', AND.lexval) free(AND)
86	<logicalOp> -> OR	<logicalOp>.addr = make_leaf('OR', OR.lexval) free(OR)
87	<relationalOp> -> LT	<relationalOp>.addr = make_leaf('LT', LT.lexval) free(LT)
88	<relationalOp> -> LE	<relationalOp>.addr = make_leaf('LE', LE.lexval) free(LE)
89	<relationalOp> -> GT	<relationalOp>.addr = make_leaf('GT', GT.lexval) free(GT)
90	<relationalOp> -> GE	<relationalOp>.addr = make_leaf('GE', GE.lexval) free(GE)
91	<relationalOp> -> EQ	<relationalOp>.addr = make_leaf('EQ', EQ.lexval) free(EQ)

92	<relationalOp> -> NE	<relationalOp>.addr = make_leaf('NE', NE.lexval) free(NE)
93	<declareStmt> -> DECLARE <idList> COLON <dataType> SEMICOL	<declareStmt>.addr = make_node('<declareStmt>', <idList>.addr, <dataType>.addr) free(DECLARE, <idList>, COLON, <dataType>, SEMICOL)

94	<condionalStmt> -> SWITCH BO ID BC START <caseStmts> <default_nt> END	ID.addr = make_leaf('ID', ID.lexval)
		<condionalStmt>.addr = make_node('<condionalStmt>', ID.addr, <caseStmts>.addr, <default_nt>.addr)
		free(SWITCH, BO, ID, BC, START, <caseStmts>, <default_nt>, END)

95	<caseStmts> -> CASE <value> COLON <statements> BREAK SEMICOL <caseStmts2>	list_node = make_node('caseStmtNode', <value>.addr, <statements>.addr)
		<caseStmts>.addr = insert_at_begin(list_node, <caseStmts2>.list)
		free(CASE, <value>, COLON, <statements>, BREAK, SEMICOL, <caseStmts2>)

96	<caseStmts2>$_1$ -> CASE <value> COLON <statements> BREAK SEMICOL <caseStmts2>$_2$	list_node = make_node('caseStmtNode', <value>.addr, <statements>.addr)
		<caseStmts2>$_1$.list = insert_at_begin(list_node, <caseStmts2>$_2$.list)
		free(CASE, <value>, COLON, <statements>, BREAK, SEMICOL, <caseStmts2>$_2$)

97	<caseStmts2> -> EPS	<caseStmts2>.list = make_linked_list() free(EPS)
98	<value> -> NUM	<value>.addr = make_leaf('NUM', NUM.lexval) free(NUM)
99	<value> -> TRUE	<value>.addr = make_leaf('TRUE', TRUE.lexval) free(TRUE)
100	<value> -> FALSE	<value>.addr = make_leaf('FALSE', FALSE.lexval) free(FALSE)
101	<default_nt> -> DEFAULT COLON <statements> BREAK SEMICOL	<default_nt>.addr = make_node('<default_nt>', <statements>.addr) free(DEFAULT, COLON, <statements>, BREAK, SEMICOL)
102	<default_nt> -> EPS	<default_nt>.addr = NULL free(EPS)

103	<iterativeStmt> -> FOR BO ID IN <range> BC START <statements> END	ID.addr = make_leaf('ID', ID.lexval)
		<iterativeStmt>.addr = make_node('for_loop', ID.addr, <range>.addr, <statements>.addr)
		free(FOR, BO, ID, IN, <range>, BC, START, <statements>, END)

104	<iterativeStmt> -> WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END	<iterativeStmt>.addr = make_node('while_loop', <arithmeticOrBooleanExpr>.addr, <statements>.addr) free(WHILE, BO, <arithmeticOrBooleanExpr>, BC, START, <statements>, END)
105	<range> -> NUM_1 RANGEOP NUM_2	NUM_1.addr = make_leaf('NUM', NUM_1.lexval) NUM_2.addr = make_leaf('NUM', NUM_2.lexval) <range>.addr = make_node('<range>', NUM_1.addr, NUM_2.addr) free(NUM_1, RANGEOP, NUM_2)