COMPILER CONSTRUCTION

GROUP NO. - 10

Akash S Revankar - 2019A7PS0294P
Harsh Butani - 2019A7PS0022P
Hemant Singh Sisodiya - 2019A7PS0070P
Mohit Sharma - 2019A7PS0100P
Siddharth Upadhaya - 2019A7PS0033P

Semantic Rule for AST creation:

1. program -> otherfunctions mainfunction
Semantic Rules:
program.addr = mknode(program,otherfunctions.addr,mainfunction.addr)

2. otherfunctions -> function otherfunctions1
Semantic Rules:
otherfunctions.addr = mknode(otherfunctions,function.addr,otherfunctions1.addr)

3. parameter_list -> datatype TK_ID remaining_list
Semantic Rules:
parameter_list.addr =
mknode(parameter_list,datatype.addr,mkleaf(TK_ID,entry.TK_ID),remaining_list.addr)
free(TK_ID);

4. datatype -> primitivedatatype
Semantic Rules:
datatype.addr = primitivedatatype.addr
free(primitivedatatype);

5. datatype -> constructeddatatype
Semantic rule:
datatype.addr = constructedatatype.addr
free(constructeddatatype);

6. stmts -> typedefinitions declarations otherstmts returnstmt
Semantic Rules:
stmts.addr = mknode(stmts, typedefinitions.addr, declarations.addr,
otherstmts.addr, returnstmt.addr);

7. typedefinitions -> actualorredefined typedefinitions1
Semantic Rules:

typedefinitions.addr= mknode(typedefinitions, actualorredefined.addr, typedefinitions1.addr);

8. actualorredefined -> definetypestmt
Semantic Rules:
	actualorredefined.addr = definetypestmt.addr

free(definetypestmt)

9. actualorredefined -> typedefinition
Semantic Rules:
	actualorredefined.addr = typedefinition.addr
free(typedefinition)

10. fielddefinitions -> fielddefinition1 fielddefinition2 morefields
Semantic Rules:
	fielddefinitions.addr = mknode(fielddefinitions, fielddefinition1.addr, fielddefinition2.addr, morefields.addr)

11. fieldtype -> primitivedatatype
Semantic Rules:
	fieldtype.addr = primitivedatatype.addr;
	 free(primitivedatatype)

12. morefields -> fielddefinition morefields1
Semantic Rules:
	morefields.addr = mknode(morefields,fielddefinition.addr,morefields1.addr)

13. termprime -> highprecedenceoperator factor termprime1
Semantic Rules:
	termprime.addr = mknode(termprime,highprecedenceoperator.addr,factor.addr,termprime1.addr)

14. expprime -> lowprecedenceoperators term expprime1
Semantic Rules:

expprime.addr =
mknode(expprime,lowprecedenceoperators.addr,term.addr,expprime1.addr)

15. moreexpansions -> oneexpansion moreexpansions1
Semantic Rules:
moreexpansions.addr =
mknode(moreexpansions,oneexpansion.addr,moreexpansions1.addr)

16. option_single_constructed -> oneexpansion moreexpansions
Semantic Rules:
option_single_constructed.addr =
mknode(option_single_constructed,oneexpansion.addr,moreexpansions.addr)

17. otherstmts -> stmt otherstmts1
Semantic Rules:
otherstmts.addr = mknode(otherstmts, stmt.addr, otherstmts1.addr)

18. stmt -> funcallstmt
Semantic Rules:
stmt.addr= funcallstmt.addr
free(funcallstmt)

19. funcallstmt -> outputparameters TK_CALL TK_FUNID TK_WITH TK_PARAMETERS
inputparameters TK_SEM

Semantic Rules:
funcallstmt.addr = mknode(funcallstmt,outputparameters.addr,
mkleaf(TK_FUNID,entry.TK_FUNTID), inputparameters.addr)

free(TK_CALL)
free(TK_WITH)
free(TK_PARAMETERS)
free(TK_FUNID)
free(TK_SEM)

20. declarations -> declaration declarations1
Semantic Rules:

declarations.addr = mknode(declarations, declaration.addr, declarations1.addr)

21. stmt -> assignmentstmt
Semantic Rules:
stmt.addr = assignmentstmt.addr;
free(assignmentstmt)

22. stmt -> iterativestmt
Semantic Rules:
stmt.addr = iterativestmt.addr;
free(iterativestmt)

23. stmt -> conditionalstmt
Semantic Rules:
stmt.addr = conditionalstmt.addr;
free(conditionalstmt)

24. stmt -> iostmt
Semantic rule:
stmt.addr = iostmt.addr
free(iostmt);

25. assignmentstmt -> singleorrecid TK_ASSIGNOP arithmeticexpression TK_SEM
Semantic rule:
assigmentstmt.addr = mknode(assignmentstmt, singleorrecid.addr, TK_ASSIGNOP.addr, arithmeticexpression.addr);
free(TK_SEM);

26. arithmeticexpression -> term expprime
Semantic rule:
arithmeticexpression.addr = mknode(arithmeticexpression, term.addr, exprime.addr);

27. term -> factor termprime
Semantic rule:
term.addr = mknode(term, factor.addr, termprime.addr);

28. factor -> var
Semantic rule:
	factor.addr = var.addr
	free(var);

29. booleanexpression -> var1 relationalop var2
Semantic rule:
	booleanexpression.addr = mknode(boolean_relop, var1.addr, relationop.addr, var2.addr);

30. var -> singleorrecid
Semantic rule:
	var.addr=singleorrecid.addr;
	free(singleorrecid);

31. factor -> TK_OP arithmeticexpression TK_CL
Semantic rule:
	factor.addr=arithmeticexpression.addr;
	free(arithmeticexpression);
	free(TK_OP);
	free(TK_CL);

32. highprecedenceoperator -> TK_MUL
Semantic rule:
	highprecedenceoperators.addr=TK_MUL.addr

33. highprecedenceoperator -> TK_DIV
Semantic rule:
	highprecedenceoperators.addr=TK_DIV.addr

34. lowprecedenceoperators -> TK_PLUS
Semantic rule:
	lowprecedenceoperators.addr=TK_PLUS.addr

35. lowprecedenceoperators -> TK_MINUS

Semantic rule:

lowprecedenceoperators.addr=TK_MINUS.addr

36. booleanexpression -> TK_OP booleanexpression1 TK_CL logicalop TK_OP1 booleanexpression2 TK_CL1

Semantic rule:

booleanexpression.addr=mknode(boolean_logic,booleanexpression1.addr,logicalop.addr,booleanexpression2,addr);

free(TK_OP);

free(TK_OP1);

free(TK_CL);

free(TK_CL1);

37. booleanexpression -> TK_NOT TK_OP booleanexpression1 TK_CL

Semantic rule:

booleanexpression.addr=mknode(boolean_not,TK_NOT.addr,booleanexpression1.addr);

free(TK_OP);

free(TK_CL);

38. var -> TK_NUM

Semantic rule:

var.addr=mkleaf(TK_NUM,entry.TK_NUM);

free(TK_NUM);

39. var -> TK_RNUM

Semantic rule:

var.addr=mkleaf(TK_RNUM,entry.TK_RNUM);

free(TK_RNUM);

40. logicalop -> TK_AND

Semantic rule:

logicalop.addr=TK_AND.addr;

41. logicalop -> TK_OR;

Semantic rule:

logicalop.addr=TK_OR.addr;

42. relationalop -> TK_LT
Semantic rule:
relationalop.addr=TK_LT.addr;

43. relationalop -> TK_LE
Semantic rule:
relationalop.addr=TK_LE.addr;

44. relationalop -> TK_EQ
Semantic rule:
relationalop.addr=TK_EQ.addr;

45. relationalop -> TK_GT
Semantic rule:
relationalop.addr=TK_GT.addr;

46. relationalop -> TK_GE
Semantic rule:
relationalop.addr=TK_GE.addr;

47. relationalop -> TK_NE
Semantic rule:
relationalop.addr=TK_NE.addr;

48. returnstmt -> TK_RETURN optionalreturn TK_SEM
Semantic rule:
returnstmt.addr=mknode(returnstmt,TK_RETURN.addr,optionalreturn.addr);
free(TK_SEM);

49. optionalreturn -> TK_SQL idlist TK_SQR
Semantic rule:
optionalreturn.addr=idlist.addr;
free(idlist);
free(TK_SQL);
free(TK_SQR);

50. idlist -> TK_ID more_ids
Semantic rule:
idlist.addr=mknode("idlist",mkleaf(TK_ID,entry.TK_ID),more_ids.addr);
free(TK_ID);

51. more_ids -> TK_COMMA idlist
Semantic rule:
more_ids.addr = idlist.addr;
free(idlist);
free(TK_COMMA);

52. primitivedatatype -> TK_INT
Semantic rule:
primitivedatatype.addr = TK_INT.addr;

53. primitivedatatype -> TK_REAL
Semantic rule:
primitivedatatype.addr = TK_REAL.addr;

54. constructeddatatype -> TK_RECORD TK_RUID
Semantic rule:
constructeddatatype.addr = mknode(constructeddatatype, TK_RECORD.addr,
mkleaf(TK_RUID, entry.TK_RUID)) ;
free(TK_RUID);

55. constructeddatatype -> TK_UNION TK_RUID
Semantic rule:
constructeddatatype.addr = mknode(constructeddatatype, TK_UNION.addr,
mkleaf(TK_RUID, entry.TK_RUID))
free(TK_RUID);

56. constructeddatatype -> TK_RUID
Semantic rule:
constructeddatatype.addr = mkleaf(TK_RUID, entry.TK_RUID)
free(TK_RUID);

57. remaining_list -> TK_COMMA parameter_list
Semantic rule:
remaining_list.addr = parameter_list.addr
free(parameter_list)
free(TK_COMMA);

58. singleorrecid -> TK_ID option_single_constructed
Semantic rule:
singleorrecid.addr = mknode(singleorrecid, mkleaf(TK_ID, entry.TK_ID),
option_single_constructed.addr)
free(TK_ID);

59. oneexpansion -> TK_DOT TK_FIELDID
Semantic rule:
oneexpansion.addr = mknode(onexpansion, TK_DOT.addr,mkleaf(TK_FIELDID,
entry.TK_FIELDID))
free(TK_FIELDID);

60. outputparameters -> TK_SQL idlist TK_SQR TK_ASSIGNOP
Semantic rule:
outputparameters.addr = mknode(outputparameters, idlist.addr,
TK_ASSIGNOP.addr)
free(TK_SQL);
free(TK_SQR);

61. inputparameters -> TK_SQL idlist TK_SQR
Semantic rule:
inputparameters.addr = idlist.addr
free(TK_SQL);
free(idlist);
free(TK_SQR);

62. iterativestmt -> TK_WHILE TK_OP booleanexpression TK_CL stmt otherstmts TK_ENDWHILE
Semantic rule:
iterativestmt.addr = mknode(iterativestmt,TK_WHILE.addr, booleanexpression.addr,
stmt.addr, otherstmts.addr)

free(TK_OP);
free(TK_CL);
free(TK_ENDWHILE);

63. conditionalstmt -> TK_IF TK_OP booleanexpression TK_CL TK_THEN stmt otherstmts elsepart
Semantic rule:
conditionalstmt.addr = mknode(conditionalstmt, TK_IF.addr,
booleanexpression.addr, TK_THEN.addr, stmt.addr, otherstmts.addr, elsepart.addr)
free(TK_OP);
free(TK_CL);

64. elsepart -> TK_ELSE stmt otherstmts TK_ENDIF
Semantic rule:
elsepart.addr = mknode(elsepart,TK_ELSE.addr, stmt.addr, otherstmts.addr)
free(TK_ENDIF);

65. elsepart -> TK_ENDIF
Semantic rule:
elsepart.addr = NULL
free(TK_ENDIF);

66. iostmt -> TK_READ TK_OP var TK_CL TK_SEM
Semantic rule:
iostmt.addr = mknode(iostmt,mkleaf(TK_READ,"read"),var.addr)
free(TK_OP);
free(TK_CL);
free(TK_SEM);

67. iostmt -> TK_WRITE TK_OP var TK_CL TK_SEM
Semantic rule:
iostmt.addr = mknode(iostmt,TK_WRITE.addr,var.addr)
free(TK_OP);
free(TK_CL);
free(TK_SEM);

68. typedefinition -> TK_RECORD TK_RUID fielddefinitions TK_ENDRECORD

Semantic rule:

typedefinition.addr=mknode(typedefinition_record,TK_RECORD.addr,mkleaf(TK_RUID,entry.TK_RUID),fielddefinitions.addr)

free(TK_RUID);

free(TK_ENDRECORD);

69. typedefinition -> TK_UNION TK_RUID fielddefinitions TK_ENDUNION

Semantic rule:

typedefinition.addr=mknode(typedefinition_union,TK_UNION.addr, mkleaf(TK_RUID,entry.TK_RUID),fielddefinitions.addr)

free(TK_RUID);

free(TK_ENDUNION);

70. fielddefinition -> TK_TYPE fieldtype TK_COLON TK_FIELDID TK_SEM

Semantic rule:

fielddefinition.addr=mknode(fielddefinition,fieldtype.addr,mkleaf(TK_FIELDID,entry.TK_FIELDID));

free(TK_TYPE);

free(TK_COLON);

free(TK_FIELDID);

free(TK_SEM);

71.fieldtype =>TK_RUID

Semantic rule:

fieldtype.addr = mkleaf(TK_RUID,entry.TK_RUID);

free(TK_RUID);

72.declaration =>TK_TYPE datatype TK_COLON TK_ID global_or_not TK_SEM

Semantic rule:

declaration.addr=mknode(declaration,datatype.addr,mkleaf(TK_ID,entry.TK_ID),global_or_not.addr);

free(TK_TYPE);

free(TK_COLON);

free(TK_SEM);

free(TK_ID);

73.global_or_not =>TK_COLON TK_GLOBAL

Semantic rule:

global_or_not.addr=TK_GLOBAL.addr;
free(TK_COLON);

74.mainfunction ->TK_MAIN stmts TK_END;
Semantic rule:

mainfunction.addr = mknode(TK_MAIN.addr,stmts.addr);
free(TK_END);

75.function ->TK_FUNID input_par output_par TK_SEM stmts TK_END
Semantic rule:

function.addr=mknode(function,mkleaf(TK_FUNID,entry.TK_FUNID),input_par.addr,output_par.addr,stmts.addr);
free(TK_FUNID);
free(TK_SEM);
free(TK_END);

76.input_par ->TK_INPUT TK_PARAMETER TK_LIST TK_SQL parameter_list TK_SQR
Semantic rule:

input_par.addr=mknode(input_par,TK_INPUT.addr,parameter_list.addr);
free(TK_PARAMETER);
free(TK_LIST);
free(TK_SQL);
free(TK_SQR);

77.output_par->TK_OUTPUT TK_PARAMETER TK_LIST TK_SQL parameter_list TK_SQR
Semantic rule:

output_par.addr=mknode(output_par,TK_OUTPUT.addr,parameter_list.addr);
free(TK_PARAMETER);
free(TK_LIST);
free(TK_SQL);
free(TK_SQR);

78.a->TK_RECORD.
Semantic rule:
	a.addr=TK_RECORD.addr;

79.a ->TK_UNION
Semantic rule:

	a.addr=TK_UNION.addr;

80.definetypestmt ->TK_DEFINETYPE a TK_RUID1 TK_AS TK_RUID2
Semantic rule:
	definetypestmt.addr=mknode(definetypestmt,a.addr,mkleaf(TK_RUID1,entry.TK_RUI
	D1),mkleaf(TK_RUID2,entry.TK_RUID2));
	free(TK_DEFINETYPE);
	free(TK_AS);
	free(TK_RUID1)
	free(TK_RUID2);

81.otherfunctions -> EPSILON
Semantic Rule:
	otherfunctions = NULL

82. output_par -> EPSILON
Semantic Rule:
	output_par = NULL

83.optionalreturn -> EPSILON
Semantic Rule:
	optionalreturn = NULL

84.more_ids -> EPSILON
Semantic Rule:
	more_ids = NULL

85.expprime -> EPSILON
Semantic Rule:
	expprime = NULL

86.termprime -> EPSILON
Semantic Rule:
	termprime = NULL

87. remaining_list -> EPSILON
Semantic Rule:
	remaining_list = NULL

88. typedefinitions -> EPSILON
Semantic Rule:
	typedefinitions = NULL

89. morefields -> EPSILON
Semantic Rule:
	morefields = NULL

90. declarations -> EPSILON
Semantic Rule:
	declarations = NULL

91. global_or_not -> EPSILON
Semantic Rule:
	global_or_not = NULL

92. otherstmts -> EPSILON
Semantic Rule:
	otherstmts = NULL

93. option_single_constructed -> EPSILON
Semantic Rule:
	option_single_constructed = NULL

94. outputparameters -> EPSILON
Semantic Rule:
outputparameters = NULL

95. moreexpansions -> EPSILON
Semantic Rule:
moreexpansions = NULL