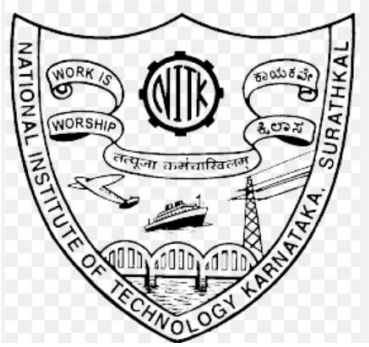
COMPILER DESIGN LAB
CS304

Project Phase3: Semantic Analysis



M Shree Harsha Bhat
211CS137
mshreeharshabhat.211cs137@nitk.edu.in

Shreekara Rajendra
211CS151
shreekararajendra.211cs151@nitk.edu.in

Shyam Balaji
211CS154
shyambalaji.211cs154@nitk.edu.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE – 57025

INTRODUCTION

ABSTRACT

In the realm of compiler design, semantic analysis stands as a pivotal phase in the translation process, where a compiler delves beyond the syntactic structure of high-level programming language statements to discern and validate their meaning. While syntax analysis ensures correct language constructs and arrangement, semantic analysis focuses on the broader semantics, aiming to guarantee that the code's intended logic aligns with the rules and constraints of the programming language.

When a programmer invokes a compiler for translation, it embarks on a multifaceted journey, with semantic analysis serving as a crucial checkpoint. At this juncture, the compiler scrutinizes the code's deeper intricacies, such as variable types, scope resolutions, and adherence to language-specific rules. Unlike syntax analysis, which verifies the surface-level correctness of statements, semantic analysis delves into the contextual nuances, aiming to uncover latent errors that may compromise the program's functionality or lead to unintended consequences.

This report zeroes in on the semantic analysis phase, shedding light on its significance in ensuring the program's logical coherence. In the semantic analysis process, our focus extends beyond the syntactic correctness established in earlier stages. We have implemented a semantic analyser tailored for the intricacies of the C programming language. This analyser employs a range of techniques to verify type compatibility, variable usage consistency, and adherence to language-specific semantics.

While syntax analysis acts as a gatekeeper for structural integrity, semantic analysis serves as the vigilant guardian of logical soundness, striving to unearth latent errors that may lurk beneath the surface. Through this meticulous examination, the semantic analyzer contributes significantly to the reliability and robustness of the final translated code, ensuring that the executed program faithfully embodies the programmer's intended logic.

C PROGRAM

The parser takes C source files as input for parsing. The input file is specified in the auxiliary functions section of the yacc script.

The workflow for testing the parser is as follows:

1. Compile the yacc script using the yacc tool

$ yacc -d parser.y

2. Compile the flex script using the flex tool

$ lex lexer.l

3. The first two steps generate lex.yy.c, y.tab.c, and y.tab.h. The header file is

included in lexer.l file. Then, lex.yy.c and y.tab.c are compiled together.

$ gcc lex.yy.c y.tab.c -w

4. Run the generated executable file

$ ./a.out

CODE

1. YACC File (.y)

```
%{
    #include<stdio.h>
```

```
int yylex();
int yyerror(char const * s);
extern FILE * yyin;
extern char *yytext;
extern int yylinenumber;
extern int intval;

#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define TABLE_SIZE 1003
#define MAX_SYMBOLS 1000000

//Scope Number
int scope=1; //Scope of Global Variables = 1
int stIterator=0;

int ID_dim =0 ;
int dimbuffer[30];
int dimit = 0;

int pointerLen = 1;
char ptrBuf[100];
int ptrBI=0;

int PD[20];
int pdi=0;

//Datatypes
char datatypesArray[][20]={"int","char","float","string"};
char CLASS[][20]={"variable","function","array","pointer","userDefType"};
int checkCompartibility(char *a,char *b){
    if(strcmp(a,datatypesArray[0])==0 && strcmp(b,datatypesArray[0])==0){
        return 0;
    }
    else if(strcmp(a,datatypesArray[1])==0 && strcmp(b,datatypesArray[1])==0){
        return 1;
    }
    else if(strcmp(a,datatypesArray[2])==0 && strcmp(b,datatypesArray[2])==0){
        return 2;
    }
    else if((strcmp(a,datatypesArray[0])==0 && strcmp(b,datatypesArray[1])==0) ||
    (strcmp(b,datatypesArray[0])==0 && strcmp(a,datatypesArray[1])==0)){
        return 0;
    }
    else if((strcmp(a,datatypesArray[0])==0 && strcmp(b,datatypesArray[1])==2) ||
    (strcmp(b,datatypesArray[0])==0 && strcmp(a,datatypesArray[1])==2)){
        return 2;
    }
    else if((strcmp(a,datatypesArray[0])==1 && strcmp(b,datatypesArray[1])==2) ||
    (strcmp(b,datatypesArray[0])==1 && strcmp(a,datatypesArray[1])==2)){
        return 2;
    }
    else{
```

```
            return -1;
        }
    }

    char * locations[MAX_SYMBOLS];
    int symTableSize=0;

    struct lines{
        int lineno; // stores the line-number
        struct lines*next;
    };
    struct SymbolTableEntry {
        char * name; //identifier
        char * type; //variable, array , pointer etc
        char * dataType; // int, char, float
        int dimension; // 0 for Variables, 1 for 1D arrays, 2 for 2D Arrays etc
        int * dimSize; //Array holding the size of the array
        int scope;
        int lineOfDeclaration;
        struct SymbolTableEntry * next;
        struct lines * head; //head of the linenumbers of reference list
        struct lines *  tail; //tail of the linenumbers of reference list
        int noOfParams;
        char ** paraList;
        char * paraTypes;
        int * paraDim;  //Dimension of array or pointers in function
        int symIndex;
    };

    struct SymbolTable {
        struct SymbolTableEntry * symTable[TABLE_SIZE];
    };

    struct SymbolTable Table;
    struct SymbolTable DST[10];

    //Hash Function
    int hashfunction(char *s) {
        int pp = 101;
        int mod = TABLE_SIZE;
        int val = 0;
        int n = strlen(s);
        int p = 1;
        for (int i = 0; i < n; i++) {
            int c = s[i];
            val = (val + ((c * p) % mod)) % mod;
            p = (p * pp) % mod;
        }
        return val;
    }

    // Function to insert a key-value pair (doesnt exist) into the SymbolTable
```

```
    struct SymbolTableEntry* insertIdentifier(char* key,char * type,char * dataType,int
dimension,int linenumber,int scope) {
        int index = hashfunction(key);

        // Create a new entry
        struct SymbolTableEntry* new_entry = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));
        struct SymbolTableEntry* new_entrydst = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));


        new_entry->name = strdup(key); // Duplicate the key string
        new_entrydst->name = strdup(key);

        new_entry->type = strdup(type); // Duplicate the string type
        new_entrydst->type = strdup(type);

        new_entry->dataType=strdup(dataType); //Duplicate the dataType of the String
        new_entrydst->dataType = strdup(dataType);

        new_entry->lineOfDeclaration=linenumber;
        new_entrydst->lineOfDeclaration = linenumber;

        new_entry->scope=scope;
        new_entrydst->scope = scope;

        new_entry->dimension=dimension;
        new_entrydst->dimension = dimension;

        new_entry->next = Table.symTable[index];
        new_entrydst->next = DST[scope].symTable[index];

        new_entry->head = (struct lines *)malloc(sizeof(struct lines));
        new_entrydst->head = (struct lines *)malloc(sizeof(struct lines));
        new_entry->head->lineno = linenumber;
        new_entrydst->head->lineno = linenumber;
        new_entry->head->next = NULL;
        new_entrydst->head->next =NULL;
        new_entry->tail = new_entry->head;
        new_entrydst->tail = new_entrydst->head;

        new_entry->dimSize=NULL;
        new_entrydst->dimSize=NULL;
        new_entry->noOfParams=-1;
        new_entrydst->noOfParams=-1;
        new_entry->paraList=NULL;
        new_entrydst->paraList=NULL;
        new_entry->paraTypes=NULL;
        new_entrydst->paraTypes=NULL;
        new_entry->paraDim=NULL;
        new_entrydst->paraDim=NULL;

        new_entrydst->symIndex=stIterator;
        new_entry->symIndex=stIterator;
```

```
        Table.symTable[stIterator++] = new_entry;
        DST[scope].symTable[index] = new_entrydst;

        locations[symTableSize++]=strdup(key);

        return new_entry;
    }

    struct SymbolTableEntry* insertArray(char* key,char * type,char * dataType,int
dimension,int linenumber,int scope,int * dimList) {
        int index = hashfunction(key);

        // Create a new entry
        struct SymbolTableEntry* new_entry = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));
        struct SymbolTableEntry* new_entrydst = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));


        new_entry->name = strdup(key); // Duplicate the key string
        new_entrydst->name = strdup(key);

        new_entry->type = strdup(type); // Duplicate the string type
        new_entrydst->type = strdup(type);

        new_entry->dataType=strdup(dataType); //Duplicate the dataType of the String
        new_entrydst->dataType = strdup(dataType);

        new_entry->lineOfDeclaration=linenumber;
        new_entrydst->lineOfDeclaration = linenumber;

        new_entry->scope=scope;
        new_entrydst->scope = scope;

        new_entry->dimension=dimension;
        new_entrydst->dimension = dimension;

        new_entry->next = Table.symTable[index];
        new_entrydst->next = DST[scope].symTable[index];

        new_entry->head = (struct lines *)malloc(sizeof(struct lines));
        new_entrydst->head = (struct lines *)malloc(sizeof(struct lines));

        new_entry->head->lineno = linenumber;
        new_entrydst->head->lineno = linenumber;

        new_entry->head->next = NULL;
        new_entrydst->head->next =NULL;

        new_entry->tail = new_entry->head;
        new_entrydst->tail = new_entrydst->head;

        new_entry->dimSize=(int *)malloc(sizeof(int)*dimension);
```

```
        if(dimList != NULL)
        {
            for(int i=0;i<dimension;i++){
                new_entry->dimSize[i]=dimList[i];
            }

            new_entrydst->dimSize=(int *)malloc(sizeof(int)*dimension);
            for(int i=0;i<dimension;i++){
                new_entrydst->dimSize[i]=dimList[i];
            }
        }


        new_entry->noOfParams=-1;
        new_entrydst->noOfParams=-1;
        new_entry->paraList=NULL;
        new_entrydst->paraList=NULL;
        new_entry->paraDim=NULL;
        new_entrydst->paraDim=NULL;

        new_entrydst->symIndex=stIterator;
        new_entry->symIndex=stIterator;

        Table.symTable[stIterator++] = new_entry;
        DST[scope].symTable[index] = new_entrydst;

        locations[symTableSize++]=strdup(key);

        return new_entry;
    }

    struct SymbolTableEntry* insertFunction(char* key,char * type,char * dataType,int
dimension,int linenumber,int scope,int params,char parList[20][20],char paraTypes[20]) {
        int index = hashfunction(key);

        // Create a new entry
        struct SymbolTableEntry* new_entry = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));
        struct SymbolTableEntry* new_entrydst = (struct
SymbolTableEntry*)malloc(sizeof(struct SymbolTableEntry));


        new_entry->name = strdup(key); // Duplicate the key string
        new_entrydst->name = strdup(key);

        new_entry->type = strdup(type); // Duplicate the string type
        new_entrydst->type = strdup(type);

        new_entry->dataType=strdup(dataType); //Duplicate the dataType of the String
        new_entrydst->dataType = strdup(dataType);

        new_entry->lineOfDeclaration=linenumber;
```

```
        new_entrydst->lineOfDeclaration = linenumber;

        new_entry->scope=scope;
        new_entrydst->scope = scope;

        new_entry->dimension=dimension;
        new_entrydst->dimension = dimension;

        new_entry->next = Table.symTable[index];
        new_entrydst->next = DST[scope].symTable[index];

        new_entry->head = (struct lines *)malloc(sizeof(struct lines));
        new_entrydst->head = (struct lines *)malloc(sizeof(struct lines));
        new_entry->head->lineno = linenumber;
        new_entrydst->head->lineno = linenumber;
        new_entry->head->next = NULL;
        new_entrydst->head->next =NULL;
        new_entry->tail = new_entry->head;
        new_entrydst->tail = new_entrydst->head;

        new_entry->dimSize=NULL;
        new_entrydst->dimSize=NULL;
        new_entry->noOfParams=params;
        new_entrydst->noOfParams=params;

        new_entry->paraList=(char **)malloc(sizeof(char*)*params);
        for(int i=0;i<params;i++){
            new_entry->paraList[i]=(char *)malloc(20*sizeof(char));
            strcpy(new_entry->paraList[i],parList[i]);
        }

        new_entrydst->paraList=(char **)malloc(sizeof(char*)*params);
        for(int i=0;i<params;i++){
            new_entrydst->paraList[i]=(char *)malloc(20*sizeof(char));
            strcpy(new_entrydst->paraList[i],parList[i]);
        }

        new_entry->paraTypes=(char *)malloc(sizeof(char)*params);
        for(int i=0;i<params;i++){
            new_entry->paraTypes[i]=paraTypes[i];
        }

        new_entrydst->paraTypes=(char *)malloc(sizeof(char)*params);
        for(int i=0;i<params;i++){
            new_entrydst->paraTypes[i]=paraTypes[i];
        }

        new_entry->paraDim=(int *)malloc(sizeof(int)*params);
        for(int i=0;i<params;i++){
            new_entry->paraDim[i]=PD[i];
        }

        new_entrydst->paraDim=(int *)malloc(sizeof(int)*params);
        for(int i=0;i<params;i++){
```

```
            new_entrydst->paraDim[i]=PD[i];
        }

        new_entrydst->symIndex=stIterator;
        new_entry->symIndex=stIterator;

        Table.symTable[stIterator++] = new_entry;
        DST[scope].symTable[index] = new_entrydst;

        locations[symTableSize++]=strdup(key);

        return new_entry;
    }

    //Function to append an entry for key-value pair that already exists in the
SymbolTable
    void update(struct SymbolTableEntry* entry,int linenumber){

        struct SymbolTableEntry* Entry=Table.symTable[entry->symIndex];
        struct lines * new_node = (struct lines *)malloc(sizeof(struct lines));
        new_node->lineno = linenumber;
        new_node->next = NULL;
        Entry->tail->next = new_node;
        Entry->tail = new_node;
    }

    // Function to Lookup
    struct SymbolTableEntry *  lookup(char* key,int scope,bool insertion) {
        int index = hashfunction(key);
        if(insertion){
            struct SymbolTableEntry* entry = DST[scope].symTable[index];
            while(entry != NULL){
                if(strcmp(entry->name,key) == 0)return entry;
                entry = entry->next;
            }
        }
        else{
            int tempscope = scope;
            while(tempscope>=1){
                struct SymbolTableEntry* entry = DST[tempscope].symTable[index];
                while(entry != NULL){
                    if(strcmp(entry->name,key) == 0)return entry;
                    entry = entry->next;

                }
                tempscope--;
            }
        }

        // Key not found
        return NULL;
    }

    void deleteDMT(int scope)
```

```
    {
        for(int i =0;i<TABLE_SIZE;i++)
        {
            struct SymbolTableEntry * temp = DST[scope].symTable[i];
            while(temp != NULL)
            {
                struct SymbolTableEntry * tempnext = temp;
                free(tempnext);
                temp = temp->next;

            }
            DST[scope].symTable[i] = NULL;
        }
    }


    struct SymbolTableEntry * varDecArray[100];
    int index=0;


    //To store type
    char typeBuffer[10];
    char userTypeBuffer[10];
    char userTypeNameBuffer[10];
    char IDBuffer[30];

    char funTypeBuffer[30];
    char funNameBuffer[30];

    //Global Variables for func parameters
    int noOfParameters=0;
    char PL[20][20];
    char PTL[20];

    struct SymbolTableEntry * forFunc;
    char pt;

    int arrayInFunc[100];
    int dimOfArrayInFunc=0;

    int parD[10];
    int parDi=0;
    int userTypeDim = 0;
%}

%union{
    char * value;
    char * type;
}
%type<type> type choice item PTR_TYPE idinsert idlook pointerAsAParameter userDefDataType
%token<type> NULLT TYPE INTEGER STRING FLOATING_NUM CHAR_CONST STRUCT UNION VOID
%token<value> ID FUN_START
```

```
%token INCLUDE PREDEF_HEADER  ELIF ELSE IF BREAK NOT FOR CONTINUE WHILE SWITCH CASE
RETURN  SL_COMMENT ML_COMMENT  EQUALTO OPEN_BRACK OPEN_FLOWER OPEN_SQ CLOSE_BRACK
CLOSE_FLOWER CLOSE_SQ AND UNARY_OP PLUS MINUS DIV MUL MOD OR AMPERSAND BIT_OR BIT_XOR
SEMICOLON COMMA ISEQUALTO LT LTE GT GTE NE PLUS_ET MINUS_ET MUL_ET DIV_ET OR_ET AND_ET
XOR_ET PRINTF SCANF MAIN COLON DEFAULT MALLOC SIZEOF TYPEDEF DOT ARROW

%right XOR_ET OR_ET AND_ET
%right PLUS_ET MINUS_ET MUL_ET DIV_ET EQUALTO
%left OR
%left AND
%left BIT_OR
%left BIT_XOR
%left AMPERSAND
%left ISEQUALTO NE
%left GT GTE
%left LT LTE
%left PLUS MINUS
%left MUL DIV MOD
%left NOT
%left UNARY_OP
%left OPEN_BRACK CLOSE_BRACK

%%
start: header startGlobal main {printf("Syntax is Correct\n");return;}
header: INCLUDE file newHeader
newHeader: header |
file : STRING | PREDEF_HEADER

startGlobal : function_defn | globalVarDec | userTypeDefination | userTypeDeclaration
startGlobal |
            |arrayDeclr startGlobal | arrayInitial startGlobal | PTR_INITIAL startGlobal |
PTR_DECLR startGlobal

globalVarDec : type idgi optionsG DGd SEMICOLON startGlobal
               | idgl optionsG DGl SEMICOLON startGlobal
DGd : COMMA idgi optionsG DGd |
DGl : COMMA idgl optionsG DGd |
optionsG : EQUALTO Exp | EQUALTO function_call1{
    if(strcmp(forFunc->dataType,"void")==0){
        char const errorMessage[100]="Void Function Does not return anything";
        int a=yyerror(errorMessage);
        return -1;
    }
} |

idgi : ID {
    struct SymbolTableEntry * search=lookup($1,1,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
```

```
        struct
SymbolTableEntry*new_entry=insertIdentifier($1,CLASS[0],typeBuffer,0,yylinenumber,1);
    }
}
idgl: ID {
            struct SymbolTableEntry * search=lookup($1,scope,false);
            if(search==NULL){
                char message[30]="Variable Not Declared";
                int res=yyerror(message);
                return -1;
            }
            else{
                update(search,yylinenumber);
            }
         }

main: MAIN mainParameters CLOSE_BRACK open_flower  start1 close_flower
mainParameters : parameter_list |

start1: varDec |print | scanf | function_call |  while | for
        | RETURN ExpF SEMICOLON start1 | CONTINUE SEMICOLON start1| BREAK SEMICOLON start1
        | SL_COMMENT start1 | ML_COMMENT start1 | arrayDeclr start1 | arrayInitial start1
        | switch | PTR_INITIAL start1 | PTR_DECLR start1 | ifElseLadder start1
        | userTypeDeclaration start1 | userTypeInitialization start1 |

type : TYPE {
            strcpy(typeBuffer,$1);
            $$=malloc(strlen($1)+1);
            strcpy($$,$1);
        }
varDec : type id options D SEMICOLON start1
         | id3 options D SEMICOLON start1
         | id3{dimit = 0;} BOX options SEMICOLON {
            struct SymbolTableEntry * entry=lookup(IDBuffer,scope,false);
            if(entry->dimension != dimit)
            {
                char const errorMessage[100]="Dimension doesnt match!";
                int a=yyerror(errorMessage);
                return -1;
            }
            for(int i =0;i<dimit;i++)
            {
                if(dimbuffer[i] != 10000 && entry->dimSize[i] <= dimbuffer[i] ||
dimbuffer[i] < 0)
                {
                    char const errorMessage[100]="Invalid Access";
                    int a=yyerror(errorMessage);
                    return -1;
                }
            }
            dimit = 0;
        } start1
        | ID{
            struct SymbolTableEntry * search=lookup($1,scope,false);
```

```
            if(search==NULL){
                char message[30]="Variable Not Declared";
                int res=yyerror(message);
                return -1;
            }
            else{
                update(search,yylinenumber);
            }
        } UNARY_OP SEMICOLON start1
        | UNARY_OP ID SEMICOLON{
            struct SymbolTableEntry * search=lookup($2,scope,false);
            if(search==NULL){
                char message[30]="Variable Not Declared";
                int res=yyerror(message);
                return -1;
            }
            else{
                update(search,yylinenumber);
            }
        } start1

D : COMMA id options D |
options : EQUALTO Exp | EQUALTO function_call1{
    if(strcmp(forFunc->dataType,"void")==0){
        char const errorMessage[100]="Void Function Does not return anything";
        int a=yyerror(errorMessage);
        return -1;
    }
} |

id: ID {
    struct SymbolTableEntry * search=lookup($1,scope,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
        struct
SymbolTableEntry*new_entry=insertIdentifier($1,CLASS[0],typeBuffer,0,yylinenumber,scope);
    }
}

open_flower : OPEN_FLOWER {scope++;}
close_flower : CLOSE_FLOWER {deleteDMT(scope);scope--;}

Exp: OPEN_BRACK Exp CLOSE_BRACK
     | Exp PLUS Exp
     | Exp MINUS Exp
     | Exp MUL Exp
     | Exp DIV Exp
     | Exp MOD Exp
     | Exp OR Exp
     | Exp AND Exp
```

```
     | Exp BIT_XOR Exp
     | Exp AMPERSAND Exp
     | Exp UNARY_OP
     | UNARY_OP Exp
     | NOT Exp
     | Exp BIT_OR Exp
     | Exp ISEQUALTO Exp
     | Exp LT Exp
     | Exp LTE Exp
     | Exp GT Exp
     | Exp GTE Exp
     | Exp NE Exp
     | Exp PLUS_ET Exp
     | Exp MINUS_ET Exp
     | Exp MUL_ET Exp
     | Exp DIV_ET Exp
     | Exp OR_ET Exp
     | Exp AND_ET Exp
     | Exp XOR_ET Exp
     | id3 {dimit = 0;}
     | id3 {dimit = 0;} BOX
     | INTEGER | CHAR_CONST

id3 : ID{
    struct SymbolTableEntry * found=lookup($1,scope,false);
        strcpy(IDBuffer,$1);
        if(found==NULL){
            char const errorMessage[100]="Variable Not Declared!!";
            int a=yyerror(errorMessage);
            return -1;
        }
        else{
            update(found,yylinenumber);
        }
    }

function_defn : function_declaration open_flower start1 close_flower startGlobal
function_declaration: fun_start parameter_list CLOSE_BRACK{
    struct SymbolTableEntry * entry = lookup(funNameBuffer,1,true);
    if(entry!=NULL){
        char message[40]="Function name is Already Declared";
        int res=yyerror(message);
        return -1;
    }
    struct SymbolTableEntry* new_entry =
insertFunction(funNameBuffer,CLASS[1],funTypeBuffer,0,yylinenumber,scope,noOfParameters,PL
,PTL);
    noOfParameters=0;
    pointerLen=1;
}
                      | fun_start CLOSE_BRACK {
                        struct SymbolTableEntry * entry = lookup(funNameBuffer,1,true);
                        if(entry!=NULL){
                            char message[40]="Function name is Already Declared";
```

```
                        int res=yyerror(message);
                        return -1;
                    }
                    struct SymbolTableEntry* new_entry =
insertFunction(funNameBuffer,CLASS[1],funTypeBuffer,0,yylinenumber,scope,noOfParameters,PL
,PTL);
                    noOfParameters=0;
                    pointerLen=1;
                }
parameter_list: parameter_list COMMA type ID choice{
    char flag;
    if(strcmp($5,"array")==0){
        flag='a';
    }
    else{
        flag='v';
    }
    char T[10];
    if(flag=='v'){
        strcpy(T,CLASS[0]);
    }
    else{
        strcpy(T,CLASS[2]);
    }
    struct SymbolTableEntry * search=lookup($4,scope+1,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else if(strcmp(T,"variable")==0){
        struct SymbolTableEntry*new_entry =
insertIdentifier($4,T,$3,0,yylinenumber,scope+1);
    }
    else if(strcmp(T,"array")==0){
        struct SymbolTableEntry*new_entry =
insertArray($4,T,$3,dimit,yylinenumber,scope+1,dimbuffer);
    }
    PD[noOfParameters]=dimit;
    dimit=0;
    strcpy(PL[noOfParameters],$3);
    PTL[noOfParameters]=flag;
    noOfParameters++;
}
                | parameter_list COMMA pointerAsAParameter{
                    struct SymbolTableEntry*new_entry =
insertIdentifier(ptrBuf,CLASS[3],$3,pointerLen,yylinenumber,scope+1);
                    PD[noOfParameters]=pointerLen;
                    strcpy(PL[noOfParameters],$3);
                    PTL[noOfParameters]='p';
                    noOfParameters++;
                }
                | pointerAsAParameter{
```

```
                    struct SymbolTableEntry*new_entry =
insertIdentifier(ptrBuf,CLASS[3],$1,pointerLen,yylinenumber,scope+1);
                    PD[noOfParameters]=pointerLen;
                    strcpy(PL[noOfParameters],$1);
                    PTL[noOfParameters]='p';
                    noOfParameters++;
                }
                | type ID choice {
                    struct SymbolTableEntry * search=lookup($2,scope+1,true);
                    if(search!=NULL){
                        char message[30]="Variable Already Declared";
                        int res=yyerror(message);
                        return -1;
                    }
                    else if(strcmp($3,"variable")==0){
                        struct SymbolTableEntry*new_entry =
insertIdentifier($2,$3,$1,0,yylinenumber,scope+1);
                    }
                    else if(strcmp($3,"array")==0){
                        struct SymbolTableEntry*new_entry =
insertArray($2,$3,$1,dimit,yylinenumber,scope+1,dimbuffer);
                    }
                    PD[noOfParameters]=dimit;
                    dimit=0;
                    strcpy(PL[noOfParameters],$1);
                    if(strcmp($3,"array")==0)PTL[noOfParameters]='a';
                    else if(strcmp($3,"variable")==0) PTL[noOfParameters]='v';
                    noOfParameters++;
                }
choice : arrayAsAParameter {
    $$=malloc(strlen("array")+1);strcpy($$,"array");

    } | {$$=malloc(strlen("variable")+1);strcpy($$,"variable");}


function_call : funName OPEN_BRACK params CLOSE_BRACK{
    int nop=forFunc->noOfParams;
    if(noOfParameters!=nop){
        char message[50]="No of Parameters of the Function Does not match";
        int res=yyerror(message);
        return -1;
    }

    for(int i=0;i<nop;i++){
        if(strcmp(PL[i],forFunc->paraList[i])!=0){
            char message[50]="DataTypes of Parameters did not match";
            int res=yyerror(message);
            return -1;
        }
    }

    for(int i=0;i<nop;i++){
        if((PTL[i]=='a' && forFunc->paraTypes[i]=='p') || (PTL[i]=='p' && forFunc-
>paraTypes[i]=='a'))continue;
```

```
        if(PTL[i]!=forFunc->paraTypes[i]){
            char message[40]="Type(Class) of Parameters did not match";
            int res=yyerror(message);
            return -1;
        }
    }

    for(int i=0;i<nop;i++){
        if(parD[i]!=forFunc->paraDim[i]){
            char message[40]="Dimension does not match";
            int res=yyerror(message);
            return -1;
        }
    }
    parDi=0;
    noOfParameters=0;
} SEMICOLON start1
                | funName OPEN_BRACK CLOSE_BRACK{
                    int nop=forFunc->noOfParams;

                    if(noOfParameters!=nop){
                        char message[50]="No of Parameters of the Function Does not
match";
                        int res=yyerror(message);
                        return -1;
                    }
                    char temp[20];
                    char t;
                    for(int i=0;i<nop/2;i++){
                        strcpy(temp,PL[i]);
                        strcpy(PL[i],PL[nop-i-1]);
                        strcpy(PL[nop-i-1],temp);
                        t=PTL[i];
                        PTL[i]=PTL[nop-i-1];
                        PTL[nop-i-1]=t;
                    }

                    for(int i=0;i<nop;i++){
                        if(strcmp(PTL[i],forFunc->paraList[i])!=0){
                            char message[40]="DataTypes of Parameters did not match";
                            int res=yyerror(message);
                            return -1;
                        }
                    }

                    for(int i=0;i<nop;i++){
                        if((PTL[i]=='a' && forFunc->paraTypes[i]=='p') || (PTL[i]=='p' &&
forFunc->paraTypes[i]=='a'))continue;
                        if(PTL[i]!=forFunc->paraTypes[i]){

                            char message[40]="Type(Class) of Parameters did not match";
                            int res=yyerror(message);
                            return -1;
                        }
```

```
                    }
                    for(int i=0;i<nop;i++){
                        if(parD[i]!=forFunc->paraDim[i]){
                            char message[40]="Dimension does not match";
                            int res=yyerror(message);
                            return -1;
                        }
                    }

                    parDi=0;
                    noOfParameters=0;
                } SEMICOLON start1

function_call1 : funName OPEN_BRACK params CLOSE_BRACK{
    int nop=forFunc->noOfParams;
    if(noOfParameters!=nop){
        char message[50]="No of Parameters of the Function Does not match";
        int res=yyerror(message);
        return -1;
    }

    for(int i=0;i<nop;i++){
        if(strcmp(PL[i],forFunc->paraList[i])!=0){
            char message[50]="DataTypes of Parameters did not match";
            int res=yyerror(message);
            return -1;
        }
    }

    for(int i=0;i<nop;i++){
        if((PTL[i]=='a' && forFunc->paraTypes[i]=='p') || (PTL[i]=='p' && forFunc-
>paraTypes[i]=='a'))continue;
        if(PTL[i]!=forFunc->paraTypes[i]){
            char message[40]="Type(Class) of Parameters did not match";
            int res=yyerror(message);
            return -1;
        }
    }

    for(int i=0;i<nop;i++){
        if(parD[i]!=forFunc->paraDim[i]){
            char message[40]="Dimension does not match";
            int res=yyerror(message);
            return -1;
        }
    }
    parDi=0;
    noOfParameters=0;
}  start1
                | funName OPEN_BRACK CLOSE_BRACK{
                    int nop=forFunc->noOfParams;

                    if(noOfParameters!=nop){
```

```
                        char message[50]="No of Parameters of the Function Does not
match";
                        int res=yyerror(message);
                        return -1;
                    }
                    char temp[20];
                    char t;
                    for(int i=0;i<nop/2;i++){
                        strcpy(temp,PL[i]);
                        strcpy(PL[i],PL[nop-i-1]);
                        strcpy(PL[nop-i-1],temp);
                        t=PTL[i];
                        PTL[i]=PTL[nop-i-1];
                        PTL[nop-i-1]=t;
                    }

                    for(int i=0;i<nop;i++){
                        if(strcmp(PTL[i],forFunc->paraList[i])!=0){
                            char message[40]="DataTypes of Parameters did not match";
                            int res=yyerror(message);
                            return -1;
                        }
                    }

                    for(int i=0;i<nop;i++){
                        if((PTL[i]=='a' && forFunc->paraTypes[i]=='p') || (PTL[i]=='p' &&
forFunc->paraTypes[i]=='a'))continue;
                        if(PTL[i]!=forFunc->paraTypes[i]){
                            char message[40]="Type(Class) of Parameters did not match";
                            int res=yyerror(message);
                            return -1;
                        }
                    }
                    for(int i=0;i<nop;i++){
                        if(parD[i]!=forFunc->paraDim[i]){
                            char message[40]="Dimension does not match";
                            int res=yyerror(message);
                            return -1;
                        }
                    }

                    parDi=0;
                    noOfParameters=0;
                }  start1
funName: ID{
    forFunc = lookup($1,1,true);
    if(forFunc==NULL){
        char message[40]="Function is not Declared/Defined";
        int res=yyerror(message);
        return -1;
    }
    else if(strcmp(forFunc->type,CLASS[1])!=0){
        char message[40]="Function is not Declared/Defined";
        int res=yyerror(message);
```

```
        return -1;
    }
    else{
        //Update line number
        update(forFunc,yylinenumber);
    }
}
params : item {
    strcpy(PL[noOfParameters],$1);
    PTL[noOfParameters]=pt;
    noOfParameters++;
}
        | params COMMA item {
            strcpy(PL[noOfParameters],$3);
            PTL[noOfParameters]=pt;
            noOfParameters++;
        }
item : ID {
    struct SymbolTableEntry * entry = lookup($1,scope,false);
    if(entry==NULL){
        char message[40]="Variable is not Declared";
        int res=yyerror(message);
        return -1;
    }
    if(strcmp(entry->type,CLASS[0])==0){
        pt='v';
        parD[parDi++]=entry->dimension;
        update(entry,yylinenumber);
    }
    else if(strcmp(entry->type,CLASS[3])==0){
        pt='p';
        parD[parDi++]=entry->dimension;
        update(entry,yylinenumber);
    }
    else if(strcmp(entry->type,CLASS[1])==0){
        char message[40]="Variable not declared";
        int res=yyerror(message);
        return -1;
    }
    else if(strcmp(entry->type,CLASS[2])==0){
        pt='a';
        parD[parDi++]=entry->dimension;
        update(entry,yylinenumber);
    }
    $$=malloc(strlen(entry->dataType)+1);strcpy($$,entry->dataType);}
    | INTEGER {$$=malloc(strlen($1)+1);strcpy($$,$1);pt='v';parD[parDi++]=0;}
    | STRING {$$=malloc(strlen($1)+1);strcpy($$,$1);pt='v';parD[parDi++]=0;}
    | CHAR_CONST {$$=malloc(strlen($1)+1);strcpy($$,$1);pt='v';parD[parDi++]=0;}
    | FLOATING_NUM {$$=malloc(strlen($1)+1);strcpy($$,$1);pt='v';parD[parDi++]=0;}
    | NULLT

fun_start : FUN_START {
    char Type[10];
    int Index=0;
```

```
    int i=0;
    int yl=strlen($1);
    for(;i<yl;i++){
        if($1[i]!=' '){
            Type[Index++]=$1[i];
        }
        else{
            Type[Index++]='\0';
            break;
        }
    }
    strcpy(funTypeBuffer,Type);
    while($1[i]==' '){
        i++;
    }
    char iden[30];
    Index=0;
    for(;i<yl;i++){
        if($1[i]!='(' && $1[i]!=' '){
            iden[Index++]=$1[i];
        }
        else{
            iden[Index++]='\0';
            break;
        }
    }
    strcpy(funNameBuffer,iden);
}


for : forScope OPEN_BRACK varDecF ExpF SEMICOLON ExpF CLOSE_BRACK OPEN_FLOWER start1
close_flower start1
        | forScope OPEN_BRACK varDecF ExpF SEMICOLON ExpF CLOSE_BRACK SEMICOLON start1
        | forScope OPEN_BRACK SEMICOLON ExpF SEMICOLON ExpF CLOSE_BRACK open_flower start1
close_flower start1
        | forScope OPEN_BRACK SEMICOLON ExpF SEMICOLON ExpF CLOSE_BRACK SEMICOLON start1

forScope : FOR {scope++;}

while : WHILE OPEN_BRACK ExpF CLOSE_BRACK open_flower start1 close_flower start1
        | WHILE OPEN_BRACK ExpF CLOSE_BRACK SEMICOLON start1

varDecF : type idFor options D SEMICOLON
         | idFor options D SEMICOLON

ExpF : Exp |

idFor:ID {
    struct SymbolTableEntry * search=lookup($1,scope,false);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
```

```
    else{
        struct
SymbolTableEntry*new_entry=insertIdentifier($1,CLASS[0],typeBuffer,0,yylinenumber,scope);
    }
}

print : PRINTF OPEN_BRACK printExpr CLOSE_BRACK SEMICOLON start1
printExpr : STRING
       | STRING printArguments
printArguments : COMMA printContent printArguments
               | COMMA printContent
printContent : Exp | arrayElement

scanf : SCANF OPEN_BRACK scanfExpr CLOSE_BRACK SEMICOLON start1
scanfExpr : STRING scanfArguments
scanfArguments : COMMA AMPERSAND ID scanfArguments
            | COMMA AMPERSAND ID
            | COMMA scanfContent scanfArguments
            | COMMA scanfContent
scanfContent : ID | arrayElement

arrayElement : ID dimension
dimension : open_sq Exp CLOSE_SQ dimension | open_sq Exp CLOSE_SQ

open_sq : OPEN_SQ{ID_dim++;}

ID2 : ID {
        struct SymbolTableEntry * found=lookup($1,scope,true);
        if(found != NULL)
        {
            char const errorMessage[100]="Variable Already Declared!!";
            int a=yyerror(errorMessage);
            return -1;
        }

        strcpy(IDBuffer,$1);

        }

arrayDeclr : type ID2
    BOX SEMICOLON {
        for(int i=0;i<dimit;i++){
            if(dimbuffer[i]<=0){
                char message[25]="Access Denied";
                int res=yyerror(message);
                return -1;
            }
        }
    struct SymbolTableEntry* temp =
insertArray(IDBuffer,CLASS[2],typeBuffer,dimit,yylinenumber,scope,dimbuffer);dimit = 0;}
BOX : BOX open_sq integer_dim CLOSE_SQ | open_sq integer_dim CLOSE_SQ

integer_dim : INTEGER {dimbuffer[dimit++] = intval;} | ID {dimbuffer[dimit++] = 10000;}
```

```
arrayInitial : type ID2 BOX EQUALTO open_flower BALANCED_BRACK close_flower SEMICOLON |
type ID2 BOX EQUALTO open_flower close_flower SEMICOLON
BALANCED_BRACK : arrayParams_unend
            | arrayParams_unend COMMA
            | arrayParams_unend COMMA open_flower BALANCED_BRACK close_flower COMMA
BALANCED_BRACK
            | arrayParams_unend COMMA open_flower BALANCED_BRACK close_flower COMMA
            | arrayParams_unend COMMA open_flower BALANCED_BRACK close_flower
            | open_flower BALANCED_BRACK close_flower
            | open_flower BALANCED_BRACK close_flower COMMA
            | open_flower BALANCED_BRACK close_flower COMMA BALANCED_BRACK
arrayParams_unend : INTEGER | arrayParams_unend COMMA INTEGER

arrayAsAParameter : OPEN_SQ integer_dim CLOSE_SQ higherDimention
higherDimention : BOX |

pointerAsAParameter : PTR_TYPE PTR_STAR ID{
    struct SymbolTableEntry * found=lookup($3,scope+1,true);
    if(found != NULL)
    {
        char const errorMessage[100]="Variable Already Declared!!";
        int a=yyerror(errorMessage);
        return -1;
    }
    else{
        strcpy(ptrBuf,$3);
        $$=malloc(strlen($1)+1);
        strcpy($$,$1);
    }
}

PTR_DECLR : PTR_TYPE PTR_STAR idinsert SEMICOLON
PTR_INITIAL : PTR_TYPE PTR_STAR idinsert EQUALTO AMPERSAND idlook {if(strcmp($3,$6)!=0){
    char message[30]="Variable Types dont Match";
        int res=yyerror(message);
        return -1;}}SEMICOLON | PTR_TYPE PTR_STAR idinsert EQUALTO idlook
{if(strcmp($3,$5)!=0){
    char message[30]="Variable Types dont Match";
        int res=yyerror(message);
        return -1;
}}SEMICOLON | PTR_TYPE PTR_STAR idinsert EQUALTO PTR_EXP SEMICOLON

idinsert : ID{

    struct SymbolTableEntry * search=lookup($1,scope,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
        $$=typeBuffer;
```

```
        struct
SymbolTableEntry*new_entry=insertIdentifier($1,CLASS[3],typeBuffer,pointerLen,yylinenumber
,scope);
        pointerLen = 1;
    }
}
idlook : ID{
        struct SymbolTableEntry * found=lookup($1,scope,false);
        if(found==NULL){
            char const errorMessage[100]="Variable Not Declared!!";
            int a=yyerror(errorMessage);
            return -1;
        }
        else{
            $$ = found->dataType;
            update(found,yylinenumber);
        }
     }
PTR_STAR : PTR_STAR MUL {
     pointerLen++;
} | MUL
PTR_TYPE : userDefDataType ID | type | VOID
PTR_EXP : OPEN_BRACK PTR_TYPE PTR_STAR CLOSE_BRACK MALLOC OPEN_BRACK SIZEOF OPEN_BRACK
PTR_TYPE PTR_STAR CLOSE_BRACK CLOSE_BRACK
        | OPEN_BRACK PTR_TYPE PTR_STAR CLOSE_BRACK MALLOC OPEN_BRACK SIZEOF OPEN_BRACK
PTR_TYPE CLOSE_BRACK CLOSE_BRACK

switch : SWITCH OPEN_BRACK ID CLOSE_BRACK open_flower switchcase default close_flower
start1
switchcase : CASE comp COLON start1 switchcase |
default : DEFAULT COLON start1 |
comp : CHAR_CONST | INTEGER

ifElseLadder: S
S: matched | unmatched
matched: IF OPEN_BRACK Exp CLOSE_BRACK open_flower start1 close_flower elif ELSE
open_flower start1 close_flower
elif : ELIF OPEN_BRACK Exp CLOSE_BRACK open_flower start1 close_flower elif |
unmatched: IF OPEN_BRACK Exp CLOSE_BRACK open_flower start1 close_flower elif

userTypeDefination : TYPEDEF userDefDataType sidi open_flower userTypeParams close_flower{
        struct
SymbolTableEntry*new_entry=insertIdentifier(userTypeNameBuffer,CLASS[4],userTypeBuffer,use
rTypeDim,yylinenumber,scope);
        userTypeDim = 0;
    } userTypeObj SEMICOLON startGlobal
                | userDefDataType sidi open_flower userTypeParams close_flower
                {
        struct
SymbolTableEntry*new_entry=insertIdentifier(userTypeNameBuffer,CLASS[4],userTypeBuffer,use
rTypeDim,yylinenumber,scope);
        userTypeDim = 0;
                }
                userTypeObj SEMICOLON startGlobal
```

```
userTypeParams : varInStruct SEMICOLON userTypeParams
               | pointerInStruct SEMICOLON userTypeParams
               | structInStruct SEMICOLON userTypeParams
               | varInStruct SEMICOLON
               | pointerInStruct SEMICOLON
               | structInStruct SEMICOLON

               //do from here
userTypeObj : uIDi | userTypeObj COMMA uIDi |

uIDi: ID {
    struct SymbolTableEntry * search=lookup($1,1,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
        struct
SymbolTableEntry*new_entry=insertIdentifier($1,userTypeBuffer,userTypeNameBuffer,0,yylinen
umber,1);
    }
}


userTypeDeclaration : structInStruct SEMICOLON | ID ID SEMICOLON
userTypeInitialization : structInStruct EQUALTO open_flower params close_flower SEMICOLON
                       | ID DOT ID {
                           struct SymbolTableEntry * search=lookup($1,1,true);

                           if(search==NULL){
                               char message[30]="Variable Not Declared";
                               int res=yyerror(message);
                               return -1;
                           }else {
                               struct SymbolTableEntry * search1=lookup($3,1,true);
                               if(search1 == NULL){
                                   char message[50]="Variable Not Part of the User
definied Data Type";
                                   int res=yyerror(message);
                               return -1;
                               }
                           if(strcmp(search->dataType,search1->dataType)!=0){
                                   char message[30]="Varible Type Mismatch!!";
                                   int res=yyerror(message);
                                   return -1;
                           }
                           }
                       } EQUALTO item SEMICOLON
                       | ID ARROW ID{
                           struct SymbolTableEntry * search=lookup($1,1,true);

                           if(search==NULL){
                               char message[30]="Variable Not Declared";
```

```
                                int res=yyerror(message);
                                return -1;
                            }else {
                                struct SymbolTableEntry * search1=lookup($3,1,true);
                                if(search1 == NULL){
                                    char message[50]="Variable Not Part of the User
definied Data Type";
                                    int res=yyerror(message);
                                return -1;
                                }
                            if(strcmp(search->dataType,search1->dataType)!=0){
                                    char message[30]="Varible Type Mismatch!!";
                                    int res=yyerror(message);
                                    return -1;
                            }
                            }
                        } EQUALTO item SEMICOLON

userDefDataType : STRUCT {
            strcpy(userTypeBuffer,$1);
            $$=malloc(strlen($1)+1);
            strcpy($$,$1);
        }
                | UNION {
            strcpy(userTypeBuffer,$1);
            $$=malloc(strlen($1)+1);
            strcpy($$,$1);
        }

sidi: ID {
    struct SymbolTableEntry * search=lookup($1,1,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
        strcpy(userTypeNameBuffer, $1);
    }
}
pointerInStruct: PTR_TYPE PTR_STAR ID{
    struct SymbolTableEntry * found=lookup($3,1,true);
    if(found != NULL)
    {
        char const errorMessage[100]="Variable Already Declared!!";
        int a=yyerror(errorMessage);
        return -1;
    }
    else{
        struct
SymbolTableEntry*new_entry=insertIdentifier($3,CLASS[3],userTypeNameBuffer,pointerLen,yyli
nenumber,1);
        pointerLen = 1;
    }
```

```
    userTypeDim++;
}

varInStruct: type ID higherDimention
{
    struct SymbolTableEntry * search=lookup($2,1,true);
    if(search!=NULL){
        char message[30]="Variable Already Declared";
        int res=yyerror(message);
        return -1;
    }
    else{
        if(dimit==0){
            struct
SymbolTableEntry*new_entry=insertIdentifier($2,CLASS[0],userTypeNameBuffer,dimit,yylinenum
ber,1);
        }else{
            struct
SymbolTableEntry*new_entry=insertIdentifier($2,CLASS[2],userTypeNameBuffer,dimit,yylinenum
ber,1);
        }

        userTypeDim++;
        dimit = 0;
    }
}

structInStruct: userDefDataType ID ID
    {
                    struct SymbolTableEntry * search=lookup($2,1,true);
                    if(search==NULL){
                        char message[30]="Variable Not Declared";
                        int res=yyerror(message);
                        return -1;
                    }else if(strcmp(search->dataType,$1)!=0){
                            char message[30]="Varible Type Mismatch!!";
                            int res=yyerror(message);
                            return -1;
                    }
                    else{struct SymbolTableEntry * search1=lookup($3,1,true);
                        if(search1!=NULL){
                        char message[30]="Variable Already Declared";
                        int res=yyerror(message);
                        return -1;
                        }else{
                            struct SymbolTableEntry*new_entry=insertIdentifier($3,search-
>dataType,$2,0,yylinenumber,1);
                        }
                    }
                    userTypeDim++;
                }

%%
```

```
int yyerror(const char *s){
    extern int yylinenumber;
    printf("Error : %s!! \nLine No : %d\n",s,yylinenumber);
}

int main(){
    for (int i=0; i<TABLE_SIZE;i++) {
        Table.symTable[i] = NULL;
    }
    yyin=fopen("input.txt","r");
    yyparse();
    printf("\n\nSYMBOL TABLE\n\n");
    printf("%-15s %-10s %-15s %-15s %-15s %-15s %-15s %-20s\n", "ID", "Type", "Data
Type","Dim","Scope","No of Params","Line of Decl","Line of Ref");
    printf("-------------------------------------------------------------------------------
-----------------------------------------------\n");

    for (int i = 0; i < stIterator; i++) {
        printf("%d ",i);
        struct SymbolTableEntry * entry = Table.symTable[i];
        printf("%-15s %-10s %-15s %-15d %-15d %-15d %-15d [ ", entry->name, entry->type,
entry->dataType,entry->dimension,entry->scope,entry->noOfParams,entry->lineOfDeclaration);
        struct lines* h = entry->head->next;
        while(h){
            printf("%d ", h->lineno);
            h = h->next;
        }
        printf("]\n");
    }
    printf("-------------------------------------------------------------------------------
-----------------------------------------------\n");
    printf("\n\n");
    fclose(yyin);
}
```

LEX File (.l)

```
%{
    #include <stdio.h>
    #include "y.tab.h"
    int yylinenumber=1;
    int intval = 0;
%}

%%
malloc {return MALLOC;}
sizeof {return SIZEOF;}
typedef {return TYPEDEF;}
else[ ]+if {return ELIF;}
NULL {return NULLT;}
```

```
if {return IF;}
else {return ELSE;}
break {return BREAK;}
continue {return CONTINUE;}
default {return DEFAULT;}
void {return VOID;}
for {return FOR;}
while {return WHILE;}
(int|void)[ ]+main[ ]*\( {return MAIN;}
(int|float|char|double|long|unsigned|void)[ ]+[a-zA-Z_][a-zA-Z0-9_]{0,30}[ ]*\( {
    yylval.value=strdup(yytext);
    return FUN_START;}
int|float|char|double|long|unsigned {yylval.type=strdup(yytext);return TYPE;}
switch {return SWITCH;}
case {return CASE;}
struct {yylval.type=strdup("struct");return STRUCT;}
union {yylval.type=strdup("union");return UNION;}
return {return RETURN;}
printf {return PRINTF;}
scanf {return SCANF;}
\#[ \t]*include {return INCLUDE;}
\<[a-zA-Z]+\.h\> {return PREDEF_HEADER;}
[a-zA-Z_][a-zA-Z0-9_]{0,30} {yylval.value=strdup(yytext);return ID;}
-?[1-9][0-9]{0,8}|-?1[0-9]{0,9}|-?2[0-1][0-4][0-7][0-4][0-8][0-3][0-6][0-4][0-7]|0 {
    int p = 1;
    intval = 0;
    for(int i =yyleng - 1;i>=0;i--)
    {
        intval += (yytext[i] - '0') * p;
        p*=10;
    }
    yylval.type=strdup("int");return INTEGER;
    }
[-]?[0-9]+\.[0-9]{1,6} {yylval.type=strdup("float");return FLOATING_NUM;}
\/\/[^\n]* {return SL_COMMENT;}
\/\*[^*/]*\*\/ {return ML_COMMENT;}
\"[^\n]*\" {yylval.type=strdup("string");return STRING;}
['].['] {yylval.type=strdup("char");return CHAR_CONST;}
\-\> {return ARROW;}
\. {return DOT;}
\= {return EQUALTO;}
\( {return OPEN_BRACK;}
\) {return CLOSE_BRACK;}
\{ {return OPEN_FLOWER;}
\} {return CLOSE_FLOWER;}
\[ {return OPEN_SQ;}
\] {return CLOSE_SQ;}
\& {return AMPERSAND;}
\+\+|\-\- {return UNARY_OP;}
\+ {return PLUS;}
\- {return MINUS;}
\* {return MUL;}
\/ {return DIV;}
\% {return MOD;}
```

```
\|\| {return OR;}
\&\& {return AND;}
\! {return NOT;}
\| {return BIT_OR;}
\^ {return BIT_XOR;}
\=\= {return ISEQUALTO;}
\<\= {return LTE;}
\>\= {return GTE;}
\!\= {return NE;}
\< {return LT;}
\> {return GT;}
\|\= {return OR_ET;}
\&\= {return AND_ET;}
\^\= {return XOR_ET;}
\+\= {return PLUS_ET;}
\-\= {return MINUS_ET;}
\*\= {return MUL_ET;}
\/\= {return DIV_ET;}
; {return SEMICOLON;}
, {return COMMA;}
: {return COLON;}
\n {++yylinenumber;};
[ \t]+ ;
. {return *yytext;}
%%

int yywrap(){
    return 1;
}
```

INPUT TEST CASES

Test Case 1: (No error)

input.txt

```
1   #include<stdio.h>
2
3   int onesdigit(int a){
4       return a%10;
5   }
6   void main(){
7       int x;
8       x = onesdigit(x);
9   }
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Syntax is Correct


SYMBOL TABLE

ID             Type       Data Type       Dim            Scope          No of Params    Line of Decl    Line of Ref
--------------------------------------------------------------------------------------------------------------------
0 a              variable   int             0              2              -1              3               [ 4 ]
1 onesdigit      function   int             0              1              1               3               [ 8 ]
2 x              variable   int             0              2              -1              7               [ 8 8 ]
--------------------------------------------------------------------------------------------------------------------
```

TEST CASE 2(error)

```c
#include<stdio.h>

int onesdigit(int a){
    return a%10;
}
void main(){
    int y;
    while(x<10){
        y = onesdigit(x);
    }
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Error : Variable Not Declared!!!!
Line No : 8


SYMBOL TABLE

ID             Type       Data Type       Dim            Scope          No of Params    Line of Decl    Line of Ref
--------------------------------------------------------------------------------------------------------------------
0 a              variable   int             0              2              -1              3               [ 4 ]
1 onesdigit      function   int             0              1              1               3               [ ]
2 y              variable   int             0              2              -1              7               [ ]
--------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: Variable ‘x’ not declared

TEST CASE 3:

```c
#include<stdio.h>

int x;

int square(int a){
    return a*a;
}
void main(){
    int i = 0;
    while(i<10){
        int x;
        int j = 0;
        while(j<10){
            x = square(i);
        }
        j = square(x);
    }
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Syntax is Correct


SYMBOL TABLE

ID             Type       Data Type       Dim            Scope          No of Params    Line of Decl    Line of Ref
-------------------------------------------------------------------------------------------------------------------
0 x              variable   int            0              1              -1              3               [ ]
1 a              variable   int            0              2              -1              5               [ 6 6 ]
2 square         function   int            0              1              1               5               [ 14 16 ]
3 i              variable   int            0              2              -1              9               [ 10 14 ]
4 x              variable   int            0              3              -1              11              [ 14 16 ]
5 j              variable   int            0              3              -1              12              [ 13 16 ]
-------------------------------------------------------------------------------------------------------------------
```

TEST CASE 4: (error)

```c
#include <stdio.h>

int main() {

    int num1,num2;

    switch (operator) {
        case '+':
            result = num1 + num2;
            printf("%.2lf + %.2lf = %.2lf\n", num1, num2, result);
            break;
        case '-':
            result = num1 - num2;
            printf("%.2lf - %.2lf = %.2lf\n", num1, num2, result);
            break;
        case '*':
            result = num1 * num2;
            printf("%.2lf * %.2lf = %.2lf\n", num1, num2, result);
            // Missing break statement intentionally, leading to fall-through
        case '/':
            if (num2 != 0) {
                result = num1 / num2;
                printf("%.2lf / %.2lf = %.2lf\n", num1, num2, result);
            } else {
                printf("Error: Division by zero!\n");
            }
            break;
        default:
            printf("Error: Invalid operator\n");
            break;
    }

    return 0;
}
```

OUTPUT

```
SYMBOL TABLE

ID             Type      Data Type      Dim           Scope          No of Params    Line of Decl    Line of Ref
-------------------------------------------------------------------------------------------------------------------
0 num1          variable   int           0             2              -1              5               [ ]
1 num2l         variable   int           0             2              -1              5               [ ]
-------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: operator variable not declared

TEST CASE 5: (error)

```c
#include <stdio.h>

void fun(int a,char b){
    a = 10;
}

int main(){
    int c;
    char d;
    int e;
    fun(c,d,e);
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Error : No of Parameters of the Function Does not match!!
Line No : 11


SYMBOL TABLE

ID              Type        Data Type       Dim             Scope           No of Params    Line of Decl    Line of Ref
--------------------------------------------------------------------------------------------------------------------------
0 a             variable    int             0               2               -1              3               [ 4 ]
1 b             variable    char            0               2               -1              3               [ ]
2 fun           function    void            0               1               2               3               [ 11 ]
3 c             variable    int             0               2               -1              8               [ 11 ]
4 d             variable    char            0               2               -1              9               [ 11 ]
5 e             variable    int             0               2               -1              10              [ 11 ]
--------------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: Number of parameters don’t match

TEST CASE 6: (error)

```c
#include <stdio.h>

int main(){
    int arr[10][10];
    int i = 0;
    int j = 0;
    while(i<20){
        while(j<20){
            arr[i][j] = i+j;
            ++j;
        }
        ++i;
    }
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Error : syntax error!!
Line No : 9


SYMBOL TABLE

ID              Type        Data Type       Dim             Scope           No of Params    Line of Decl    Line of Ref
--------------------------------------------------------------------------------------------------------------------
0 arr             array       int             2               2               -1              4               [ 9 ]
1 i               variable    int             0               2               -1              5               [ 7 ]
2 j               variable    int             0               2               -1              6               [ 8 ]
--------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: Array size is 10 X 10. However, in the while loop, an attempt is made to access arr[11][11]

TEST CASE 7: (error)

```c
#include <stdio.h>

void fun(int a){
    a++;
}
int main(){
    int a = fun(a);
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Error : Void Function Does not return anything!!
Line No : 7


SYMBOL TABLE

ID              Type        Data Type       Dim             Scope           No of Params    Line of Decl    Line of Ref
--------------------------------------------------------------------------------------------------------------------
0 a               variable    int             0               2               -1              3               [ 4 ]
1 fun             function    void            0               1               1               3               [ 7 ]
2 a               variable    int             0               2               -1              7               [ 7 ]
--------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: void doesn't return anything

TEST CASE 8: (error)

```c
#include <stdio.h>

int main(){
    int arr[10][10];
    arr[1][2][3] = 1;
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Error : Dimension doesnt match!!!
Line No : 5


SYMBOL TABLE

ID             Type       Data Type       Dim            Scope          No of Params    Line of Decl    Line of Ref
------------------------------------------------------------------------------------------------------------------------
0 arr            array      int             2              2              -1              4               [ 5 ]
------------------------------------------------------------------------------------------------------------------------
```

EXPLANATION: dimensions don’t match

TEST CASE 9:

input.txt

```c
#include <stdio.h>

int main(){
    int i = 0;
    while(i<10){
        int i = 0;
        while(i<10){
            int i = 0;
            while(i<10){
                ++i;
            }
            ++i;
        }
        ++i;
    }
}
```

OUTPUT

```
shreekara@shreekara-HP-Pavilion-Gaming-Laptop-15-ec2xxx:~/phase3$ ./a.out input.txt
Syntax is Correct


SYMBOL TABLE

ID             Type       Data Type       Dim            Scope          No of Params    Line of Decl    Line of Ref
------------------------------------------------------------------------------------------------------------------------
0 i              variable   int             0              2              -1              4               [ 5 ]
1 i              variable   int             0              3              -1              6               [ 7 14 ]
2 i              variable   int             0              4              -1              8               [ 9 10 12 ]
------------------------------------------------------------------------------------------------------------------------
```

FUTURE WORK

In the realm of future developments, it's essential to note that while the yacc script presented in this report adeptly manages the rules governing the C language, it still falls short of being entirely comprehensive. Our

forthcoming efforts will be laser-focused on fortifying the script's capabilities to ensure it can effectively handle all intricacies of the C language, all the while striving to optimize its performance and efficiency.

CONCLUSION

In summary, the semantic analysis phase, showcased in this study, emerges as a vital component in the compiler's design. Beyond mere syntax checking, it ensures the program's logical coherence and adherence to language rules. By adeptly handling variable types and scrutinizing language-specific semantics, the semantic analyzer significantly contributes to the reliability of the compiled code. Its role extends beyond error detection, enhancing overall code quality and ultimately streamlining the compilation process for increased efficiency.

REFERENCES

• Compiler Principles, Techniques and Tool by Alfred V. Aho, Monica S. Lam,Ravi Sethi, Jefferey D. Ullman

• https://www.geeksforgeeks.org/introduction-to-yacc/