Compiler Design Lab

Semantic Analyser Report

Submitted By :
Shishir Gangwar (15CO147)
Saumyadip Mandal (15CO144)

Submitted To :
Ms.Susmitha,
Faculty, Dept. of Computer Science,
NITK, Surathkal.

Index

Introduction

A large part of semantic analysis consists of tracking variable/function/type declarations and type checking. In many languages, identifiers have to be declared before they're used. As the compiler encounters a new declaration, it records the type information assigned to that identifier. Then, as it continues examining the rest of the program, it verifies that the type of an identifier is respected in terms of the operations being performed. For example, the type of the right side expression of an assignment statement should match the type of the left side, and the left side needs to be a properly declared and assignable identifier. The parameters of a function should match the arguments of a function call in both number and type. The language may require that identifiers be unique, thereby forbidding two global declarations from sharing the same name. Arithmetic operands will need to be of numeric—perhaps even the exact same type (no automatic int to double conversion, for instance).

Semantics of a language provide meaning to its constructs, like tokens and syntax structure. Semantics help interpret symbols, their types, and their relations with each other. Semantic analysis judges whether the syntax structure constructed in the source program derives any meaning or not.

CFG + semantic rules = Syntax Directed Definitions

Role Of Semantic Analyser

1) Compilers use semantic analysis to enforce the static semantic rules of a language.

2) It is hard to generalize the exact boundaries between semantic

analysis and the generation of intermediate representations (or even just straight to final representations); this demarcation is the logical boundary between the front-end of a compiler (lexical analysis and parsing) and the back-end of the compiler (intermediate representations and final code).

3) For instance, a completely separated compiler could have a well-defined lexical analysis and parsing stage generating a parse tree, which is passed wholesale to a semantic analyzer, which could then create a syntax tree and populate a symbol table, and then pass it all on to a code generator.

4) Or a completely interleaved compiler could intermix all of these stages, literally generating final code as part of the parsing engine.

Attribute Grammar

Attribute grammar is a special form of context-free grammar where some additional information (attributes) are appended to one or more of its non-terminals in order to provide context-sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions. Attribute grammar is a medium to provide semantics to the context-free grammar and it can help specify the syntax and semantics of a programming language. Attribute grammar (when viewed as a parse-tree) can pass values or information among the nodes of a tree.

Eg : $E \rightarrow E1 + T \{ E.value = E1.value + T.value \}$

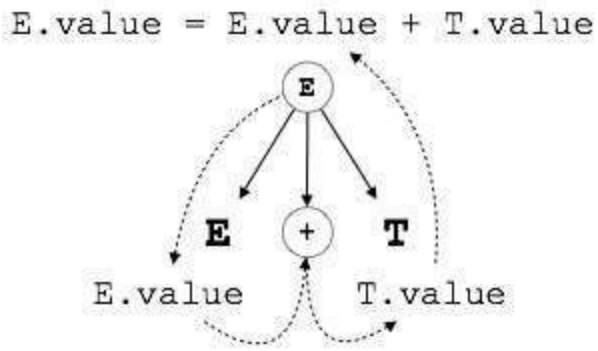
Synthesized attributes

These attributes get values from the attribute values of their child nodes. To illustrate, assume the following production:
S ---> XYZ

X can get values from Y,Z and S. B can take values from S, X and Y.

S-attributed SDT

If an SDT uses only synthesized attributes, it is called as S-attributed SDT. These attributes are evaluated using S-attributed SDTs that have their semantic actions written after the production (right hand side).S-attributed SDTs are evaluated in bottom-up parsing, as the values of the parent nodes depend upon the values of the child nodes.



L-attributed SDT

This form of SDT uses both synthesized and inherited attributes with restriction of not taking values from right siblings.In L-attributed SDTs, a non-terminal can get values from its parent, child, and sibling nodes.

For eg: S→ ABC

S can take values from A, B, and C (synthesized). A can take values from S only. B can take values from S and A. C can get values from S, A, and B. No non-terminal can get values from the sibling to its right.

Semantic Errors

1) Accessing an out of scope variable
2) Actual and formal parameter mismatch
3) Undeclared variable
4) Type mismatch
5) Multiple declaration of variable in a scope
6) Reserved identifier misuse
7) Index out of bound

Lex Code

```
%{
    #include<bits/stdc++.h>
    vector<string>lines;
    using namespace std;
    int charcount=0;
    string str;
    string func;
    string fname;
    vector<string>fargs;
     int lineno=1;
%}
identifier      [a-zA-Z][_a-zA-Z0-9]*
header          "#include"[ ]*["<""/""]{identifier}".h"?[">""/""]
```

```
type
"const"|"short"|"signed"|"unsigned"|"int"|"float"|"long"|"double"|"ch
ar"|"void"
storage         "auto"|"extern"|"register"|"static"|"typedef"
qualifier  "const"|"volatile"
digits          [0-9]+
decimal    0|[1-9][0-9]*
lint       {decimal}"L"
llint      {decimal}"LL"
double          {decimal}?"."{digits}
float           {double}"f"
scientific   {double}"e"{decimal}
scientificf  {scientific}"f"
str_literal  [a-zA-Z_]?\"(\\.|[^\\"])*"\""
character  "'"."'"
space      " "
tab              \t
next_line  \n
s_operator      "["|"]"|","|":"|"{"|"}"|"("|")"|"="|[-+*%/<>&|^!]

%x mlcomment
%x slcomment
%%
"/*"            {charcount+=yyleng;BEGIN(mlcomment);}
<mlcomment>[^*\n]*   {          charcount+=yyleng;;}
<mlcomment>\n         {    charcount=0;++lineno;}
<mlcomment>"*"+[^/]   {          charcount+=yyleng;;}
<mlcomment>"*"+"/"   { charcount+=yyleng;BEGIN(INITIAL);}

"//"            {charcount+=yyleng;BEGIN(slcomment);}
<slcomment>[^\n]*     {charcount+=yyleng;        ;}
<slcomment>\n         {charcount+=yyleng; BEGIN(INITIAL);}

{tab}                       {charcount+=1;}
{space}               {charcount+=1;}
{next_line}           {charcount=0;lineno++;}
{header}              {charcount+=yyleng;;}
{type}                {charcount+=yyleng;yylval.s =
```

```
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);str =
yytext;return TYPE;}
{storage}            {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
STORAGE;}
{qualifier}          {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
QUALIFIER;}
printf               {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PRINTF;}
scanf                {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SCANF;}
while                {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return WHILE;}
if                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return IF;}
else                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ELSE;}
return               {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return RETURN;}
do                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return DO;}
continue             {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
CONTINUE;}
break                {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return BREAK;}
goto                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return GOTO;}
default              {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
DEFAULT;}
enum                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ENUM;}
case                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return CASE;}
switch               {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SWITCH;}
```

```
for                     {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return FOR;}
sizeof                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return SIZEOF;}
struct                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STRUCT;}
union                   {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return UNION;}
{decimal}               {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{float}                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{double}                {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{llint}                 {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{lint}                  {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{scientific}            {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{scientificf}           {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NUM;}
{character}             {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STR;}
{str_literal}           {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return STR;}
{identifier}            {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ID;}
"+="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PE;}
"-="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return MI;}
"*="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return ME;}
"/="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return DE;}
">>"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return RS;}
```

```
"<<"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LS;}
"++"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return PP;}
"--"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return MM;}
"=="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return EE;}
"!="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return NE;}
">="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return GE;}
"<="                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LE;}
"||"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LO;}
"&&"                    {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return LA;}

{s_operator}            {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
yytext[0];}
";"                     {charcount+=yyleng;yylval.s =
(char*)malloc(sizeof(yytext));strcpy(yylval.s,yytext);return
yytext[0];}
.                       {charcount+=yyleng;printf("error at line no
%d",lineno,"NA");}
%%
int yywrap(){
return 1;}
```

Yacc Code

```
%{
      #include <bits/stdc++.h>
      #include "symbol.cpp"
```

```
      using namespace std;
      int yylex(void);
    void yyerror(char *);
    void trim(string& s)
      {    size_t p = s.find_first_not_of(" \t");
       s.erase(0, p);
            p = s.find_last_not_of(" \t");
       if (string::npos != p)
      s.erase(p+1);
       }
       int first,chc,lno;string typ,dtyp;
    string filename;
    extern string fname;
    extern vector<string>fargs;
    extern int charcount;
    extern string str;
    extern string func;
      extern int lineno;
      int ff=0,fflag=0,dd=0;
%}
%union{
    char *s;
}
%token NUM INT LONG VOID ID
%token STR STORAGE QUALIFIER TYPE
%token FLOAT DOUBLE BOOL CHAR
%token IF ELSE WHILE RETURN SCANF
%token GE ">=" LE "<=" EE "==" NE "!=" LO "||" LA "&&" PP "++" MM
"--"
%token PE "+=" MI "-=" ME "*=" DE "/=" LS "<<" RS ">>"
%token PRINTF GOTO CONTINUE DEFAULT BREAK ENUM DO
%token CASE SWITCH FOR SIZEOF STRUCT UNION
%right '='
%right "+=" "-=" "*=" "/="
%left "||"
%left "&&"
%left '|'
%left '&'
```

```
%left "==" "!="
%left '<' ">=" '>' "<="
%left "<<" ">>"
%left '+' '-'
%left '*' '/' '%'
%right '!'
%left '(' ')' '[' ']' "++" "--"
%%
start:          function {temp = head; func = "";} start
|               declaration {temp = head;} start
|
;
    function:         TYPE ID '(' {
insVar("function",$<s>1,$<s>2,lineno,charcount,true);lno=lineno;func
= $<s>2;chc=charcount;typ=$<s>2;dtyp=$<s>1;} declordef
                            {if(fflag)
search(head,"function",typ,dtyp,chc);ff=0;fflag=0;}
    ;
      declordef:      paramdecls ')' compstmt
{temp->func_def_flag=0;}
      |               fparamdecls ')' ';'
{temp->func_def_flag=1;}
      |               paramdecls ')' ';'    {temp->func_def_flag=1;}
      ;
      paramdecls:     paramdecl
      |
      ;
           paramdecl:     param ',' paramdecl
           |              param
           ;
                param:    TYPE ID         {
if(temp->func_def_flag==0)insVar("argument",$<s>1,$<s>2,lineno,charco
unt);
                                                                  else{

if(!fflag && ff<temp->token.size() && temp->datatype[ff]==$<s>1){

      temp->type[ff] = $<s>2;temp->position[ff]=charcount;++ff;
```

```
                                                                    }

else{

      fflag=1;
                                                                    }
                                                                }
                                                            }
                ;
      fparamdecls:    fparamdecl
      ;
            fparamdecl:    fparam ',' fparamdecl
            |              fparam
            |                fparam ',' paramdecl
            |                param ',' fparamdecl
            ;
                  fparam:   TYPE     {
insVar("argument",$<s>1,"---",lineno,charcount);}
                  ;
    declaration:      declstmt ';'
    |                 expr ';'
    |                 functcall ';'
    ;


      declstmt:       TYPE decllist
      ;
            decllist:       ID assign {
insVar("variable",str,$<s>1,lineno,charcount);} ',' decllist
            |               ID assign {
insVar("variable",str,$<s>1,lineno,charcount);}
            |               ID arraydecl ',' decllist    {
insVar("variable",str,$<s>1,lineno,charcount);checkDim();cout << dd
<< "sf";fargs.clear();dd=0;}
            |               ID arraydecl     {
insVar("variable",str,$<s>1,lineno,charcount);checkDim();cout << dd
<< "sf";fargs.clear();dd=0;}
            |               ID arraydecl'=' '{' arrassign '}' {
```

```
insVar("variable",str,$<s>1,lineno,charcount);checkDim();fargs.clear(
);dd=0;}
            ;
                 arraydecl:     '[' const ']' {++dd;} arraydecl

                 |               '[' const ']'
{++dd;}
                 ;
                 arrassign:      NUM ',' arrassign
                 |              NUM
                 ;
                 assign:        '=' expr
                 |
                 ;
stmt:           compstmt
|               forstmt
|               whilestmt
|               dowhilestmt
|               ifstmt
|               returnstmt
|               declaration
|               ';'
;
    compstmt:         '{' { if(temp!=head && temp->first!=1)
insVar("_block_","null","    null",lineno,charcount,true); else
if(temp!=head) ++temp->first;} stmtlist '}' {if(temp!=head)temp =
temp->parent;}
    ;
      stmtlist:      stmt stmtlist
      |
      ;
    forstmt:    FOR '(' fexp ';' fexp ';' fexp ')' stmt
    ;
      fexp:          expr
      |
      ;
    whilestmt:       WHILE '(' expr ')' stmt
    ;
```

```
    dowhilestmt:      DO compstmt WHILE '(' expr ')' ';'
    ;
    ifstmt:           IF '(' expr ')' stmt
    ;
    returnstmt:       RETURN ';'     {checkReturn(1);}
    |                 RETURN expr ';'{checkReturn(0);}
    ;

functcall:       ID  '(' argdecls ')'   {fname=$<s>1;paramcheck();}
;
    argdecls:     argdecl
    |
    ;
       argdecl:         const ',' argdecl
       |                const
       ;
            const:           NUM  {fargs.push_back($<s>1);}
            |                     ID   {fargs.push_back($<s>1);}
            ;
expr :           '(' expr ')'
|                ID "+=" expr     {scope(temp,$<s>1,charcount);}| ID
"+=" error
|                ID "-=" expr     {scope(temp,$<s>1,charcount);}| ID
"-=" error
|                ID "*=" expr     {scope(temp,$<s>1,charcount);}| ID
"*=" error
|                ID "/=" expr     {scope(temp,$<s>1,charcount);}| ID
"/=" error
|                ID '=' expr      {scope(temp,$<s>1,charcount);}| ID
'=' error
|                expr '+' expr     {}| expr '+' error
|                expr '-' expr     {}| expr '-' error
|                expr '*' expr     {}| expr '*' error
|                expr '/' expr     {}| expr '/' error
|                expr '<' expr     {}| expr '<' error
|                expr '>' expr     {}| expr '>' error
|                "++" ID     {scope(temp,$<s>1,charcount);}|    "++"
error
```

```
|                  "--" ID         {scope(temp,$<s>1,charcount);}|
"--" error
|                expr "<<" expr {}| expr "<<" error
|                expr ">>" expr {}| expr ">>" error
|                expr "==" expr {}| expr "==" error
|                expr "!=" expr {}| expr "!=" error
|                expr ">=" expr {}| expr ">=" error
|                expr "<=" expr {}| expr "<=" error
|                expr "||" expr {}| expr "||" error
|                expr "&&" expr {}| expr "&&" error
|                ID '=' functcall {scope(temp,$<s>1,charcount);}
|                '-' expr  %prec '!'     {}| '-' error
|                ID "++" {scope(temp,$<s>1,charcount);}
|                  ID "--" {scope(temp,$<s>1,charcount);}
|                '!' expr  {}
|                ID  {scope(temp,$<s>1,charcount);}
|                NUM {}
|                  STR {}
;
%%
#include"lex.yy.c"
int count=0;
int main(int argc, char *argv[])
{
   yyin = fopen(argv[1], "r");
   filename = argv[1];
   ifstream myfile(argv[1]);
   string line;
   int cc = 0;
   cout << "\n_____________________________________________\n";
      if (myfile.is_open()) {
      while (getline (myfile, line)) {
            trim(line);
            cout << "|" << ++cc << "|     " << line << endl;
            lines.push_back(line);
      }
      myfile.close();
      }
```

```
   cout << "|_|_____________________________________________\n\n";
   if(!yyparse())
      cout << "\nParsing complete\n";
      else
      cout << "\nParsing failed\n";
      print(head);
      printformat(head);
      fclose(yyin);
      return 0;
}

void yyerror(char *s) {
      printf("Line %d : %s before '%s'\n", lineno, s, yytext);
}
```

Semantic And Symbol Table Code

```
#include<bits/stdc++.h>
using namespace std;
extern vector<string>lines,fargs;
extern string filename,fname;
extern string func;
extern int lineno;
extern int first,lno;
extern int charcount;
extern int fflag,dd;
struct node{
    vector<string>type, datatype, token;
    vector<int>lineno;
    vector<int>dim;
    vector<int>position;
    vector<node*>next;
    node *parent;
```

```
    string name;
    int no_of__block_s;
    int first;
    int func_def_flag;
};
node *head = new node();
node *temp = head;
void checkReturn(int flag){
    int i=0;
    while(head->next[i]){
      if(head->next[i]->name==temp->name)
           break;
      ++i;
    }
    if(head->datatype[i]=="void"){
      if(flag) return;
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: return type mismatch\n";
    }
    else{
      if(!flag) return;
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: return type mismatch\n";
    }
}
void checkDim(){
    for(int i=0;i<fargs.size();++i){
      if(fargs[i][0]=='-'){
           cout << filename << ":" << lineno  << ":" << charcount <<
":" << "error: size of array is negative \n";
           cout << lines[lineno-1];
      }
    }
}
void prtformat(node *temp,int level){
     for(int i=0;i<temp->token.size();++i){
      string fff="-";
      if(temp->token[i]=="function")
```

```
            fff = "1";
      if(temp->token[i]=="_block_")
            fff = "null";
      if(temp->token[i]=="_block_")
            cout <<"\t" << temp->token[i] << "\t\t" << temp->type[i]
<< "\t\t" <<  temp->datatype[i] << "\t\t" << temp->dim[i]  << "\t\t"
<< fff << "\t\t" << level << endl;
      else
            cout <<"\t" << temp->token[i] << "\t" << temp->type[i] <<
"\t\t" <<  temp->datatype[i] << "\t\t" << temp->dim[i]  << "\t\t" <<
fff << "\t\t" << level << endl;

      }
    int j=0;
    for(int i=0;i<temp->token.size();++i){
      if(temp->token[i]=="function" || temp->token[i]=="_block_"){
            prtformat(temp->next[j++],level+1);
      }
     }
}
void printformat(node *temp){
    cout << "\n\n\t\tGLOBAL\n\n";
    cout <<
"\t------------------------------------------------------------------
------------------------\n";
      cout << "\tType\t         Name        Datatype
Dimension    flag     nesting level\n";
      cout <<
"\t------------------------------------------------------------------
------------------------\n";
    prtformat(temp,0);
}

void prt(node *temp){
    cout << "\t-----------------------------------------------\n";
      cout << "\tToken\t        Type   Datatype   line\n";
      cout << "\t-----------------------------------------------\n";
     for(int i=0;i<temp->token.size();++i){
```

```
      cout << "\t" << temp->token[i] << "\t" << temp->type[i] <<
"\t" << setw(5) << temp->datatype[i] << "\t" << setw(5) <<
temp->lineno[i] << endl;
      }
      cout << "\t-------------------------------------------\n";

    int j=0;
    for(int i=0;i<temp->token.size();++i){
      if(temp->token[i]=="function" || temp->token[i]=="_block_"){
            cout << "\n\n\t\t" << temp->type[i]<< "\n\n";
            prt(temp->next[j++]);
      }
     }
}
void print(node *temp){
    cout << "\n\n\t\tGLOBAL\n\n";
    prt(temp);
}
void paramcheck(){
    int k=0;
    node *temp = head;
    for(int i=0;i<head->token.size();++i){
      if(head->token[i]=="function"){
            temp = head->next[k++];
            if(head->type[i]==fname){
                  break;
            }
      }
    }
    --k;
    if(!temp){
      cout << filename << ":" << lineno  << ":" << charcount << ":"
<< "error: undefined reference to '"<< fname << "'\n";return;
    }
    int i;

    for(i=0;i<temp->token.size();++i){
      if(temp->token[i]=="argument"){
```

```
            cout << temp->type[i] <<"arg\n";
            if(i==fargs.size()){
                  cout << filename << ":" << lineno  << ":" <<
charcount << ":" << "error: too few arguments to function '"<< fname
<< "'\n " << lines[lineno-1] << endl;
                  fargs.clear();
                  return;
            }
      }
      else
            break;
    }
    if(i==fargs.size()){    fargs.clear();return;}

    cout << filename << ":" << lineno  << ":" << charcount << ":" <<
"error: too many arguments to function '"<< fname << "'\n " <<
lines[lineno-1] << endl;
    fargs.clear();
}
void placePoint(int position){
    for(int i=1;i<position;++i){
      cout << " ";
    }
    cout << "^" << endl;
}

bool search(node *temp, string token, string type, string
datatype,int position){
    int flag=0;
    for(int i=0;i<temp->token.size();++i){
      if(fflag && temp->type[i]==type && temp->token[i]==token){
            cout << filename << "::" << lineno  << ":" <<
(position-1) << ":" << "error: conflicting types for '"<<
temp->type[i] << "'\n " << lines[lno-1] << endl;
            cout << filename << "::" << temp->lineno[i]  << ":" <<
temp->position[i] << ":" << "note: previous definition of '" <<
temp->type[i] << "' was here\n " << lines[i+1] << endl;
            flag=1;
```

```
            return true;
      }
      if(temp->type[i]==type && temp->token[i]==token &&
temp->datatype[i]!=datatype){
            cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: conflicting types for '"<< temp->type[i] << "'\n "
<< lines[lineno-1] << endl;
            cout << filename << ":" << (i+1)  << ":" <<
temp->position[i] << ":" << "note: previous definition of '" <<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
            flag=1;
      }
      if(temp->type[i]==type && temp->token[i]==token &&
temp->datatype[i]==datatype){
            if(token=="function"){
                  int k=0;
                  node *temp2 = head->next[k++];
                  while(temp2->name!=type){temp2=head->next[k++];}
                  if(temp2->func_def_flag==1){
                        ::temp = temp2;
                  }
                  return true;
            }
            cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: redeclaration of '"<< temp->type[i] << "' with no
linkage\n " << lines[lineno-1] << endl;
            cout << filename << ":" << (i+1)  << ":" <<
temp->position[i] << ":" << "note: previous declaration of '" <<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
            flag=1;
      }
      if(temp->type[i]==type && temp->token[i]!=token){
            cout << filename << ":" << lineno << ":" << (position-1)
<< ":" << "error: '" << temp->type[i] << "' redeclared as different
kind of symbol\n " << lines[lineno-1] << endl;
            cout << filename << ":" << (i+1) << ":" <<
temp->position[i] << ":" << "note: previous definition of '"<<
temp->type[i] << "' was here\n " << lines[i+2] << endl;
```

```
            flag=1;
        }
        if(flag) return true;
    }
    return false;
}
void insVar(string token, string datatype, string type, int lineno,
int position, bool isfunc=false){
    if(isfunc==true){
        if(token=="_block_"){
            stringstream ss;
            temp->no_of__block_s++;
            ss << temp->no_of__block_s;
            string str = ss.str();
            node *ptr = new node();
            temp->next.push_back(new node());
            temp->token.push_back(token);
            temp->dim.push_back(-1);
            temp->lineno.push_back(lineno);
            temp->type.push_back(temp->name + "." + str);
            temp->position.push_back(position);
            temp->datatype.push_back(datatype);
            temp->next.back()->parent = temp;
            temp = temp->next.back();
            temp->no_of__block_s=0;
            temp->name = temp->parent->name + "." + str;
            temp->first=2;
            return;
        }
        if(search(head,token,type,datatype,position))
            return;

        node *ptr = new node();
        head->next.push_back(new node());
        head->token.push_back(token);
        temp->dim.push_back(-1);
        head->lineno.push_back(lineno);
        head->type.push_back(type);
```

```
      head->no_of__block_s=0;
      head->position.push_back(position);
      head->datatype.push_back(datatype);
      head->next.back()->parent = head;
      temp = head->next.back();
      temp->name = type;
      temp->first=1;
    }
    else{
      if(search(temp,token,type,datatype,position))
           return;
      temp->token.push_back(token);
      temp->type.push_back(type);
      temp->dim.push_back(dd);
      temp->lineno.push_back(lineno);
      temp->position.push_back(position);
      temp->datatype.push_back(datatype);
    }
}
void scope(node *temp,string type, int position){
    if(temp==NULL){
      if(func!=""){
           cout << filename << ": In function '" << func << "' :\n";
           cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "error: '" << type << "' undeclared (first use of this
function)\n " << lines[lineno-1] << endl;
           placePoint(position);
           cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "note: each undeclared identifier is reported only once for
each function it appears in\n";
           insVar("  NULL","NULL",type,lineno,position);
      }
      else{
           cout << filename << ":" << lineno  << ":" << (position-1)
<< ":" << "warning: data definition has no type or storage class \n "
<< lines[lineno-1] << endl;
           placePoint(position);
           insVar("  NULL","NULL",type,lineno,position);
```

```
      }
      return;
    }
    for(int i=0;i<temp->token.size();++i){
      if(temp->type[i]==type && temp->token[i]=="variable"){
           return;
      }
    }
    scope(temp->parent,type,position);
}
```

Implementation Details

We have used structure data structure to build the semantic phase. First we had to modify the parser code to insert the values in the symbol table. The structure contains all the information related to the function it contains like line no, datatype, token, position, its parent pointer.

```
struct node{
    vector<string>type, datatype, token;
    vector<int>lineno;
    vector<int>dim;
    vector<int>position;
    vector<node*>next;
    node *parent;
    string name;
    int no_of__block_s;
    int first;
```

```
    int func_def_flag;
};
```

The print format function is there to print the values which are inside the symbol table. There is separate symbol tables for separate functions. The symbol table contains all the information related to size , datatype , array dimension if used,etc. This phase shows the first error in the screen and also shows the complete symbol table. The ‘paramcheck’ function checks whether the parameters passed in the function are correct or not whether the number of parameters are greater or less than the actual one. The ‘search’ function is to detect the type of error during compilation of the program.It checks all the conditions whether the variable is redeclared or not. The ‘insVar’ function is there to detect whether the variable is function or any datatype variable. If it is function a new block is created to store all the vital information in it and if it is a variable we search for its parent (main block). The scope function is present to find the scope of the variable. Scope tells where the variable is defined or in which part.

Test Cases

Test case #1

```
#include<stdio.h>

int a(int x,int y){
    float b;
}

int main()
{
    int x,y;
    int short;
```

```
    return 0;
}
```

Output :

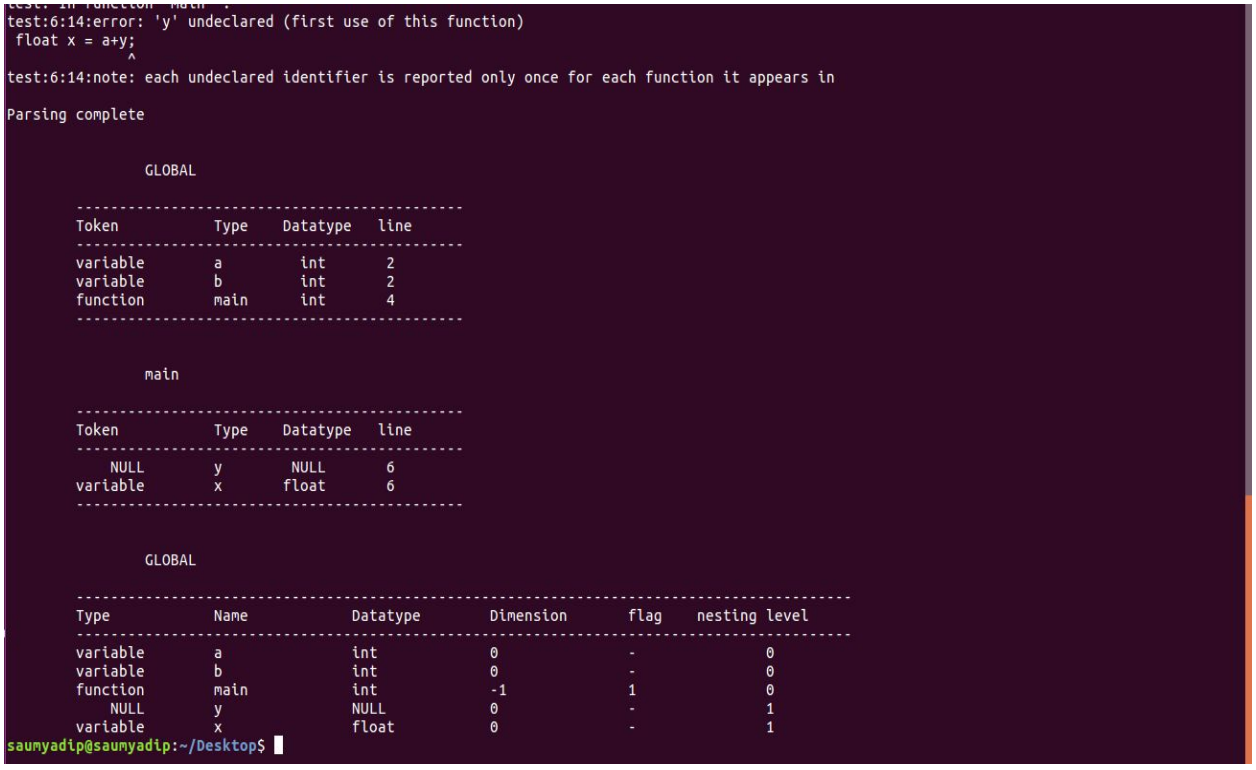
```
Line 11 : syntax error before 'short'

Parsing failed

                GLOBAL

        ------------------------------------------
        Token           Type    Datatype   line
        ------------------------------------------
        function        a         int       3
        function        main      int       7
        ------------------------------------------


                a

        ------------------------------------------
        Token           Type    Datatype   line
        ------------------------------------------
        argument        x         int       3
        argument        y         int       3
        variable        b       float       4
        ------------------------------------------


                main

        ------------------------------------------
        Token           Type    Datatype   line
        ------------------------------------------
        variable        x         int      10
        variable        y         int      10
        ------------------------------------------
```

```
            GLOBAL

     ------------------------------------------------------------------------------------
     Type          Name          Datatype       Dimension      flag   nesting level
     ------------------------------------------------------------------------------------
     function      a             int            -1             1      0
     function      main          int            -1             1      0
     argument      x             int            0              -      1
     argument      y             int            0              -      1
     variable      b             float          0              -      1
     variable      x             int            0              -      1
     variable      y             int            0              -      1
aumyadip@saumyadip:~/Desktop$
```

Test case #2

```
#include<stdio.h>
int a,b=1;
int main(){
    float x = a+y;
    return 0;
}
```

Output

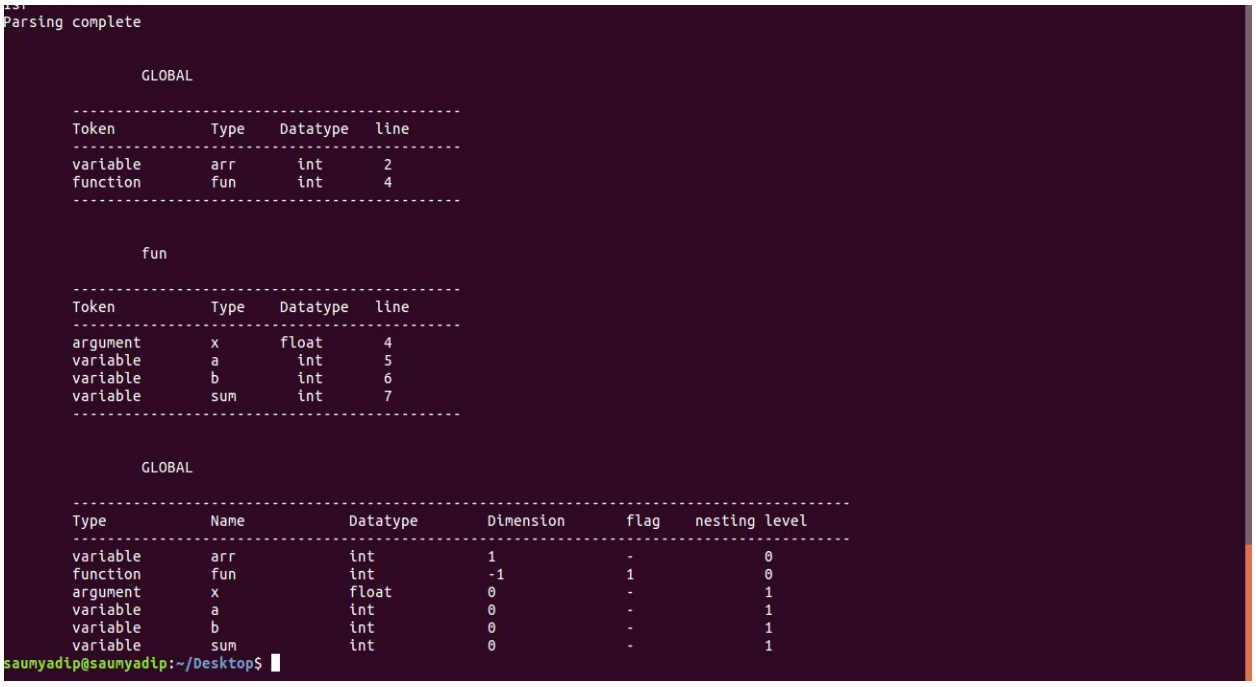


```
test:6:14:error: 'y' undeclared (first use of this function)
 float x = a+y;
           ^
test:6:14:note: each undeclared identifier is reported only once for each function it appears in

Parsing complete


               GLOBAL

       ------------------------------------------
       Token           Type    Datatype   line
       ------------------------------------------
       variable        a         int       2
       variable        b         int       2
       function        main      int       4
       ------------------------------------------


               main

       ------------------------------------------
       Token           Type    Datatype   line
       ------------------------------------------
           NULL        y        NULL       6
       variable        x       float       6
       ------------------------------------------


               GLOBAL

       ------------------------------------------------------------------------------------------
       Type            Name            Datatype        Dimension       flag    nesting level
       ------------------------------------------------------------------------------------------
       variable        a               int             0               -               0
       variable        b               int             0               -               0
       function        main            int             -1              1               0
           NULL        y               NULL            0               -               1
       variable        x               float           0               -               1
saumyadip@saumyadip:~/Desktop$
```

Test case #3

```
#include<stdio.h>
int arr[10005];

int fun(float x){
    int a=1;
    int b=2;
    int sum=a+b;
    return 0;
}
```

Output



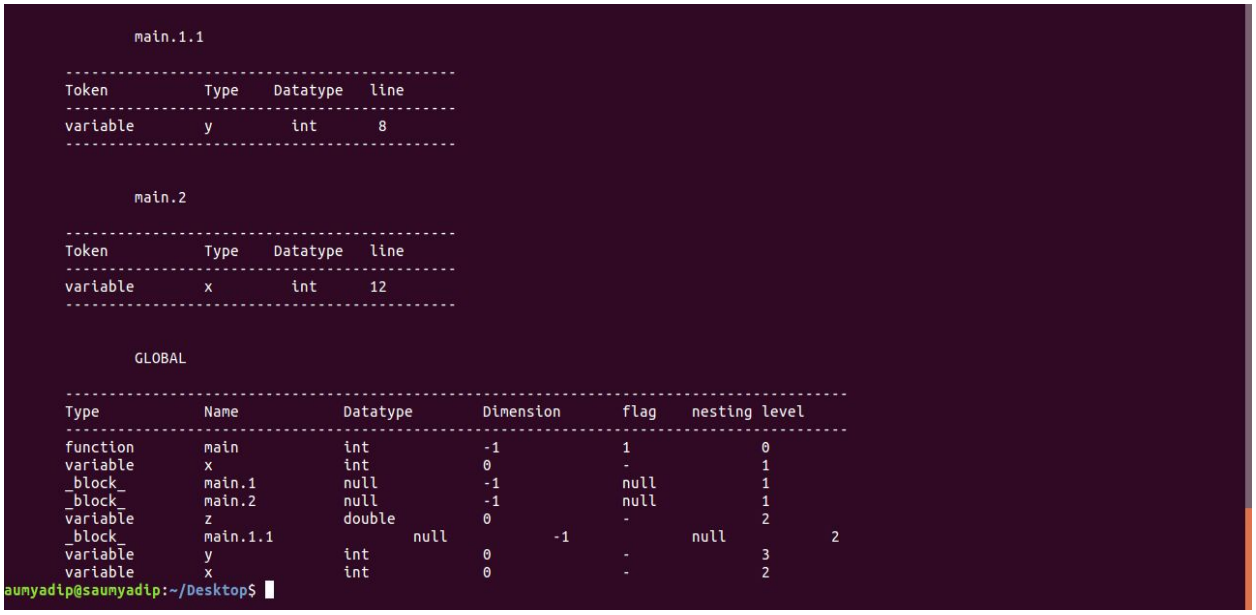
Test case #4

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

```
int main(){
    int x;
    while(x){
       double z;
       {
             int y;
       }
    }
    {
       int x;
    }
}
```

Output

```
Parsing complete


              GLOBAL

      -------------------------------------------
      Token           Type    Datatype   line
      -------------------------------------------
      function        main      int       3
      -------------------------------------------


              main

      -------------------------------------------
      Token           Type    Datatype   line
      -------------------------------------------
      variable        x         int       4
      _block_ main.1   null       5
      _block_ main.2   null      11
      -------------------------------------------


              main.1

      -------------------------------------------
      Token           Type    Datatype   line
      -------------------------------------------
      variable        z       double      6
      _block_ main.1.1         null       7
      -------------------------------------------
```

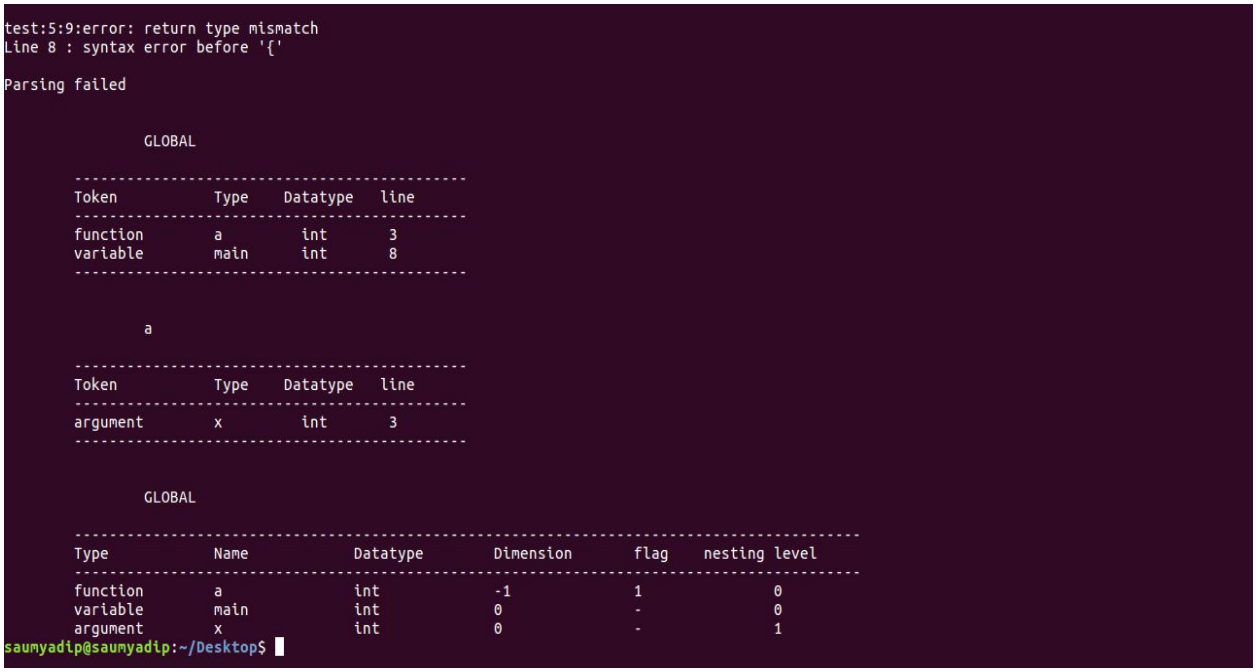


```
        main.1.1

-------------------------------------------
Token           Type    Datatype   line
-------------------------------------------
variable        y         int       8
-------------------------------------------


        main.2

-------------------------------------------
Token           Type    Datatype   line
-------------------------------------------
variable        x         int      12
-------------------------------------------


        GLOBAL

-----------------------------------------------------------------------------------------
Type            Name            Datatype        Dimension       flag    nesting level
-----------------------------------------------------------------------------------------
function        main            int             -1              1               0
variable        x               int             0               -               1
_block_         main.1          null            -1              null            1
_block_         main.2          null            -1              null            1
variable        z               double          0               -               2
_block_         main.1.1                null            -1              null            2
variable        y               int             0               -               3
variable        x               int             0               -               2
aumyadip@saumyadip:~/Desktop$
```

Test case #5

```
include<stdio.h>

int a(int x)
{
    return ;
}

int main{
    a(3);
    return 0;
}
```

Output :



Conclusion

The semantic phase has handled all the cases of errors which were present in the course plan. The following tasks are performed in semantic analysis Scope resolution, Type checking, Array-bound checking,etc. We have not made any changes in the Lex code but changes have been there in the yacc or syntax part. We have used attribute grammar to the grammar to provide context sensitive information. Each attribute has well-defined domain of values, such as integer, float, character, string, and expressions. We have added the semantic code in the symbol table code itself. The symbol table shows all the information about the function like line number , dimension of array , all variables , data types used , etc. Our semantic analyzer has handled all the different errors and it shows the first error when there is an error. For the next phase Intermediate Code generator we have to write assembly level code to implement it.