```
%{
#include<iostream>
#include<cstdlib>
#include<cstring>
#include<cmath>
#include "SymbolInfo.h"

#define YYSTYPE SymbolInfo*

using namespace std;

int yyparse(void);
int yylex(void);
extern FILE *yyin;
 FILE *out;

int labelCount=0;
int tempCount=0;

typedef struct _var{
   string var;
   string ara_size;
}var;
var declar[1000];
int dec_number=0;

typedef struct _parameter{
   string name;
   int typ;
}parameter;
parameter func_param[1000];
int param_size=0;
int arg_size=0;
int param_size2=0;

int param_array[100];
int param=0;

string all_parameter[1000];
int all_param_count=0;

char *newLabel()
{
      char *lb= new char[4];
      strcpy(lb,"L");
      char b[3];
      sprintf(b,"%d", labelCount);
      labelCount++;
      strcat(lb,b);
      return lb;
}

char *newTemp()
```

```
{
	char *t= new char[4];
	strcpy(t,"t");
	char b[3];
	sprintf(b,"%d", tempCount);
	tempCount++;
	strcat(t,b);
	return t;
}

SymbolTable *table =new SymbolTable();
int scope=1;
extern int line_count;
extern int error;

//FILE* fp;
//FILE* fp2;

void yyerror(char *s)
{
	//fprintf(err,"syntax error");
	//fprintf(stderr,"error at line %d: %s\n",line_count,s);
	//fprintf(out,"error at line %d: %s\n",line_count,s);
	//return;
}

SymbolInfo *info = new SymbolInfo();
SymbolInfo *type = new SymbolInfo();
SymbolInfo *tmp = new SymbolInfo();


string current_func;

%}

%token IF FOR WHILE INT FLOAT DOUBLE CHAR RETURN VOID MAIN
%token PRINTLN ADDOP MULOP ASSIGNOP RELOP LOGICOP  NOT SEMICOLON
%token COMMA LPAREN RPAREN LCURL RCURL LTHIRD RTHIRD INCOP DECOP
%token CONST_INT CONST_FLOAT ID

%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

%%

start : program
	{
		ofstream fout;
		fout.open("code.asm");
		fout<<".model small\n.stack 100h\n.DATA\n";
		for(int i=0;i<dec_number;i++){
			if(strcmp(declar[i].ara_size.c_str(),"0")==0){
				fout<<declar[i].var<<" DW ?\n";
			}
```

```
                  else{
                        fout<<declar[i].var <<" DW
"<<declar[i].ara_size<<" DUP(-1)\n";
                  }
            }
            for(int i=0;i<tempCount;i++){
                  fout<<"t"<<i<<" DW ?\n";
            }
            for(int i=0;i<all_param_count;i++){
                  fout<<all_parameter[i]<<" DW ?\n";
            }
            fout<<".CODE\n";
            fout << $1->code;
            fout<<"END MAIN\n";

      }
      ;

program : program unit
      {
            $$=$1;
            $$->code += $2->code;
            delete $2;
      }
      |
      unit
      {
            $$=$1;
      }
      ;

unit : var_declaration{
      }
      |
      func_declaration
      {
                  $$=$1;
      }
      |
      func_definition
      {
                  $$=$1;
      }
      ;

func_declaration : type_specifier func_ID_Lparan parameter_list RPAREN
SEMICOLON
                  {
                  }
                  ;

func_definition :INT MAIN LPAREN RPAREN compound_statement{
                              $$=new SymbolInfo();
                              $$->code ="MAIN PROC\n";
```

```
                              $$->code +=$5->code;
                              $$->code +="MAIN ENDP\n";
                              delete $5;
                        }
                        |type_specifier func_ID_Lparan parameter_list
RPAREN compound_statement{
                              $$=$2;
                              $$->code += $3->code;
                              $$->code += $5->code;
                              char *temp=newTemp();

                              $$->code +="POP BP\n";
                              $$->code +="POP DX\n";
                              $$->code +="POP CX\n";
                              $$->code +="POP BX\n";
                              //$$->code +="POP AX\n";
                              //$$->code +="PUSH "+string(temp)+"\n";
                              char b[8];
                              sprintf(b,"%d", 2*param_size);
                              $$->code +="RET "+string(b)+"\n";
                              $$->code +=$2->get_name()+" ENDP\n";
                              param_size=0;
                              param_size2=0;
                              delete $3;
                              delete $5;

                              table->Exit_Scope();
                              scope--;

                  }
                  ;

func_ID_Lparan    :ID LPAREN{
                  current_func=$1->get_name();
                  tmp=table->Look_Up($1->get_name());
                  if(tmp==NULL){
                        param_size=0;
                        SymbolInfo *temporary = new SymbolInfo($1-
>get_name(),$1->get_type());
                        temporary->data_type=4;
                        temporary->ret_type=type->data_type;
                        table->Insert(temporary);
                        table->Enter_Scope();
                        scope++;
                        $$= temporary;
                        $$->code =$1->get_name()+" PROC\n";
                        //$$->code +="PUSH AX\n";
                        $$->code +="PUSH BX\n";
                        $$->code +="PUSH CX\n";
                        $$->code +="PUSH DX\n";
                        $$->code +="PUSH BP\n";
                        $$->code +="MOV BP,SP\n";
                        }
                  else{
```

```
                        fprintf(out,"Error at Line %d Multiple declaration
of %s\n\n",line_count,tmp->get_name().c_str());
                        error++;
                  }
                  }
                  ;
parameter_list : parameter_lists {
                              $$=new SymbolInfo();
                              tmp= table->Look_Up(current_func);
                              for(int i=0;i<param_size;i++){

      all_parameter[all_param_count]=func_param[i].name;
                                    all_param_count++;
                                    tmp->myList[i] = func_param[i].typ;
                                    char b[8];
                                    sprintf(b,"%d", 2*param_size2);
                                    $$->code += "MOV
AX,[BP+8+"+string(b)+"]\n";
                                    $$->code +="MOV
"+func_param[i].name+",AX\n";
                                    param_size2--;
                              }
                              tmp->list_size=param_size;
                        }
                | {param_size=0;
                          param_size2=0;
                        }
                ;

parameter_lists: parameter_lists COMMA type_specifier ID {
                              tmp= table->Lk_Up($4->get_name());
                              if(tmp==NULL){
                                    SymbolInfo *ob;
                                    if(type->data_type==0){
                                          ob=new SymbolInfo($4-
>get_name(),"int");
                                          ob->data_type=0;
                                          ob->siz=0;
                                    }
                                    else{
                                          ob=new SymbolInfo($4-
>get_name(),"float");
                                          ob->data_type=1;
                                          ob->siz=0;
                                    }
                                    func_param[param_size].name=$4-
>get_name();
                                    func_param[param_size].typ=ob-
>data_type;
                                    param_size++;
                                    param_size2++;
                                    table->Insert(ob);
                              }
                              else{
```

```
                                    fprintf(out,"Error at Line %d multiple
declaration of %s\n\n",line_count,tmp->get_name().c_str());
                                    error++;
                              }
                        }
                | type_specifier ID {
                              tmp= table->Lk_Up($2->get_name());
                              if(tmp==NULL){
                                    SymbolInfo *ob;
                                    if(type->data_type==0){
                                          ob=new SymbolInfo($2-
>get_name(),"int");
                                          ob->data_type=0;
                                          ob->siz=0;
                                    }
                                    else{
                                          ob=new SymbolInfo($2-
>get_name(),"float");
                                          ob->data_type=1;
                                          ob->siz=0;
                                    }
                                    func_param[param_size].name=$2-
>get_name();
                                    func_param[param_size].typ=ob-
>data_type;
                                    param_size++;
                                    param_size2++;
                                    table->Insert(ob);
                              }
                              else{
                                    fprintf(out,"Error at Line %d multiple
declaration of %s\n\n",line_count,tmp->get_name().c_str());
                                    error++;
                              }

                        }
                ;

compound_statement : LCURL statements RCURL{
                  $$=$2;
                  }
                | LCURL RCURL {
                        $$=new SymbolInfo("compound_statement","dummy");
                  }
                ;

var_declaration : type_specifier declaration_list SEMICOLON {
             }
             ;

type_specifier   : INT {
                  type->data_type=0;

            }
```

```
            | FLOAT {
                  type->data_type=1;

            }
            | VOID {
                  type->data_type=2;
            }
            ;

declaration_list : declaration_list COMMA ID {
                        tmp= table->Lk_Up($3->get_name());
                        if(tmp==NULL){
                              SymbolInfo *temporary = new SymbolInfo($3-
>get_name(),"notarray");
                              temporary->data_type=type->data_type;
                              temporary->siz=0;
                              declar[dec_number].var=$3->get_name();
                              declar[dec_number].ara_size="0";
                              dec_number++;
                              table->Insert(temporary);
                        }
                        else{
                              fprintf(out,"Error at Line %d %s is already
declared\n\n",line_count,$3->get_name().c_str());
                              error++;
                        }
              }
              | declaration_list COMMA ID LTHIRD CONST_INT RTHIRD {
                        tmp= table->Lk_Up($3->get_name());
                        if(tmp==NULL){
                              SymbolInfo *temporary = new SymbolInfo($3-
>get_name(),"array");
                              temporary->data_type=type->data_type;
                              temporary->siz= $5->val.i;
                              declar[dec_number].var=$3->get_name();
                              declar[dec_number].ara_size=$5->get_name();
                              dec_number++;
                              table->Insert(temporary);
                        }
                        else{
                              fprintf(out,"Error at Line %d %s is already
declared\n\n",line_count,$3->get_name().c_str());
                              error++;
                        }
              }
              | ID LTHIRD CONST_INT RTHIRD {
                        tmp= table->Lk_Up($1->get_name());

                        if(tmp==NULL){
                              SymbolInfo *temporary = new SymbolInfo($1-
>get_name(),"array");
                              temporary->data_type=type->data_type;
                              temporary->siz=$3->val.i;
                              declar[dec_number].var=$1->get_name();
```

```
                              declar[dec_number].ara_size=$3->get_name();
                              dec_number++;
                              table->Insert(temporary);

                        }
                        else{
                              fprintf(out,"Error at Line %d %s is already
declared\n\n",line_count,$1->get_name().c_str());
                              error++;
                        }

              }
              | ID {
                        tmp= table->Lk_Up($1->get_name());
                        if(tmp==NULL){
                              SymbolInfo *temporary = new SymbolInfo($1-
>get_name(),"notarray");
                              temporary->data_type=type->data_type;
                              temporary->siz=0;
                              declar[dec_number].var=$1->get_name();
                              declar[dec_number].ara_size="0";
                              dec_number++;
                              table->Insert(temporary);
                        }
                        else{
                              fprintf(out,"Error at Line %d %s is already
declared \n\n",line_count,$1->get_name().c_str());
                              error++;
                        }
              }

              ;

statements : statement {
                  $$=$1;
          }
          | statements statement {
                  $$=$1;
                  $$->code += $2->code;
                  delete $2;
          }
          ;

statement : var_declaration {
        }
        | expression_statement {
                  $$=$1;
        }
        | compound_statement{
                  $$=$1;

        }
        | conditional expression_statement expression_statement
expression RPAREN statement{
```

```
            $$=$2;
            char *label1=newLabel();
            char *label2=newLabel();
            $$->code +=string(label1)+": \n";
            $$->code += $3->code;
            $$->code+="mov ax, "+$3->get_name()+"\n";
            $$->code+="cmp ax, 0\n";
            $$->code+="je "+string(label2)+"\n";
            $$->code += $6->code;
            $$->code += $4->code;
            $$->code +="jmp "+string(label1)+"\n";
            $$->code +=string(label2)+": \n";


    }
    | conditional expression RPAREN statement %prec LOWER_THAN_ELSE {

            $$=$2;
            char *label=newLabel();
            $$->code+="mov ax, "+$2->get_name()+"\n";
            $$->code+="cmp ax, 0\n";
            $$->code+="je "+string(label)+"\n";
            $$->code+=$4->code;
            $$->code+=string(label)+":\n";
            $$->set_name("if");
    }
    | conditional expression RPAREN statement ELSE statement{
            $$=$2;
            char *label1=newLabel();
            char *label2=newLabel();
            $$->code+="mov ax, "+$2->get_name()+"\n";
            $$->code+="cmp ax, 0\n";
            $$->code+="je "+string(label1)+"\n";
            $$->code+=$4->code;
            $$->code+="JMP "+string(label2)+"\n";
            $$->code+=string(label1)+":\n";
            $$->code+=$6->code;
            $$->code+=string(label2)+":\n";
            $$->set_name("if");
    }
    | WHILE LPAREN  expression RPAREN statement{
            //$$ = new SymbolInfo();
            $$=$3;
            char *label1=newLabel();
            char *label2=newLabel();
            $$->code =string(label1)+": \n";
            $$->code+=$3->code;
            $$->code+="mov ax, "+$3->get_name()+"\n";
            $$->code+="cmp ax, 0\n";
            $$->code+="je "+string(label2)+"\n";
            $$->code += $5->code;
            $$->code +="jmp " +string(label1)+"\n";
            $$->code +=string(label2)+": \n";
            delete $5;
```

```
        }
        | PRINTLN LPAREN ID RPAREN SEMICOLON{
                  tmp= table->Look_Up($3->get_name());
                  if(tmp!=NULL){
                        if(tmp->siz==0){
                              $$=new SymbolInfo("println","nonterminal");
                              $$->code+="mov AH,2\n";
                              $$->code+="mov DX,"+$3->get_name()+"\n";
                              $$->code+="ADD DX,'0'\n";
                              $$->code+="INT 21H\n";
                        }
                  }
                  else{
                        fprintf(out,"Error at Line %d Undeclared
identifier",line_count);
                        error++;
                  }
        }
        | RETURN expression SEMICOLON{
                  $$=new SymbolInfo($2);
                  $$->code +="MOV AX,"+$2->get_name()+"\n";

        }
        ;

conditional :IF LPAREN {
                        table->Enter_Scope();
                        scope++;
                  }
                  |FOR LPAREN{
                        table->Enter_Scope();
                        scope++;
                  }
                  ;


expression_statement   : SEMICOLON      {
                                                $$=new
SymbolInfo(";","SEMICOLON");
                                                $$->code="";
                                    }
                  | expression SEMICOLON {
                        $$=$1;
                  }
                  ;

variable : ID     {
                  $$= new SymbolInfo($1);
                  $$->code="";
                  $$->set_type("notarray");
       }
       | ID LTHIRD expression RTHIRD {
                         $$ = new SymbolInfo();
```

```
                        SymbolInfo* info = table->Look_Up($1-
>get_name());
                        SymbolInfo* value;
                        if(info!=NULL){

                             if(info->siz >0){
                                  if(($3->data_type==0) && ($3->val.i
>=0) && ($3->val.i <info->siz)){
                                       $$=info;
                                       $$->set_type("array");
                                       $$->code=$3->code+"mov bx, " +$3-
>get_name() +"\nadd bx, bx\n";
                                       delete $3;
                                  }
                                  else if($3->data_type==1){
                                       fprintf(out,"Error at line %d
index can not be a float\n\n",line_count);
                                       error++;
                                  }
                                  else if(($3->val.i <0)&&($3->val.i
>=info->siz )){
                                       fprintf(out,"Error at line %d
index out of bound\n\n",line_count);
                                       error++;
                                  }
                             }
                             else{
                                  fprintf(out,"Error at line %d index on
non array\n\n",line_count);
                                  error++;
                             }

                        }
                        else{
                             fprintf(out,"Error at line %d Undeclared
Variable %s\n\n",line_count,$1->get_name().c_str());
                             error++;

                        }
                   }
                   ;

expression : logic_expression     {
                   $$=$1;
              }
          | variable ASSIGNOP logic_expression {
                   $$=new SymbolInfo($1);

                   $$->code=$3->code+$1->code;

                   $$->code+="mov ax, "+$3->get_name()+"\n";
                   if($$->get_type()=="notarray"){
                        $$->code+= "mov "+$1->get_name()+", ax\n";
                   }
```

```
                  else{
                        $$->code+= "mov  "+$1->get_name()+"[bx], ax\n";
                  }
                  delete $3;

         }
         ;

logic_expression : rel_expression {
                        $$=$1;
                  }
              | rel_expression LOGICOP rel_expression {


                              $$ = new SymbolInfo();
                              $$=$1;
                              $$->code+=$3->code;
                              //cout<<$$->code<<endl;
                              char *temp=newTemp();
                              char *label1=newLabel();
                              char *label2=newLabel();
                              if($2->get_name()=="&&"){
                                    /*
                                    Check whether both operands value is 1.
If both are one set value of a temporary variable to 1
                                    otherwise 0
                                    */

                                    $$->code+="cmp " + $1-
>get_name()+",1\n";
                                    $$->code+="jne "+string(label1)+"\n";
                                    $$->code+="cmp " + $3-
>get_name()+",1\n";
                                    $$->code+="jne "+string(label1)+"\n";
                                    $$->code+="mov "+string(temp) +", 1\n";
                                    $$->code+="jmp "+string(label2) +"\n";
                                    $$->code+=string(label1)+":\n";
                                    $$->code+="mov "+string(temp) +", 0\n";
                                    $$->code+=string(label2)+":\n";
                              }
                              else if($2->get_name()=="||"){

                                    $$->code+="cmp " + $1-
>get_name()+",1\n";

                                    $$->code+="je "+string(label1)+"\n";

                                    $$->code+="cmp " + $3-
>get_name()+",1\n";

                                    $$->code+="je "+string(label1)+"\n";
                                    $$->code+="mov "+string(temp) +", 0\n";
                                    $$->code+="jmp "+string(label2) +"\n";
```

```
                                    $$->code+=string(label1)+":\n";
                                    $$->code+="mov "+string(temp) +", 1\n";
                                    $$->code+=string(label2)+":\n";

                              }
                              $$->set_name(temp);
                              delete $3;
            }
             ;

rel_expression   : simple_expression {
                 $$=$1;
                 }
            | simple_expression RELOP simple_expression   {
                        $$ = new SymbolInfo();
                        $$=$1;
                        $$->code+=$3->code;
                        $$->code+="mov ax, " + $1->get_name()+"\n";
                        $$->code+="cmp ax, " + $3->get_name()+"\n";
                        char *temp=newTemp();
                        char *label1=newLabel();
                        char *label2=newLabel();
                        if($2->get_name()=="<"){
                              $$->code+="jl " + string(label1)+"\n";
                        }
                        else if($2->get_name()=="<="){
                              $$->code+="jle " + string(label1)+"\n";
                        }
                        else if($2->get_name()==">"){
                              $$->code+="jg " + string(label1)+"\n";
                        }
                        else if($2->get_name()==">="){
                              $$->code+="jge " + string(label1)+"\n";
                        }
                        else if($2->get_name()=="=="){
                              $$->code+="je " + string(label1)+"\n";
                        }
                        else{
                              $$->code+="jne " + string(label1)+"\n";
                        }

                        $$->code+="mov "+string(temp) +", 0\n";
                        $$->code+="jmp "+string(label2) +"\n";
                        $$->code+=string(label1)+":\nmov "+string(temp)+",
1\n";
                        $$->code+=string(label2)+":\n";
                        $$->set_name(temp);
                        delete $3;
            }
            ;

simple_expression : term {
                 $$=$1;
                 }
```

```
              | simple_expression ADDOP term {
                        $$ = new SymbolInfo();
                        $$=$1;
                        char *temp=newTemp();
                        //$$->code+=$1->code;
                        $$->code+=$3->code;
                        $$->code+="mov ax, " + $1->get_name()+"\n";


                        // move one of the operands to a register, perform
addition or subtraction with the other operand and move the result in a
temporary variable

                        if($2->get_name()=="+"){
                              $$->code+="ADD ax,"+ $3->get_name()+"\n";
                        }
                        else{
                              $$->code+="SUB ax,"+ $3->get_name()+"\n";
                        }
                        $$->code+="mov "+string(temp) +", ax\n";
                        $$->set_name(temp);

                        delete $3;


              }
              ;

term :      unary_expression {
            $$=$1;

            }
     |  term MULOP unary_expression{
                        $$ = new SymbolInfo();
                        $$=$1;
                        $$->code += $3->code;
                        $$->code += "mov ax, "+ $1->get_name()+"\n";
                        $$->code += "mov bx, "+ $3->get_name() +"\n";
                        char *temp=newTemp();
                        if($2->get_name()=="*"){
                              $$->code += "mul bx\n";
                              $$->code += "mov "+ string(temp) + ", ax\n";
                        }
                        else if($2->get_name()=="/"){
                              // clear dx, perform 'div bx' and mov ax to
temp
                              $$->code += "mov dx,0\n";
                              $$->code += "DIV bx\n";
                              $$->code += "mov "+ string(temp) + ", ax\n";

                        }
                        else{
                              // clear dx, perform 'div bx' and mov dx to
temp
```

```
                        $$->code += "mov dx,0";
                        $$->code += "DIV bx\n";
                        $$->code += "mov "+ string(temp) + ", dx\n";
                    }
                    $$->set_name(temp);

                    delete $3;
      }

    ;

unary_expression : ADDOP unary_expression  {
                $$ = new SymbolInfo();
                $$=$2;
                char *temp=newTemp();
                if(strcmp($1->get_name().c_str(),"-")==0){
                    if($2->siz==0){
                        $$->code="mov ax, " + $2->get_name() + "\n";
                        $$->code+="NEG ax\n";
                        $$->code+="mov "+string(temp)+", ax";
                        $$->set_name(temp);
                    }
                    else{
                        $$->code="mov ax, " + $2->get_name() +
"[bx]\n";
                        $$->code+="NEG ax\n";
                        $$->code+="mov "+string(temp)+", ax";
                        $$->set_name(temp);
                    }
                }

            }
            | NOT unary_expression {

                $$ = new SymbolInfo();
                $$=$2;
                char *temp=newTemp();
                    if($2->siz==0){
                        $$->code="mov ax, " + $2->get_name() + "\n";
                        $$->code+="not ax\n";
                        $$->code+="mov "+string(temp)+", ax";
                    }
                    else{
                        $$->code="mov ax, " + $2->get_name()
+"[bx]\n";
                        $$->code+="not ax\n";
                        $$->code+="mov "+string(temp)+", ax";

                    }
            }
            | factor {
                $$=$1;
            }
            ;
```

```
factor      : variable{
                  $$= $1;

                  if($$->siz==0){

                  }

                  else{
                        char *temp= newTemp();
                        $$->code+="mov ax, " + $1->get_name() + "[bx]\n";
                        $$->code+= "mov " + string(temp) + ", ax\n";
                        $$->set_name(temp);
                  }
            }
      | ID LPAREN argument_list RPAREN{

                  $$ = new SymbolInfo();
                  tmp= table->Look_Up($1->get_name());

                  int flag=0;
                  if(tmp!=NULL){
                        if(tmp->data_type==4){
                              if(tmp->list_size==arg_size){
                                    for(int i=0;i<arg_size;i++){
                                          if(func_param[i].typ!=tmp-
>myList[i]){
                                                fprintf(out,"Error at Line
%d %d th parameter mismatch\n\n",line_count);
                                                error++;
                                                flag=1;
                                                break;
                                          }
                                    }
                                    if(flag==0){
                                          $$->code =$3->code;
                                          char *temp= newTemp();
                                          for(int i=0;i<arg_size;i++){
                                                $$->code +="PUSH
"+func_param[i].name+"\n";
                                          }
                                          $$->code +="CALL "+$1-
>get_name()+"\n";

                                          $$->code +="MOV
"+string(temp)+",AX\n";

                                          $$->set_name(temp);

                                          arg_size=0;
                                          delete $3;
                                    }
                              }
                              else{
```

```
                                    fprintf(out,"Error at Line %d Total
Number of Arguments mismatch in funtion %s\n\n",line_count,$1-
>get_name().c_str());
                                    error++;
                                }
                        }
                        else{
                                fprintf(out,"Error at Line %d %s is not a
function\n\n",line_count,$1->get_name().c_str());
                                error++;
                        }
                }
                else{
                        fprintf(out,"Error at Line %d Undeclared function
%s\n\n",line_count,$1->get_name().c_str());
                        error++;
                }
                arg_size=0;

      }
      | LPAREN expression RPAREN{
                $$= $2;
      }
      | CONST_INT {
            $$= $1;
      }
      | CONST_FLOAT{
            $$= $1;
            }
      | variable INCOP {
                $$=$1;
                if($1->siz==0){
                        $$->code+= "INC "+$1->get_name()+"\n";
                }
                else{
                        $$->code+= "INC  "+$1->get_name()+"[bx]\n";
                }


      }
      | variable DECOP {

                $$=$1;
                if($1->siz==0){
                        $$->code+= "DEC "+$1->get_name()+"\n";
                }
                else{
                        $$->code+= "DEC  "+$1->get_name()+"[bx]\n";
                }
      }
      ;

argument_list : arguments {
                        $$=$1;
```

```
                        }
                |{arg_size=0;}
                ;
arguments: arguments COMMA logic_expression  {
                                $$=$1;
                                $$->code += $3->code;
                                func_param[arg_size].name=$3->get_name();
                                func_param[arg_size].typ=$3->data_type;
                                arg_size++;
                                delete $3;
                    }
            | logic_expression {
                                $$=new SymbolInfo($1);
                                func_param[arg_size].name=$1->get_name();
                                func_param[arg_size].typ=$1->data_type;
                                arg_size++;

                    }
            ;


%%
int main(int argc,char *argv[])
{

      FILE *fin=fopen(argv[1],"r");
      if(fin==NULL){
            printf("Cannot open specified file\n");
            return 0;
      }

      out= fopen("log.txt","w");



      yyin= fin;
      yyparse();
      cout << endl;


      printf("\nTotal Lines: %d\n",line_count);
      printf("\nTotal Errors: %d\n",error);

      fprintf(out,"\nTotal Lines: %d\n",line_count);
      fprintf(out,"\nTotal Errors: %d\n",error);

      printf("\n");
      return 0;


}
```