Lexer1.lex

```
%{
	#include <bits/stdc++.h>
	#include "parser1.tab.h"
	using namespace std;
%}

%option yylineno
%option noyywrap
%s HASH

lineComment     "//".*
blockComment    "/*"((("*"[^/])?)|[^*])*"*/"
SEMI   ";"
EQUAL          "="
ADD            "+"
SUB            "-"
MUL            "*"
DIV            "/"
GT             ">"
LT             "<"
GE             ">="
LE             "<="
EQ             "=="
NE             ("!=")
OR             "||"
AND    "&&"
LC             "{"
RC             "}"
LB             "("
RB             ")"
LBP            "["
RBP            "]"
COMMA          ","
MAIN   "main"
INT            "int"
VOID   "void"
FLOAT          "float"
RETURN         "return"
IF             "if"
FOR            "for"
WHILE "while"
ELSE   "else"
BREAK   "break"
PRINT          "printf"
```

```
READ  "scanf"
CONTINUE    "continue"
SWITCH                  "switch"
CASE            "case"
DEFAULT         "default"
COLON                   ":"
INTEGERS                                            ([0-9]+)
FLOATING_POINTS                                     ([0-9]+\.[0-9]+)
LIBRARY                                             (\#include[ \n\t]*\<.+\>)|((\#include[
\t\n]*\".+\"))
ID                                                          ([A-Za-z_]([A-Za-z0-9_])*)
WHITE_SPACES                                        ([ \t]+)
NEW_LINE                                            ([\n])
STRING                                                      \"(\\.|[^\"])*\"

%%
{lineComment}   {}
{blockComment}  {}
{SEMI}                  {return SEMI;}
{EQUAL}                         {return EQUAL;}
{ADD}                   {return ADD;}
{SUB}                   {return SUB;}
{MUL}                   {return MUL;}
{DIV}                   {return DIV;}
{GT}                    {return GT;}
{LT}                    {return LT;}
{GE}                    {return GE;}
{LE}                    {return LE;}
{EQ}                    {return EQ;}
{NE}                    {return NE;}
{MAIN}                          {return MAIN;}
{INT}                   {return INT;}
{VOID}                          {return VOID;}
{FLOAT}                 {return FLOAT;}
{RETURN}                {return RETURN;}
{OR}                    {return OR;}
{AND}                   {return AND;}
{IF}                    {return IF;}
{FOR}                   {return FOR;}
{WHILE}                 {return WHILE;}
{ELSE}                          {return ELSE;}
{BREAK}                 {return BREAK;}
{CONTINUE}              {return CONTINUE;}
{LC}                    {return LC;}
{RC}                    {return RC;}
{LB}                    {return LB;}
```

```
{RB}                    {return RB;}
{LBP}                   {return LBP;}
{RBP}                   {return RBP;}
{COMMA}                         {return COMMA;}
{SWITCH}        {return SWITCH;}
{CASE}                  {return CASE;}
{DEFAULT}               {return DEFAULT;}
{COLON}                 {return COLON;}
{PRINT}                         {return PRINT;}
{STRING}                {yylval.stringVal = strdup(yytext);return STRING;}
{READ}                          {return READ;}

{INTEGERS}                              {yylval.stringVal = strdup(yytext);return INTEGERS;}
{LIBRARY}                               {yylval.stringVal = strdup(yytext);return LIBRARY;}
{FLOATING_POINTS}                       {yylval.stringVal = strdup(yytext);return
FLOATING_POINTS;}
{ID}                                    {yylval.stringVal = strdup(yytext);return ID;}

{NEW_LINE}                      {}
{WHITE_SPACES}                  {}

.                                       {cerr<< "TOKEN CANNOT BE MATCHED :\t"<< yytext
<<"\t"<<endl;}

%%
```

Lexer2.lex

```
%{

#include<stdio.h>
#include<iostream>
#include "parser2.tab.h"
using namespace std;

%}
%%
"NULL"
        {return(NULLL);}
"read"
        {return(READD);}
"print"
        {return(PRINTT);}
"decl"
        {return(DECL);}
```

```
"func"
        {return(FUNC);}
"begin"
        {return(BEGINN);}
"return"
        {return(RETURN);}
"end"
        {return(END);}
"param"
                {return(PARAM);}
"refparam"
        {return(REFPARAM);}
"call"
        {return(CALL);}
"args"
        {return(ARGS);}
"if"
        {return(IF);}
"goto"
        {return(GOTO);}
\"(\\.|[^\"])*\"
{return(STRINGG);}
[a-zA-z]+[a-zA-z0-9._]*[(][a-zA-z0-9._]+[)]                         {return(ID);}
[a-zA-z]+[a-zA-z0-9._]*
{return(ID);}
"=="
        {return(ARITH_REL_OPS);}
"<="
        {return(ARITH_REL_OPS);}
">="
        {return(ARITH_REL_OPS);}
"!="
        {return(ARITH_REL_OPS);}
[-+*/<>]
        {return(ARITH_REL_OPS);}
[0-9]+
        {return(INT);}
[-][0-9]+
        {return(INT);}
[0-9]+[.][0-9]+
{return(FLOAT);}
[-][0-9]+[.][0-9]+
{return(FLOAT);}
[=]
        {return(EQ);}
```

```
[a-zA-z]+[a-zA-z0-9]*[:]
{return(LABEL);}

(.|\n)
%%

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

Parser1.y

```
%{
	#include <bits/stdc++.h>
	#include<string.h>
	#define pb push_back

	using namespace std;

	extern int yylex();
	extern int yyparse();
	extern int yylineno;
	void yyerror(string s);
	extern char* yytext;
	extern int yyleng;
	int syntaxERROR = 0;

	void yyerror(char *s){
		syntaxERROR = 1;
		printf ("Syntax Error in line no. %d\n",yylineno);
	}

	struct ptr{
		vector<ptr*> children;
		string dtype;
		float value;
		string svalue;
		string tag;
		int gScope = 0;
		vector<int> dimptr;
		vector<int> dimptrorg;
		vector<string> dimptrstr;
	};

	struct variable{
		vector<int> dim;
		string name;
		int array;
		int scope;
		string dtype;
	};

	struct func{
		int numparam;
		vector< variable * > params;
		string returntype;
		string name;
	};

	vector < map< string , variable* >> SymTable;
	map<string , func* > FuncTable;
	string activeFunc = "";
	string returnType = "";
	vector<vector<string>> para;
	string currFunc = "";
	vector<string> callFunc;
	int gScope = 0;
	int semanticERROR = 0;
	ptr * treeRoot;
	string gtype = "";
	string gid = "";
	vector<int> gdimv;
	vector<string> gfcallparam;
	vector<vector<string>> gfcallparam2d;
	vector< variable * > gparams;
	void SymTablePrint();
	string convert(string s);
	int checkOutofBound(vector<int> v);
	int findScope(string gid);
	string decideintfloat(string a, string b);
	variable * gvar;
	string chk;
	int printFlag = 0;
	vector<vector<int>> gdimv2d;
	vector<string> brk ,cont;
	FILE * f = fopen("intermediate.txt", "w");
	FILE * q = fopen("quadruple.txt", "w");
	vector<ptr*> funcList;

	vector<vector<string>> brlist;
%}
```

```
%union
{
        struct ptr* Ptr;
        char * stringVal;
}

%token ADD SUB MUL DIV GT LT GE LBP RBP LE EQ NE MAIN INT FLOAT PRINT
RETURN OR AND IF FOR READ WHILE ELSE BREAK CONTINUE INTEGERS
FLOATING_POINTS ID SEMI LC RC LB RB COMMA EQUAL  LIBRARY VOID SWITCH
CASE DEFAULT COLON STRING
%type<Ptr> grammar_start libraries decls decl break continue var_decl type var_list var id
br_list br_list1 for_exp func_decl lbf lcf rcf func_end decl_plist decl_pl decl_param body
stmts stmt exp case_exp default_exp return_exp exp_type_1 exp_type_2 exp_type_3
arith_exp_type_1 arith_exp_type_2 unary_exp term func_call args args_list args1 args_list1
consts intg floats plus_minus_op mul_div_op relation_op unary_operator string
%start  grammar_start

%%
grammar_start :         libraries decls INT MAIN LB RB lcf body rcf

                        {

                                treeRoot = new ptr;

                                treeRoot->children.pb($1);

                                treeRoot->children.pb($2);

                                treeRoot->children.pb($7);

                                treeRoot->children.pb($8);

                                treeRoot->children.pb($9);

                                treeRoot->tag = "START";

                        }
                |   error RC                                            { yyerrok;
syntaxERROR = 1;treeRoot = new ptr;}

libraries :             LIBRARY libraries

                        {

                                ptr *t = new ptr;
                        {
                                t->tag = "LIBRARIES";
                                t->gScope = gScope;
                                (t->children).pb($2);
                                $$ = t;
                        }
                                |  LIBRARY
                        {
                                ptr *t = new ptr;
                                t->gScope = gScope;
                                t->tag = "LIBRARIES";
                                $$ = t;
                        }
decls :         decls decl
                        {
                                ptr *t = new ptr;
                                t->tag = "GDECLS";
                                t->gScope = gScope;
                                (t->children).pb($1);
                                (t->children).pb($2);
                                $$ = t;
                        }
                                |
                        {
                                ptr * t = new ptr;
```

```
                    {
                            t->gScope = gScope;
                            t->tag = "GDECLS";
                            $$ = t;
                    }
decl :              func_decl
                    {
                            ptr * t = new ptr;
                            t->tag = "GDECL";
                            t->gScope = gScope;
                            (t->children).pb($1);
                            $$ = t;
                    }
                            |       var_decl
                    {
                            ptr * t = new ptr;
                            t->tag = "GDECL";
                            t->gScope = gScope;
                            (t->children).pb($1);
                            $$ = t;
                    }
                            | exp SEMI

                    {
```

```
                    ptr * t = new ptr;


                    t->tag = "STMTEXP";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    $$ = t;

                }

                        | error SEMI
        {yyerrok;syntaxERROR=1;}

var_decl :          type var_list SEMI

                {

                    ptr * t = new ptr;

                    t->tag = "VARDECL";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    (t->children).pb($2);

                    $$ = t;

                }
                        | type var EQUAL exp_type_1 SEMI

                {

                    ptr * t = new ptr;

                    t->tag = "VARDECL";

                    t->gScope = gScope;

                    (t->children).pb($1);
                }

                    (t->children).pb($2);

                    (t->children).pb($4);

                    $$ = t;

                }

type :                  INT

                {

                    ptr * t = new ptr;

                    t-> dtype = "int";

                    t->gScope = gScope;

                    gtype = "int";

                    t-> tag = "TYPE";

                    $$ = t;

                }
                        | FLOAT

                {

                    ptr * t = new ptr;

                    t-> dtype = "float";

                    t->gScope = gScope;

                    gtype = "float";

                    t-> tag = "TYPE";

                    $$ = t;

                }

void :                  VOID
```

```
                    {
                        gtype = "void";
                    }
var_list :              var_list COMMA var
                    {
                        ptr * t = new ptr;
                        t->gScope = gScope;
                        t->tag = "VARLIST";
                        t->dtype = gtype;
                        (t->children).pb($1);
                        (t->children).pb($3);
                        $$ = t;
                    }
                            | var
                    {
                        ptr * t = new ptr;
                        t->gScope = gScope;
                        t->tag = "VARLIST";
                        t->dtype = gtype;
                        (t->children).pb($1);
                        $$ = t;
                    }
var :                   id
```

```
        {
            ptr * t = new ptr;
            t->tag = "VAR";
            t->dtype = gtype;
            t->gScope = gScope;
            t->svalue = gid;
            (t->children).pb($1);
            $$ = t;

            if( SymTable[gScope].find(gid)==SymTable[gScope].end() ){
                if(gScope==2 && gparams.size()!=0 &&
SymTable[1].find(gid)!=SymTable[1].end() ){
                    cout << "Semantic Error : Redecleration of
param as variable " << gid << " in line no. " << yylineno<< endl;
                    semanticERROR = 1;
                }
                else{
                    variable * v = new variable;
                    v->array = 0;
                    v->name = gid;
                    v->dtype = gtype;
                    v->scope = gScope;
                    SymTable[gScope][gid] = v;
                    gid = "";
```

```
                                        gvar = v;

                                    }

                                }else{

                                    cout <<  "Semantic Error : Multiple declarations of
Variable : " << gid << " in line no. " << yylineno<< endl;

                                    semanticERROR = 1;

                                }

                                // SymTablePrint();

                            }

                                    | id br_list

                            {

                                ptr * t = new ptr;

                                t->tag = "VARARRAY";

                                t->dtype = gtype;

                                t->gScope = gScope;

                                t->svalue = gid;

                                (t->children).pb($1);

                                (t->children).pb($2);

                                $$ = t;


                                if(SymTable[gScope].find(gid)==SymTable[gScope].end()){

                                    if(gScope==2 && gparams.size()!=0 &&
SymTable[1].find(gid)!=SymTable[1].end() ){

                                        cout << "Semantic Error : Redecleration of
param as var " << gid << " in line no. " << yylineno<< endl;
```

```
                                        semanticERROR = 1;

                                }

                                else{

                                        variable * v = new variable;

                                        v->array = 1;

                                        v->name = gid;

                                        v->dtype = gtype;

                                        v->dim = gdimv;

                                        t->dimptr=gdimv;

                                        gdimv = gdimv2d.back();

                                        gdimv2d.pop_back();

                                        v->scope = gScope;

                                        SymTable[gScope][gid] = v;

                                        gid = "";

                                        gvar = v;

                                }

                        }else{

                                cout <<  "Semantic Error : Multiple declarations of
Variable : " << gid << " in line no. " << yylineno<< endl;

                                semanticERROR = 1;

                        }

                        //SymTablePrint();

                }
id:                     ID
```

```
                    {
                            ptr * t = new ptr;
                            t->tag = "ID";
                            t->dtype = gtype;
                            t->svalue = yylval.stringVal;
                            gid = yylval.stringVal;
                            int scp = findScope(gid);

                            if(scp==-1)
                            {
                                    t->gScope = gScope;
                            }
                            else
                            {
                                    t->dtype = SymTable[scp][t->svalue]->dtype;
                                    t->gScope = scp;
                            }
                            $$ = t;
                    }
br_list :                   LBP intg RBP
                    {
                            ptr * t = new ptr;
                            t->tag = "BRLIST";
```

```
                        t->gScope = gScope;

                        (t->children).pb($2);

                        gdimv2d.pb(gdimv);

                        gdimv.clear();

                        gdimv.pb(($2)->value);

                        $$ = t;

                }
                              | br_list LBP intg RBP

                {

                        ptr * t = new ptr;

                        t->tag = "BRLIST";

                        t->gScope = gScope;

                        (t->children).pb($1);

                        (t->children).pb($3);

                        gdimv.pb(($3)->value);

                        $$ = t;

                }


br_list1 :              LBP exp_type_1 RBP

                {

                        ptr * t = new ptr;

                        t->tag = "BRLIST1";

                        t->gScope = gScope;

                        (t->children).pb($2);
                        t->value=1;

                        gdimv2d.pb(gdimv);

                        gdimv.clear();

                        gdimv.pb(($2)->value);

                        $$ = t;

                }
                        | br_list1 LBP exp_type_1 RBP

                {

                        ptr * t = new ptr;

                        t->tag = "BRLIST1";

                        t->gScope = gScope;

                        (t->children).pb($1);

                        (t->children).pb($3);

                        t->value=$1->value+1;

                        gdimv.pb(($3)->value);

                        $$ = t;

                }

func_decl :     type id lbf decl_plist RB

        {

                if(FuncTable.find($2->svalue)==FuncTable.end()  )

                {

                        func * f  = new func;

                        f->numparam = gparams.size();
```

```
                    f->returntype = $1->dtype;

                    f->params = gparams;

                    f->name = $2->svalue;

                    FuncTable[$2->svalue] = f;

                }

                else{

                    cout << "Semantic Error : Multiple functions have the same name" <<
$2->svalue << "in lineno. "<< yylineno<< endl;

                    semanticERROR = 1;

                }

                gparams.clear();

            }
                                lcf body rcf func_end

                    {

                        ptr * t = new ptr;

                        t->tag = "FUNCDECL";

                        (t->children).pb($1);

                        (t->children).pb($2);

                        (t->children).pb($3);

                        (t->children).pb($4);

                        (t->children).pb($7);

                        (t->children).pb($8);

                        (t->children).pb($9);

                        (t->children).pb($10);
```

```
        }
                    t->gScope = gScope;
                    t->svalue = $2->svalue;
                    $$ = t;
                    funcList.pb(t);
                }
                        | void id lbf decl_plist RB
    {
        if(FuncTable.find($2->svalue)==FuncTable.end()  )
        {
            func * f  = new func;
            f->numparam = gparams.size();
            f->returntype = "void";
            f->params = gparams;
            f->name = $2->svalue;
            FuncTable[$2->svalue] = f;
        }
        else{
            cout << "Semantic Error : Multiple functions have the same name" <<
$2->svalue << "in lineno. "<< yylineno<< endl;
            semanticERROR = 1;
        }
        gparams.clear();
    }
                            lcf body rcf func_end
            {
                    ptr * t = new ptr;
                    t->tag = "FUNCDECL";
                    t->gScope = gScope;
                    (t->children).pb($2);
                    (t->children).pb($3);
                    (t->children).pb($4);
                    (t->children).pb($7);
                    (t->children).pb($8);
                    (t->children).pb($9);
                    (t->children).pb($10);
                    t->svalue = $2->svalue;
                    $$ = t;
                    funcList.pb(t);
            }
lbf :                       LB
        {
                    ptr * t = new ptr;
                    t->tag = "LBF";
                    t->gScope = gScope;
                    $$ = t;
                    gScope++;
                    map< string , variable* > mp;
```

```
                                        SymTable.push_back(mp);
                              }

lcf :                                   LC
                              {
                                        ptr * t = new ptr;
                                        t->tag = "LCF";
                                        t->gScope = gScope;
                                        $$ = t;
                                        gScope++;

                                        map< string , variable* > mp;
                                        SymTable.push_back(mp);
                              }
rcf :                                   RC
                              {
                                        ptr * t = new ptr;
                                        t->tag = "RCF";
                                        t->gScope = gScope;
                                        $$ = t;
                                        gScope--;
                                        SymTable.pop_back();
                              }
```

```
func_end :
                    {

                              ptr * t = new ptr;

                              t->tag = "FUNCEND";

                              t->gScope = gScope;

                              $$ = t;

                              gScope--;

                              SymTable.pop_back();

                         }

decl_plist :             {activeFunc = gid; returnType = gtype; } decl_pl


                         {

                              ptr * t = new ptr;

                              t->gScope = gScope;

                              t->tag = "DECLPLIST";

                              (t->children).pb($2);

                              $$ = t;

                         }
                                   |

                         {

                              ptr * t = new ptr;

                              t->gScope = gScope;

                              t->tag = "DECLPLIST";
                    }

                              $$ = t;

                              activeFunc = gid;

                              returnType = gtype;

                    }

decl_pl :                     decl_param COMMA decl_pl

                    {

                              ptr * t = new ptr;

                              t->tag = "DECLPL";

                              t->gScope = gScope;

                              (t->children).pb($1);

                              (t->children).pb($3);

                              $$ = t;

                    }

                                        | decl_param


                    {

                              ptr * t = new ptr;

                              t->gScope = gScope;

                              t->tag = "DECLPL";

                              (t->children).pb($1);

                              $$ = t;

                    }

decl_param :        type var                                                        {
```

```
                ptr * t = new ptr;
                t->tag = "DECLPARAM";
                t->gScope = gScope;
                (t->children).pb($1);
                (t->children).pb($2);
                gparams.push_back(gvar);
                $$ = t;
            }
body:           stmts

            {
                ptr * t = new ptr;
                t->gScope = gScope;
                t->tag = "BODY";
                (t->children).pb($1);
                $$ = t;
            }
                    |

            {
                ptr * t = new ptr;
                t->gScope = gScope;
                t->tag = "BODY";
                $$ = t;
```

```
                    }
stmts :                         stmt stmts

                    {
                                ptr * t = new ptr;
                                t->gScope = gScope;
                                t->tag = "STMTS";
                                (t->children).pb($1);
                                (t->children).pb($2);
                                $$ = t;
                    }
                                        | stmt

                    {
                                ptr * t = new ptr;
                                t->tag = "STMTS";
                                t->gScope = gScope;
                                (t->children).pb($1);
                                $$ = t;
                    }
stmt:                           var_decl

                    {
                                ptr * t = new ptr;
```

```
          ;

}

          t->tag = "STMTVARDECL";

          t->gScope = gScope;

          (t->children).pb($1);

          $$ = t;

}
                    | exp semi

{

          ptr * t = new ptr;

          t->tag = "STMTEXP";

          t->gScope = gScope;

          (t->children).pb($1);

          $$ = t;

}
                    | exp_type_1 semi

{

          ptr * t = new ptr;

          t->tag = "STMTEXP";

          t->gScope = gScope;

          (t->children).pb($1);

          $$ = t;

}
```

```
                                        | FOR LB exp semi for_exp semi for_exp RB lcf body
rcf
                        {
                                ptr * t = new ptr;

                                t->tag = "FOREXP";
                                t->gScope = gScope;
                                (t->children).pb($3);
                                (t->children).pb($5);
                                (t->children).pb($7);
                                (t->children).pb($9);
                                (t->children).pb($10);
                                (t->children).pb($11);
                                $$ = t;
                        }
                                        | WHILE LB exp_type_1 RB lcf body rcf

                        {
                                ptr * t = new ptr;
                                t->tag = "WHILEEXP";
                                t->gScope = gScope;
                                (t->children).pb($3);
                                (t->children).pb($5);
                                (t->children).pb($6);
                                (t->children).pb($7);
```

```
            $$ = t;
    }
                | IF LB exp_type_1 RB lcf body rcf ELSE lcf body rcf
    {
            ptr * t = new ptr;
            t->tag = "IFELSEEXP";
            t->gScope = gScope;
            (t->children).pb($3);
            (t->children).pb($5);
            (t->children).pb($6);
            (t->children).pb($7);
            (t->children).pb($9);
            (t->children).pb($10);
            (t->children).pb($11);
            $$ = t;
    }
                | IF LB exp_type_1 RB lcf body rcf

    {
            ptr * t = new ptr;
            t->tag = "IFEXP";
            t->gScope = gScope;
            (t->children).pb($3);
            (t->children).pb($5);
```

```
                    }
                        (t->children).pb($6);
                        (t->children).pb($7);
                        $$ = t;
                    }
                            | SWITCH LB exp_type_1 RB LC case_exp default_exp
RC
                    {
                        ptr * t = new ptr;
                        t->tag = "SWITCHEXP";
                        t->gScope = gScope;
                        (t->children).pb($3);
                        (t->children).pb($6);
                        (t->children).pb($7);
                        $$ = t;
                    }
                            | continue semi
                    {
                        ptr * t = new ptr;
                        t->tag = "CONTINUEEXP";
                        t->gScope = gScope;
                        (t->children).pb($1);
                        $$ = t;


                    }
                            | break semi

                    {

                        ptr * t = new ptr;

                        t->tag = "STMTBREAK";

                        t->gScope = gScope;

                        (t->children).pb($1);

                        $$ = t;

                    }
                            | return_exp semi

                    {

                        ptr * t = new ptr;

                        t->tag = "STMTRETURN";

                        t->gScope = gScope;

                        (t->children).pb($1);

                        if(returnType!=($1->dtype)){

                            cout<<"Semantic Error : Return type does not match
function return type in line no. "<<yylineno<<"\n";

                        }

                        $$ = t;

                    }
                            | lcf body rcf

                    {

                        ptr * t = new ptr;

                        t->tag = "STMTBODY";
```

```
                        t->gScope = gScope;

                        (t->children).pb($1);

                        (t->children).pb($2);

                        (t->children).pb($3);

                        $$ = t;

                    }
                            | PRINT LB args1 RB semi                      {

                        ptr * t = new ptr;

                        t->tag = "PRINTEXP";

                        t->gScope = gScope;

                        (t->children).pb($3);

                        $$ = t;

                    }
                            | READ LB args RB semi                        {

                        ptr * t = new ptr;

                        t->tag = "READEXP";

                        t->gScope = gScope;

                        (t->children).pb($3);

                        $$ = t;

                    }
                            | error  SEMI
            {yyerrok; syntaxERROR = 1;}
                            | error RC
                {yyerrok; syntaxERROR = 1;}

for_exp:                exp_type_1
```

```
                {
                        ptr * t = new ptr;
                        t->tag = "FOREXPERR";
                        t->gScope = gScope;
                        (t->children).pb($1);
                        $$ = t;
                }
                            |

                {
                        ptr * t = new ptr;
                        t->tag = "FOREXPERR";
                        t->gScope = gScope;
                        $$ = t;
                }
semi:                   SEMI
            {}
                            |       error SEMI
                        {yyerrok; syntaxERROR = 1;}

args1   :   args_list1
                {
                        ptr * t = new ptr;
                        t->tag = "ARGS1";
                        t->gScope = gScope;
```

```
                    (t->children).pb($1);

                    $$ = t;

                }
            |


                {

                    ptr * t = new ptr;

                    t->gScope = gScope;

                    t->tag = "ARGS1";

                    $$ = t;

                }
            ;

args_list1      :       args_list1 COMMA arith_exp_type_1

                {

                    ptr * t = new ptr;

                    t->tag = "ARGSLIST1";

                    t->svalue = "1";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    (t->children).pb($3);

                    $$ = t;

                }
                |       arith_exp_type_1


                {
```

```
            ptr * t = new ptr;

            t->tag = "ARGSLIST1";

            t->svalue = "2";

            t->gScope = gScope;

            (t->children).pb($1);

            $$ = t;

    }
    | args_list1 COMMA string

    {

            ptr * t = new ptr;

            t->tag = "ARGSLIST1";

            t->svalue = "3";

            t->gScope = gScope;

            (t->children).pb($1);

            (t->children).pb($3);

            $$ = t;

    }
    |       string

    {

            ptr * t = new ptr;

            t->tag = "ARGSLIST1";

            t->svalue = "4";

            t->gScope = gScope;

            (t->children).pb($1);
```

```
                                        $$ = t;

                                }

string :                        STRING                          {

                        ptr * t = new ptr;

                        t->tag = "STRING";

                        t->gScope = gScope;

                        t->svalue = yylval.stringVal;

                        $$ = t;

        }

break:                                  BREAK


                                {

                                        ptr * t = new ptr;

                                        t->tag = "BREAK";

                                        t->gScope = gScope;

                                        $$ = t;

                                }

continue:                       CONTINUE


                                {

                                        ptr * t = new ptr;

                                        t->gScope = gScope;

                                        t->tag = "CONTINUE";

                                        $$ = t;
```

```
                    }
case_exp :                  CASE LB arith_exp_type_1 RB COLON lcf stmts rcf case_exp

                    {

                            ptr * t = new ptr;

                            t->gScope = gScope;

                            t->tag = "CASEEXP";

                            (t->children).pb($3);

                            (t->children).pb($6);

                            (t->children).pb($7);

                            (t->children).pb($8);

                            (t->children).pb($9);

                            $$ = t;

                    }
                                    |


                    {

                            ptr * t = new ptr;


                            t->gScope = gScope;

                            t->tag = "CASEEXP";

                            $$ = t;

                    }

default_exp :       DEFAULT COLON lcf stmts rcf

                    {

                            ptr * t = new ptr;
```

```
                                        ;

                                }

                                        t->gScope = gScope;

                                        t->tag = "DEFAULTEXP";

                                        (t->children).pb($3);

                                        (t->children).pb($4);

                                        (t->children).pb($5);

                                        $$ = t;

                                }
                                        |

                                {

                                        ptr * t = new ptr;

                                        t->gScope = gScope;

                                        t->tag = "DEFAULTEXP";

                                        $$ = t;

                                }

return_exp      :                       RETURN

                                {

                                        ptr * t = new ptr;

                                        t->gScope = gScope;

                                        t->tag = "RETURN";

                                        t->dtype = "void";

                                        $$ = t;

                                }
```

```
                    |           RETURN exp_type_1

            {

                    ptr * t = new ptr;

                    t->gScope = gScope;

                    t->tag = "RETURN";

                    (t->children).pb($2);

                    t->dtype = ($2->dtype);

                    $$ = t;

            }
                    ;


exp :               id EQUAL exp_type_1


            {

                    int scp = findScope($1->svalue);



                    if(scp==-1)

                    {

                            cout << "Semantic Error : Variable "<< $1->svalue <<"
is not declared in lineno. "<< yylineno << endl;

                            semanticERROR = 1;

                    }


                    else{

                            if(SymTable[scp][$1->svalue]->dtype=="int" &&
$3->dtype=="float" )
```

```
                                        {
                                                cout << "Semantic Error : Invalid data type
assignment in lineno. " << yylineno << endl;
                                                semanticERROR = 1;
                                        }
                                }

                                ptr * t = new ptr;
                                t->tag = "EXP";
                                t->gScope = scp;
                                (t->children).pb($1);
                                (t->children).pb($3);
                                $$ = t;
                        }
                                | id br_list1 EQUAL exp_type_1
                                                                        {
                                ptr * t = new ptr;
                                int scp = findScope($1->svalue);
                                if(scp == -1){
                                        cout<<"Semantic Error : Array not declared in lineno.
"<< yylineno << endl;
                                        semanticERROR = 1;
                                }
                                else{
                                        if(!SymTable[scp][$1->svalue]->array)

                                        {

                                                cout << "Semantic Error : Variable is not of
array type in lineno. " << yylineno << endl;

                                                semanticERROR = 1;

                                        }

                                        else if(SymTable[scp][$1->svalue]->dtype=="int" &&
$4->dtype=="float" ){

                                                cout << "Semantic Error : Invalid data type
assignment in lineno. " << yylineno << endl;

                                                semanticERROR = 1;

                                        }else
if($2->value!=SymTable[scp][$1->svalue]->dim.size()){

                                                cout<< "Semantic Error : Invalid dimensions of
array " << $1->svalue << " in lineno " << yylineno <<  endl;

                                                semanticERROR = 1;



                                        }

                                        else
if(checkOutofBound(SymTable[scp][$1->svalue]->dim))

                                        {

                                                cout << "Semantic Error : Out Of Bound array "
<< $1->svalue << " in lineno. " << yylineno << endl;

                                        }

                                        else{

                                                t->dimptrorg =
SymTable[scp][$1->svalue]->dim;

                                        t->dtype = SymTable[scp][$1->svalue]->dtype;

                                   }

                              }

                              t->tag = "EXP";

                              t->gScope = gScope;

                              (t->children).pb($1);

                              (t->children).pb($2);

                              (t->children).pb($4);t->dimptr=gdimv;

                                        gdimv = gdimv2d.back();

                                        gdimv2d.pop_back();

                              $$ = t;

                         }

exp_type_1     :              exp_type_1 OR exp_type_2

                         {

                              ptr * t = new ptr;

                              t->tag = "EXPTYPE1";

                              t->gScope = gScope;

                              (t->children).pb($1);

                              (t->children).pb($3);

                              t->dtype = decideintfloat($1->dtype , $3->dtype);

                              t->value = 0;

                              $$ = t;
```

```
                    }
                                        |         exp_type_2

                    {

                              ptr * t = new ptr;

                              t->tag = "EXPTYPE1";

                              t->gScope = gScope;

                              (t->children).pb($1);

                              t->dtype = $1->dtype;

                              t->value = $1->value;

                              $$ = t;

                    }

exp_type_2     :    exp_type_2 AND exp_type_3                              {

                              ptr * t = new ptr;

                              t->tag = "EXPTYPE2";

                              t->gScope = gScope;

                              (t->children).pb($1);

                              (t->children).pb($3);

                              t->dtype = decideintfloat($1->dtype , $3->dtype);

                              t->value = 0;

                              $$ = t;

                    }
                              |         exp_type_3


                    {
```

```
                    ptr * t = new ptr;

                    t->tag = "EXPTYPE2";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    t->dtype = $1->dtype;

                    t->value = $1->value;

                    $$ = t;

            }
                    ;

exp_type_3    :                     exp_type_3 relation_op arith_exp_type_1

            {


                    ptr * t = new ptr;

                    t->tag = "EXPTYPE3";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    (t->children).pb($2);

                    (t->children).pb($3);

                    t->dtype = decideintfloat($1->dtype , $3->dtype);

                    if($1->dtype!="int" || $3->dtype!="int"){

                        cout<<"Semantic Error : Relation operator used with
non-integer type in lineno. "<< yylineno <<endl;

                        semanticERROR=1;

                    }
                    t->value = 0;

                    $$ = t;


          }
                              |          arith_exp_type_1


          {

                    ptr * t = new ptr;

                    t->tag = "EXPTYPE3";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    t->dtype = $1->dtype;

                    t->value = $1->value;

                    $$ = t;

          }
                              ;

arith_exp_type_1    :         arith_exp_type_1 plus_minus_op arith_exp_type_2

          {

                    ptr * t = new ptr;

                    t->tag = "ARITHEXPTYPE1";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    (t->children).pb($2);

                    (t->children).pb($3);
                    t->dtype = decideintfloat($1->dtype , $3->dtype);

                    t->value = 0;

                    $$ = t;

                }
                          |         arith_exp_type_2

                {

                    ptr * t = new ptr;

                    t->tag = "ARITHEXPTYPE1";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    t->dtype = $1->dtype;

                    t->value = $1->value;

                    $$ = t;

                }
                          ;

arith_exp_type_2    :   arith_exp_type_2 mul_div_op unary_exp

                {

                    ptr * t = new ptr;

                    t->tag = "ARITHEXPTYPE2";

                    t->gScope = gScope;

                    (t->children).pb($1);

                    (t->children).pb($2);

                    (t->children).pb($3);

                    t->dtype = decideintfloat($1->dtype , $3->dtype) ;
```

```
                                        t->value = 0;

                                        $$ = t;

                                }
                                                |           unary_exp


                                {

                                        ptr * t = new ptr;

                                        t->tag = "ARITHEXPTYPE2";

                                        t->gScope = gScope;

                                        (t->children).pb($1);

                                        t->dtype = $1->dtype;

                                        t->value = $1->value;

                                        $$ = t;

                                }
                                                ;

unary_exp       :               unary_operator term

                                {

                                        ptr * t = new ptr;

                                        t->tag = "UNARYEXP";

                                        t->gScope = gScope;

                                        (t->children).pb($1);

                                        (t->children).pb($2);

                                        t->dtype = $2->dtype;

                                        t->value = 0;
```

```
                        $$ = t;

                }

                        |       term

                {

                        ptr * t = new ptr;

                        t->tag = "UNARYEXP";

                        t->gScope = gScope;

                        (t->children).pb($1);

                        t->dtype = $1->dtype;

                        t->value = $1->value;

                        $$ = t;

                }
                        ;

term    :       LB exp_type_1 RB


                {

                        ptr * t = new ptr;

                        t->tag = "TERM";

                        t->gScope = gScope;

                        (t->children).pb($2);

                        t->value = $2->value;

                        t->dtype = $2->dtype;

                        $$ = t;

                }
```

```
|       func_call

        {
                ptr * t = new ptr;
                t->tag = "TERM";
                t->gScope = gScope;
                (t->children).pb($1);
                t->dtype = $1->dtype;
                t->value = 0;
                $$ = t;
        }
|       consts

        {
                ptr * t = new ptr;
                t->tag = "TERM";
                t->gScope = gScope;
                t->dtype = $1->dtype;
                t->value = $1->value;
                (t->children).pb($1);
                $$ = t;
        }
|       id
        {
                ptr * t = new ptr;
```

```
                    t->gScope = gScope;

                    int scp = findScope(gid);



                    if(scp==-1)

                    {

                         cout << "Semantic Error : Variable "<<$1->svalue<< " is
not declared in lineno. " << yylineno << endl;

                         semanticERROR = 1;

                    }


                    else{

                         t->dtype = SymTable[scp][gid]->dtype;

                    }


                    t->tag = "TERM";

                    (t->children).pb($1);

                    t->value = 0;

                    $$ = t;

               }
          |   id br_list1

               {

                    ptr * t = new ptr;

                    int scp = findScope($1->svalue);

                    if(scp == -1){;
```

```
                                        cout<<"Semantic Error : Array "<<$1->svalue<< " not
declared in line no." << yylineno <<endl;

                                        semanticERROR = 1;

                                }

                                else{

                                        if(!SymTable[scp][$1->svalue]->array){

                                                cout << "Semantic Error : Variable is not of
array type in lineno. " << yylineno << endl;

                                                semanticERROR = 1;

                                        }else
if($2->value!=SymTable[scp][$1->svalue]->dim.size()){

                                                cout << "Semantic Error : Invalid dimension of
array " << $1->svalue << " in lineno. " << yylineno << endl;

                                                semanticERROR = 1;



                                        }

                                        else
if(checkOutofBound(SymTable[scp][$1->svalue]->dim))

                                        {

                                                cout << "Semantic Error : Out Of Bound array "
<< $1->svalue << " in lineno. " << yylineno << endl;

                                        }

                                        else{

                                                t->dimptrorg =
SymTable[scp][$1->svalue]->dim;

                                                t->dtype = SymTable[scp][$1->svalue]->dtype;
                                        }

                                }

                                t->tag = "TERM";

                                t->gScope = gScope;

                                (t->children).pb($1);

                                (t->children).pb($2);

                                t->dimptr=gdimv;

                                gdimv = gdimv2d.back();

                                gdimv2d.pop_back();

                                t->value = 0;

                                $$ = t;


                        }


                ;

func_call       :       id LB args RB

                        {

                                ptr * t = new ptr;

                                t->tag = "FUNCCALL";

                                t->gScope = gScope;

                                (t->children).pb($1);

                                (t->children).pb($3);


                                if(FuncTable.find($1->svalue)==FuncTable.end())
```

```
                                        {
                                                cout << "Semantic Error : " << $1->svalue << " function
is not declared in lineno. " << yylineno << endl;
                                                semanticERROR =1;
                                        }
                                        else{
                                                func * f = FuncTable[$1->svalue];
                                                t->dtype = f->returntype;
                                                if(f->numparam == gfcallparam2d.back().size() )
                                                {
                                                        for(int j = 0 ; j< f->numparam ; j++ )
                                                        {
                                                                if(f->params[j]->dtype !=
gfcallparam2d.back()[j])
                                                                {
                                                                        cout << "Semantic Error :
Datatype mismatched in parameters in line no. " << yylineno << endl;
                                                                        semanticERROR = 1;
                                                                }
                                                        }
                                                }
                                                else{
                                                        cout << "Semantic Error : No. of parameters not
matched in line no. " << yylineno << endl;
                                                        semanticERROR = 1;
```

```
                                    }
                                }
                                gfcallparam2d.pop_back();
                                $$ = t;
                            }
                                ;
args    :       args_list
                            {
                                ptr * t = new ptr;
                                t->tag = "ARGS";
                                t->gScope = gScope;
                                (t->children).pb($1);
                                $$ = t;
                            }
                |

                            {
                                ptr * t = new ptr;
                                t->gScope = gScope;
                                t->tag = "ARGS";
                                gfcallparam.clear();
                                gfcallparam2d.pb(gfcallparam);
                                $$ = t;
                            }
                ;
```

```
            }


args_list       :       args_list COMMA arith_exp_type_1

                        {

                                ptr * t = new ptr;

                                t->tag = "ARGSLIST";

                                t->gScope = gScope;

                                (t->children).pb($1);

                                (t->children).pb($3);

                                gfcallparam2d.back().pb($3->dtype);

                                $$ = t;

                        }
                        |       arith_exp_type_1


                        {

                                ptr * t = new ptr;

                                t->tag = "ARGSLIST";

                                t->gScope = gScope;

                                (t->children).pb($1);

                                gfcallparam.clear();

                                gfcallparam2d.pb(gfcallparam);

                                gfcallparam2d.back().pb($1->dtype);

                                $$ = t;

                        }

consts  :       intg
```

```
                {

                        ptr * t = new ptr;

                        t->gScope = gScope;

                        t->tag = "CONSTS";

                        (t->children).pb($1);

                        t->dtype = "int";

                        t->value = $1->value;

                        $$ = t;

                }
                | floats

                {

                        ptr * t = new ptr;

                        t->gScope = gScope;

                        t->tag = "CONSTS";

                        (t->children).pb($1);

                        t->dtype = "float";

                        t->value = $1->value;

                        $$ = t;

                }

intg  :   INTEGERS
                                                                    {

        ptr * t = new ptr;

        t->gScope = gScope;

        t->tag = "INTG";
```

```
          t->value = stof( yylval.stringVal );

          t->dtype = "int";

          $$ = t;
                                                                                                                  }
                              | SUB INTEGERS
                                                                                                                  {

          ptr * t = new ptr;

          t->gScope = gScope;

          t->tag = "INTG";

          t->value = -1*stof( yylval.stringVal );

          t->dtype = "int";

          $$ = t;
                                                                                                                  }
floats  :             FLOATING_POINTS
                                                                                                                  {

          ptr * t = new ptr;

          t->gScope = gScope;

          t->tag = "FLOATS";

          t->dtype ="float";

          t->value = stof( yylval.stringVal );

          $$ = t;
                                                                                                                  }
                              |          SUB FLOATING_POINTS
                                                                                                                  {

          ptr * t = new ptr;

          t->gScope = gScope;
          t->tag = "FLOATS";

          t->dtype ="float";

          t->value = -1*stof( yylval.stringVal );

          $$ = t;
                                                                                    }

plus_minus_op          :          ADD
                                                                                    {

                    ptr * t = new ptr;

                    t->gScope = gScope;

                    t->tag = "PLUSMINUSOP";

                    t->svalue = "+";

                    $$ = t;
                                                                                    }
                              |          SUB
                                                                                    {

          ptr * t = new ptr;

          t->gScope = gScope;

          t->tag = "PLUSMINUSOP";

          t->svalue = "-";

          $$ = t;
                                                                                    }
                              ;

mul_div_op     :          MUL
                                                                                    {

          ptr * t = new ptr;

          t->gScope = gScope;
          t->tag = "MULDIVOP";

          t->svalue = "*";

          $$ = t;
                                                                                    }
                    |          DIV
                                                                                    {

          ptr * t = new ptr;

          t->gScope = gScope;

          t->tag = "MULDIVOP";

          t->svalue = "/";

          $$ = t;
                                                                                    }
                    ;

relation_op     :          GT
                                                                                    {

          ptr * t = new ptr;

          t->gScope = gScope;

          t->tag = "RELATIONOP";

          t->svalue =">";

          $$ = t;
                                                                                    }
                    |          LT
                                                                                    {

          ptr * t = new ptr;

          t->svalue = "<";

          t->gScope = gScope;

          t->tag = "RELATIONOP";
```

```
$$ = t;
                                                                                    }
            |       GE
                                                                                    {
ptr * t = new ptr;
t->svalue = ">=";
t->gScope = gScope;
t->tag = "RELATIONOP";
$$ = t;
                                                                                    }
            |       LE
                                                                                    {
ptr * t = new ptr;
t->svalue = "<=";
t->gScope = gScope;
t->tag = "RELATIONOP";
$$ = t;
                                                                                    }
            |       EQ
                                                                                    {
ptr * t = new ptr;
t->svalue = "==";
t->gScope = gScope;
t->tag = "RELATIONOP";
$$ = t;
                                                                                    }
            |       NE
                                                                                    {
ptr * t = new ptr;
```

```
        t->svalue = "!=";

        t->gScope = gScope;

        t->tag = "RELATIONOP";

        $$ = t;
                                                                                    }
                ;

unary_operator      :       SUB SUB
                                                                                    {

        ptr * t = new ptr;

        t->gScope = gScope;

        t->tag = "UNARYOPERATOR";

        t->svalue = "--";

        $$ = t;
                                                                                    }
                        |       ADD ADD
                                                                                    {

        ptr * t = new ptr;

        t->gScope = gScope;

        t->tag = "UNARYOPERATOR";

        t->svalue = "++";

        $$ = t;
                                                                                    }

%%

void printSpace(int cnt)
{
        for(int i=0;i<cnt;i++) cout<<"\t";
}

void PrintTree(ptr *n,int cnt)
```

```
{
        printSpace(cnt);
        if(n==NULL){
                return;
        }
        cout << n->tag << endl;
        for (int i = 0; i < (n->children).size(); ++i)
        {
                PrintTree((n->children)[i],cnt+1);
        }
}

string decideintfloat(string s1,string s2){
        if( s1 == "float" || s2 == "float"){
                return "float";
        }
        else{
                return "int";
        }
}

int checkOutofBound(vector<int> v){

        int n=gdimv.size();
        for(int i=0;i<n;i++){
                if(gdimv[i]>=v[i]){
                        return i+1;
                }
        }
        return 0;
}

void SymTablePrint()
{
        cout << "Sym Table" << endl;
        for(int g = 0; g < SymTable.size(); g++ )
        {
                for(auto i : SymTable[g])
                {
                        cout << g << "\t\t" << i.first << "\t\t" << i.second->dtype << "\t\t";

                        if(i.second->array)
                        {
                                cout << "array\t\t" ;
                                for(int i1 = 0; i1 < i.second->dim.size(); i1++ )
                                        cout << i.second->dim[i1] << " ";
```

```
                    cout << endl;
                }
                else{
                    cout << "single\t\t" << endl ;
                }
            }
        }
}

int findScope(string gid){
        for(int i=gScope;i>=0;i--){
            if(SymTable[i].find(gid)!=SymTable[i].end()){
                return i;
            }
        }
        return -1;
}

int temp = -1;

string getTemp(){
        temp++;
        string t = "temp_";
        t += to_string(temp);
        return t;
}

int label = -1;

string getLabel(){
        label++;
        string l = "label_";
        l += to_string(label);
        return l;
}

string generateCode(ptr * root){
        vector<ptr*> v = root->children;
        if(root->tag=="VARDECL"){
            if(root->children.size()==3){
                string val1 = generateCode(v[2]);
                fprintf(f, "%s.%s.%d%s = %s.%s\n", v[0]->dtype.c_str(),
v[1]->svalue.c_str(), v[1]->gScope, currFunc.c_str(),  v[2]->dtype.c_str(), val1.c_str());
                fprintf(q, " , %s.%s , , %s.%s.%d%s\n", v[2]->dtype.c_str(),
val1.c_str(), v[0]->dtype.c_str(), v[1]->svalue.c_str(), v[1]->gScope, currFunc.c_str());
```

```
		}
		else{
			string val1 = generateCode(v[1]);
		}
	}else if(root->tag=="VARLIST"){
		if(root->children.size()==2){
			string val1 = generateCode(v[1]);
		}
		else{
			string val1 = generateCode(v[0]);
		}
	}else if(root->tag=="EXP"){
		if(v.size()==2)
		{
			string val1 = generateCode(v[1]);
			if(v[0]->gScope>=1){
				fprintf(f, "%s.%s.%d%s = %s.%s\n", v[0]->dtype.c_str(),
v[0]->svalue.c_str(), v[0]->gScope, currFunc.c_str(), v[1]->dtype.c_str(), val1.c_str());
				fprintf(q, " , %s.%s , , %s.%s.%d%s\n", v[1]->dtype.c_str(),
val1.c_str(), v[0]->dtype.c_str(), v[0]->svalue.c_str(), v[0]->gScope, currFunc.c_str());
			}
			else{
				fprintf(f, "%s.%s.%d = %s.%s\n", v[0]->dtype.c_str(),
v[0]->svalue.c_str(), v[0]->gScope, v[1]->dtype.c_str(), val1.c_str());
				fprintf(q, " , %s.%s , , %s.%s.%d\n", v[1]->dtype.c_str(),
val1.c_str(), v[0]->dtype.c_str(), v[0]->svalue.c_str(), v[0]->gScope);
			}
		}
		else{
			string t = v[0]->svalue;
			vector<int> dimptrorg = root->dimptrorg;
			string val1 = generateCode(v[2]);
			vector<string> str;
			brlist.pb(str);
			generateCode(v[1]);
			vector<string> list = brlist.back();
			brlist.pop_back();
			string var1 = getTemp();
			fprintf(f, "int.%s = int.%s\n", var1.c_str(), list[0].c_str());
			fprintf(q, " , int.%s , , int.%s\n", list[0].c_str(), var1.c_str());
			for(int i=0;i<list.size()-1;i++){
				fprintf(f, "int.%s = int.%s * %d\n", var1.c_str(), var1.c_str(),
dimptrorg[i+1]);
				fprintf(q, " * , int.%s , %d , int.%s\n", var1.c_str(),
dimptrorg[i+1], var1.c_str());
```

```
                                        fprintf(f, "int.%s = int.%s + int.%s\n", var1.c_str(), var1.c_str(),
list[i+1].c_str());
                                        fprintf(q, " + , int.%s , int.%s , int.%s\n", var1.c_str(),
list[i+1].c_str(), var1.c_str());

                              }
                              if(v[0]->gScope>=1)   {
                                        fprintf(f, "%s.%s.%d%s(int.%s) = %s.%s\n",
root->dtype.c_str(), t.c_str(), v[0]->gScope, currFunc.c_str(), var1.c_str(), v[2]->dtype.c_str(),
val1.c_str());
                                        fprintf(q, " , %s.%s , , %s.%s.%d%s(int.%s)\n",
v[2]->dtype.c_str(), val1.c_str(), root->dtype.c_str(), t.c_str(), v[0]->gScope, currFunc.c_str(),
var1.c_str());
                              }else{
                                        fprintf(f, "%s.%s.%d(int.%s) = %s.%s\n", root->dtype.c_str(),
t.c_str(), v[0]->gScope, var1.c_str(), v[2]->dtype.c_str(), val1.c_str());
                                        fprintf(q, " , %s.%s , , %s.%s.%d(int.%s)\n", v[2]->dtype.c_str(),
val1.c_str(), root->dtype.c_str(), t.c_str(), v[0]->gScope, var1.c_str());
                              }
                    }
          }else if(root->tag=="BRLIST1"){
                    if(v.size()==1){
                              string t = generateCode(v[0]);
                              brlist.back().pb(t);
                    }else{
                              generateCode(v[0]);
                              string t = generateCode(v[1]);
                              brlist.back().pb(t);
                    }
          }else if(root->tag=="EXPTYPE1"){
                    if(v.size()==1)
                              return generateCode(v[0]);
                    else{
                              string var1 = generateCode(v[0]);
                              string l1 = getLabel();
                              string l2 = getLabel();
                              string l3 = getLabel();
                              string l4 = getLabel();
                              string t = getTemp();
                              fprintf(f, "int.%s = %s.%s <= 0\n", t.c_str(), v[0]->dtype.c_str(),
var1.c_str());
                              fprintf(q, " <= , %s.%s , 0 , int.%s\n", v[0]->dtype.c_str(), var1.c_str(),
t.c_str());

                              fprintf(f, "if int.%s goto %s\n", t.c_str(), l1.c_str());
                              fprintf(q, " if , int.%s , %s , goto\n", t.c_str(), l1.c_str());
```

```
				fprintf(f, "goto %s\n", l2.c_str());
				fprintf(q, " , %s , , goto\n", l2.c_str());

				fprintf(f, "%s:\n", l1.c_str());
				fprintf(q, "%s:\n", l1.c_str());

				string var2 = generateCode(v[1]);
				string t1 = getTemp();
				fprintf(f, "int.%s = %s.%s <= 0\n", t1.c_str(), v[1]->dtype.c_str(),
var2.c_str());
				fprintf(q, " <= , %s.%s , 0 , int.%s\n", v[1]->dtype.c_str(), var2.c_str(),
t1.c_str());

				fprintf(f, "if int.%s goto %s\n", t1.c_str(), l3.c_str());
				fprintf(q, "if , int.%s , %s , goto\n", t1.c_str(), l3.c_str());

				fprintf(f, "goto %s\n", l2.c_str());
				fprintf(q, " , %s , , goto\n", l2.c_str());

				fprintf(f, "%s:\n", l3.c_str());
				fprintf(q, "%s:\n", l3.c_str());

				fprintf(f, "int.%s = 0\n", t.c_str());
				fprintf(q, " , 0 , , int.%s\n", t.c_str());

				fprintf(f, "goto %s\n", l4.c_str());
				fprintf(q, " , %s , , goto\n", l4.c_str());

				fprintf(f, "%s:\n", l2.c_str());
				fprintf(q, "%s:\n", l2.c_str());

				fprintf(f, "int.%s = 1\n", t.c_str());
				fprintf(q, " , 1 , , int.%s\n", t.c_str());

				fprintf(f, "%s:\n", l4.c_str());
				fprintf(q, "%s:\n", l4.c_str());

				return t;
			}

		}else if(root->tag=="EXPTYPE2"){
			if(v.size()==1)
				return generateCode(v[0]);
			else{
				string var1 = generateCode(v[0]);
```

```
                string l1 = getLabel();
                string l2 = getLabel();
                string l3 = getLabel();
                string l4 = getLabel();
                string t = getTemp();
                fprintf(f, "int.%s = %s.%s > 0\n", t.c_str(), v[0]->dtype.c_str(),
var1.c_str());
                fprintf(q, " > , %s.%s , 0 , int.%s\n", v[0]->dtype.c_str(), var1.c_str(),
t.c_str());

                fprintf(f, "if int.%s goto %s\n", t.c_str(), l1.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", t.c_str(), l1.c_str());

                fprintf(f, "goto %s\n", l2.c_str());
                fprintf(q, " , %s , , goto\n", l2.c_str());

                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string var2 = generateCode(v[1]);
                string t1 = getTemp();
                fprintf(f, "int.%s = %s.%s > 0\n", t1.c_str(), v[1]->dtype.c_str(),
var2.c_str());
                fprintf(q, " > , %s.%s , 0 , int.%s\n", v[1]->dtype.c_str(), var2.c_str(),
t1.c_str());

                fprintf(f, "if int.%s goto %s\n", t1.c_str(), l3.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", t1.c_str(), l3.c_str());

                fprintf(f, "goto %s\n", l2.c_str());
                fprintf(q, " , %s , , goto\n", l2.c_str());

                fprintf(f, "%s:\n", l3.c_str());
                fprintf(q, "%s:\n", l3.c_str());

                fprintf(f, "int.%s = 1\n", t.c_str());
                fprintf(q, " , 1 , , int.%s\n", t.c_str());

                fprintf(f, "goto %s\n", l4.c_str());
                fprintf(q, " , %s , , goto\n", l4.c_str());

                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

                fprintf(f, "int.%s = 0\n", t.c_str());
                fprintf(q, " , 0 , , int.%s\n", t.c_str());
```

```
				fprintf(f, "%s:\n", l4.c_str());
				fprintf(q, "%s:\n", l4.c_str());

				return t;
			}
		}else if(root->tag=="EXPTYPE3" || root->tag=="ARITHEXPTYPE1" ||
root->tag=="ARITHEXPTYPE2"){
			if(v.size()==1){
				return generateCode(v[0]);
			}
			string var1 = getTemp();
			string val1 = generateCode(v[0]);
			string val2 = generateCode(v[2]);
			fprintf(f, "%s.%s = %s.%s %s %s.%s\n", root->dtype.c_str(), var1.c_str(),
v[0]->dtype.c_str(), val1.c_str(), v[1]->svalue.c_str(), v[2]->dtype.c_str(), val2.c_str());
			fprintf(q, " %s , %s.%s , %s.%s , %s.%s\n", v[1]->svalue.c_str(),
v[0]->dtype.c_str(), val1.c_str(), v[2]->dtype.c_str(), val2.c_str(), root->dtype.c_str(),
var1.c_str());

			return var1;

		}else if(root->tag=="UNARYEXP"){
			if(v.size()==1){
				return generateCode(v[0]);
			}
			string var1 = generateCode(v[1]);
			if(v[0]->svalue=="++"){
				fprintf(f, "%s.%s = %s.%s + 1\n", v[1]->dtype.c_str(), var1.c_str(),
v[1]->dtype.c_str(), var1.c_str() );

				fprintf(q, " + , %s.%s , 1 , %s.%s\n", v[1]->dtype.c_str(), var1.c_str(),
v[1]->dtype.c_str(), var1.c_str());


			}
			if(v[0]->svalue=="--"){
				fprintf(f, "%s.%s = %s.%s - 1\n", v[1]->dtype.c_str(), var1.c_str(),
v[1]->dtype.c_str(), var1.c_str() );
				fprintf(q, " - , %s.%s , 1 , %s.%s\n", v[1]->dtype.c_str(), var1.c_str(),
v[1]->dtype.c_str(), var1.c_str() );

			}
			return var1;

		}else if(root->tag=="TERM" || root->tag=="CONSTS"){
```

```
		if(v.size()==1){
			return generateCode(v[0]);
		}
		else{
			string t = v[0]->svalue;
			vector<int> dimptrorg = root->dimptrorg;
			vector<string> str;
			brlist.pb(str);
			generateCode(v[1]);
			vector<string> list = brlist.back();
			brlist.pop_back();
			string var1 = getTemp();
			fprintf(f, "int.%s = int.%s\n", var1.c_str(), list[0].c_str());
			fprintf(q, " , int.%s , , int.%s\n", list[0].c_str(),  var1.c_str());

			for(int i=0;i<list.size()-1;i++){
				fprintf(f, "int.%s = int.%s * %d\n", var1.c_str(), var1.c_str(),
dimptrorg[i+1]);
				fprintf(q, " * , int.%s , %d , int.%s\n", var1.c_str(),
dimptrorg[i+1], var1.c_str());

				fprintf(f, "int.%s = int.%s + int.%s\n", var1.c_str(), var1.c_str(),
list[i+1].c_str());
				fprintf(q, " + , int.%s , int.%s , int.%s\n", var1.c_str(),
list[i+1].c_str(), var1.c_str());

			}
			string temp;
			if(v[0]->gScope>=1){
				temp = t + "." + to_string(v[0]->gScope) + currFunc + "(int." +
var1 + ")";
			}else{
				temp = t + "." + to_string(v[0]->gScope) + "(int." + var1 + ")";
			}
			//	fprintf(f, "%s.%s = %s.%s.%d%s(int.%s)\n",
root->dtype.c_str(), temp.c_str(), root->dtype.c_str(), t.c_str(), v[0]->gScope,
currFunc.c_str(), var1.c_str());
			//else{
			//	fprintf(f, "%s.%s = %s.%s.%d(int.%s)\n", root->dtype.c_str(),
temp.c_str(),  root->dtype.c_str(), t.c_str(), v[0]->gScope, var1.c_str());
			//}
			return temp;
		}
	}else if(root->tag=="INTG" || root->tag=="FLOATS"){
		string var1 = getTemp();
		if(root->dtype=="int"){
```

```
                        int a = root->value;
                        fprintf(f, "int.%s = %d\n", var1.c_str(), a);
                        fprintf(q, " , %d , , int.%s\n", a, var1.c_str());

                }else{
                        fprintf(f, "float.%s = %f\n", var1.c_str(), root->value);
                        fprintf(q, " , %f , , float.%s\n", root->value, var1.c_str());

                }
                return var1;
        }else if(root->tag=="STMTBREAK" || root->tag=="STMTRETURN" ||
root->tag=="CONTINUEEXP" || root->tag=="STMTVARDECL" || root->tag=="STMTEXP"){
                return generateCode(v[0]);
        }else if(root->tag=="STMTBODY"){
                return generateCode(v[1]);
        }else if(root->tag=="FOREXP"){
                string init = generateCode(v[0]);
                string l1 = getLabel();
                string l2 = getLabel();
                string l3 = getLabel();
                string l4 = getLabel();
                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string cond = generateCode(v[1]);
                fprintf(f, "if int.%s goto %s\n", cond.c_str(), l2.c_str());
                fprintf(q, "if , int.%s , %s , goto\n", cond.c_str(), l2.c_str());

                fprintf(f, "goto %s\n", l3.c_str());
                fprintf(q, " , %s , , goto\n", l3.c_str());

                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

                brk.pb(l3);
                cont.pb(l4);
                string body = generateCode(v[4]);
                brk.pop_back();
                cont.pop_back();
                fprintf(f, "%s:\n", l4.c_str());
                fprintf(q, "%s:\n", l4.c_str());

                string itr = generateCode(v[2]);
                fprintf(f, "goto %s\n", l1.c_str());
                fprintf(q, " , %s , , goto\n", l1.c_str());
                fprintf(f, "%s:\n", l3.c_str());
                fprintf(q, "%s:\n", l3.c_str());

                return "";
        }else if(root->tag=="BREAK"){
                fprintf(f, "goto %s\n", brk[brk.size()-1].c_str());
                fprintf(q, " , %s , , goto\n", brk[brk.size()-1].c_str());

        }else if(root->tag=="CONTINUE"){
                fprintf(f, "goto %s\n", cont[cont.size()-1].c_str());
                fprintf(q, " , %s , , goto\n", cont[cont.size()-1].c_str());

        }else if(root->tag=="WHILEEXP"){
                string l1 = getLabel();
                string l2 = getLabel();
                string l3 = getLabel();
                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string cond = generateCode(v[0]);
                fprintf(f, "if int.%s goto %s\n", cond.c_str(), l2.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", cond.c_str(), l2.c_str());

                fprintf(f, "goto %s\n", l3.c_str());
                fprintf(q, " , %s , , goto\n", l3.c_str());

                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

                brk.pb(l3);
                cont.pb(l1);
                string body = generateCode(v[2]);
                brk.pop_back();
                cont.pop_back();
                fprintf(f, "goto %s\n", l1.c_str());
                fprintf(q, " , %s , , goto\n", l1.c_str());

                fprintf(f, "%s:\n", l3.c_str());
                fprintf(q, "%s:\n", l3.c_str());

                return "";

        }else if(root->tag=="ID"){
                string var1 = "";
                var1 += root->svalue;
                if(root->dimptr.size()){
                        vector<int> temp = root->dimptr;
                        for(int i=0;i<temp.size();i++){
                                var1 += ".";
                                var1 += to_string(temp[i]);
                        }
                }
                var1 += ".";
                var1 += to_string(root->gScope);
                if(root->gScope>=1){
                        var1 += currFunc;
                }
                return var1;
        }else if(root->tag=="IFEXP"){
                string l1 = getLabel();
                string l2 = getLabel();
                string cond = generateCode(v[0]);
                fprintf(f, "if int.%s goto %s\n", cond.c_str(), l1.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", cond.c_str(), l1.c_str());

                fprintf(f, "goto %s\n", l2.c_str());
                fprintf(q, " , %s , , goto\n", l2.c_str());

                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string body = generateCode(v[2]);
                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

        }else if(root->tag=="IFELSEEXP"){
                string l1 = getLabel();
                string l2 = getLabel();
                string l3 = getLabel();
                string cond = generateCode(v[0]);
                fprintf(f, "if int.%s goto %s\n", cond.c_str(), l1.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", cond.c_str(), l1.c_str());

                fprintf(f, "goto %s\n", l2.c_str());
                fprintf(q, " , %s , , goto\n", l2.c_str());

                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string body = generateCode(v[2]);
                fprintf(f, "goto %s\n", l3.c_str());
                fprintf(q, " , %s , , goto\n", l3.c_str());
                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

                string el = generateCode(v[5]);
                fprintf(f, "%s:\n", l3.c_str());
                fprintf(q, "%s:\n", l3.c_str());

        }else if(root->tag=="SWITCHEXP"){
                chk = generateCode(v[0]);
                brk.pb(getLabel());
                generateCode(v[1]);
                generateCode(v[2]);
                fprintf(f, "%s:", brk[brk.size()-1].c_str());
                fprintf(q, "%s:", brk[brk.size()-1].c_str());

                brk.pop_back();
        }else if(root->tag=="CASEEXP"){
                if(v.size()==0){
                        return "";
                }
                string var1 = generateCode(v[0]);
                string t1 = getTemp();
                string l1 = getLabel();
                string l2 = getLabel();
                fprintf(f, "int.%s = int.%s == int.%s\n", t1.c_str(), chk.c_str(), var1.c_str());
                fprintf(q, " == , int.%s , int.%s , int.%s\n", chk.c_str(), var1.c_str(), t1.c_str());

                fprintf(f, "if int.%s goto %s\n", t1.c_str(), l1.c_str());
                fprintf(q, " if , int.%s , %s , goto\n", t1.c_str(), l1.c_str());

                fprintf(f, "goto %s\n", l2.c_str());
                fprintf(q, " , %s , , goto\n", l2.c_str());

                fprintf(f, "%s:\n", l1.c_str());
                fprintf(q, "%s:\n", l1.c_str());

                string body = generateCode(v[2]);
                fprintf(f, "%s:\n", l2.c_str());
                fprintf(q, "%s:\n", l2.c_str());

                generateCode(v[4]);
                return "";
        }else if(root->tag=="DEFAULTEXP"){
                if(v.size()==0){
                        return "";
                }
                string body = generateCode(v[1]);
                return "";
        }else if(root->tag=="FUNCDECL"){
                return "";
        }else if(root->tag=="RETURN"){
                if(v.size()==0){
                        fprintf(f, "return NULL\n");
                        fprintf(q, " , NULL , , return\n");

                }else{
                        string var1 = generateCode(v[0]);
                        fprintf(f, "return %s.%s\n", v[0]->dtype.c_str() ,var1.c_str());
                        fprintf(q, " , %s.%s , , return\n", v[0]->dtype.c_str() ,var1.c_str());

                }
        }else if(root->tag=="FUNCCALL"){
                int temp = printFlag;
                printFlag = 0;
                string fName = v[0]->svalue;
                callFunc.pb(fName);
                generateCode(v[1]);
                callFunc.pop_back();
                fprintf(f, "call %s\n", fName.c_str());
                fprintf(q, " , %s , , call\n", fName.c_str());

                string var1 = "";
                if(FuncTable[fName]->returntype!="void"){
                        var1 = getTemp();
                        fprintf(f, "refparam %s.%s\n", FuncTable[fName]->returntype.c_str(),
var1.c_str());
                        fprintf(q, " , %s.%s , , refparam\n",
FuncTable[fName]->returntype.c_str(), var1.c_str());

                }
                printFlag = temp;
                return var1;
        }else if(root->tag=="PRINTEXP"){
                generateCode(v[0]);
                fprintf(f, "print \"\\n\" \n");
                fprintf(q, " , \"\\n\" , , print\n");

        }else if(root->tag=="ARGS1"){
                if(v.size()!=0){
                        vector<string> param;
                        para.pb(param);
                generateCode(v[0]);
                for(string s : para[para.size()-1]){
                        fprintf(f, "print %s\n", s.c_str());
                        fprintf(q, " , %s , , print\n", s.c_str());

                }
                para.pop_back();
                return "";
        }
}else if(root->tag=="ARGSLIST1"){
        if(root->svalue=="1"){
                generateCode(v[0]);
                string t = "";
                t += v[1]->dtype;
                t += ".";
                t += generateCode(v[1]);
                para[para.size()-1].pb(t);
        }else if(root->svalue=="2"){
                string t = "";
                t += v[0]->dtype;
                t += ".";
                t += generateCode(v[0]);
                para[para.size()-1].pb(t);
        }else if(root->svalue=="3"){
                generateCode(v[0]);
                string str = v[1]->svalue;
                string t = "";
                t += str;
                para[para.size()-1].pb(t);
        }else{
                string str = v[0]->svalue;
                string t = "";
                t += str;
                para[para.size()-1].pb(t);
        }
}else if(root->tag=="READEXP"){
        printFlag = 1;
        generateCode(v[0]);
        printFlag = 0;
}else if(root->tag=="ARGS"){
        if(v.size()!=0){
                vector<string> param;
                para.pb(param);
                generateCode(v[0]);
                for(string s : para[para.size()-1]){
                        if(!printFlag){
```

```
                                        fprintf(f, "param %s\n", s.c_str());
                                        fprintf(q, " , %s , , param\n", s.c_str());
                                }
                                else{
                                        fprintf(f, "read %s\n", s.c_str());
                                        fprintf(q, " , %s , , read\n", s.c_str());
                                }

                        }
                        para.pop_back();
                        return "";
                }
        }else if(root->tag=="ARGSLIST"){
                if(v.size()==1){
                        string t = "";
                        t += v[0]->dtype;
                        t += ".";
                        t += generateCode(v[0]);
                        para[para.size()-1].pb(t);
                }else{
                        generateCode(v[0]);
                        string t = "";
                        t += v[1]->dtype;
                        t += ".";
                        t += generateCode(v[1]);
                        para[para.size()-1].pb(t);
                }
        }else if(root->tag=="VARARRAY"){
                string t = v[0]->svalue;
                int a = 1;
                vector<int> temp = root->dimptr;
                for(int i=0;i<temp.size();i++)
                {
                        a = a*temp[i];
                }
                if(v[0]->gScope>=1)    {
                        fprintf(f, "decl %s.%s.%d%s(%d)\n", v[0]->dtype.c_str(), t.c_str(),
v[0]->gScope, currFunc.c_str(),a );
                        fprintf(q, " , %s.%s.%d%s(%d) , , decl\n", v[0]->dtype.c_str(), t.c_str(),
v[0]->gScope, currFunc.c_str(),a );

                }else{
                        fprintf(f, "decl %s.%s.%d(%d)\n", v[0]->dtype.c_str(), t.c_str(),
v[0]->gScope,a );
                        fprintf(q, " , %s.%s.%d(%d) , , decl\n", v[0]->dtype.c_str(), t.c_str(),
v[0]->gScope,a );
```

```
                }
        }else if(root->tag=="FOREXPERR"){
                if(v.size()!=0){
                        return generateCode(v[0]);
                }
        }
        else{
                for(int i=0;i<root->children.size();i++){
                        generateCode(root->children[i]);
                }
        }
        return "";
}

void generateFunc(ptr * root){
        fprintf(f, "func begin %s\n", root->svalue.c_str());
        fprintf(q, " begin , func , %s , \n", root->svalue.c_str());

        vector< variable * > v = FuncTable[root->svalue]->params;
        for(variable * var : v){
                fprintf(f, "args %s.%s.%d.%s\n", var->dtype.c_str(), var->name.c_str(),
var->scope, root->svalue.c_str());
                fprintf(q, " , %s.%s.%d.%s , , args\n", var->dtype.c_str(), var->name.c_str(),
var->scope, root->svalue.c_str());

        }
        currFunc = "." + root->svalue;
        if(root->children.size()==7){
                generateCode(root->children[4]);
        }else{
                generateCode(root->children[5]);
        }
        currFunc = "";

        fprintf(f, "func end\n");
        fprintf(q, " end , func , , \n");

}

int main(){

                map< string , variable* > mp;
                SymTable.pb(mp);
                yyparse();
                // PrintTree(treeRoot,0);
```

```
		//SymTablePrint();

		if(semanticERROR || syntaxERROR)
		{
			cout << "" << endl;
		}
		else
		{
			fprintf(q, " operator , arg1 , arg2 , result\n");
			//SymTablePrint();
			for( ptr * p : funcList)
				generateFunc(p);
			fprintf(f, "func begin main\n");
			fprintf(q, " begin , func , main , \n");

			generateCode(treeRoot);
			fprintf(f, "func end\n");
			fprintf(q, " end , func , , \n");

		}
}
```

Parser2.y

```
%{
#define YYSTYPE char *
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <map>
#include <sstream>
#include <vector>
#include <bits/stdc++.h>
using namespace std;

vector <string> allVar;

int yylex(void);

void yyerror (char const *s) {
	fprintf (stderr, "%s\n", s);
}

FILE *user_code, *final_code;
extern char *yytext;
int labelID=0;
int globArgsIntReg=0;
int globArgsFloReg=6;
int stringCounter=0;


int stringType(string);
void add_operation(char*, char *, char *);
void sub_operation(char*, char *, char *);
void mul_operation(char*, char *, char *);
void div_operation(char*, char *, char *);
void less_than_op(char*, char *, char *);
void great_than_op(char*, char *, char *);
void equal_op(char*, char *, char *);
void less_eq_op(char*, char *, char *);
void great_eq_op(char*, char *, char *);
void not_eq_op(char*, char *, char *);
void checkNewDeclare(char *);
char * getArrayParam(char *);
char * getArrayName(char *);
%}

%start funcs
%token INT FLOAT ID EQ DECL
%token ARITH_REL_OPS
%token IF GOTO LABEL PRINTT STRINGG READD
%token FUNC BEGINN RETURN END PARAM REFPARAM CALL ARGS NULLL
%%

funcs:                  func funcs {}
                            | func {}

func:                   FUNC BEGINN funcname intm_code FUNC END
                            {
                                fprintf(user_code,"jr $ra\n");
                            }
funcname:               var_ID
                            {
                                $$ = $1;
                                fprintf(user_code, "\n%s:\n", $$);
                            }

intm_code:          /* empty */
                    | intm_code intm_line /* do nothing */
```

```
intm_line:                    binary_operation {globArgsFloReg = 6; globArgsIntReg = 0;}
                                    | assignment {globArgsFloReg = 6; globArgsIntReg =
0;}
                                    | jump_Cond {globArgsFloReg = 6; globArgsIntReg =
0;}
                                    | jump_unCond {globArgsFloReg = 6; globArgsIntReg =
0;}
                                    | label{globArgsFloReg = 6; globArgsIntReg = 0;}

                                    | arr_decl_stmt {globArgsFloReg = 6; globArgsIntReg =
0;}
                                    | args_stmt
                                    | param_stmt
                                    | refparam_stmt {globArgsFloReg = 6; globArgsIntReg
= 0;}
                                    | call_stmt {globArgsFloReg = 6; globArgsIntReg = 0;}
                                    | return_stmt {globArgsFloReg = 6; globArgsIntReg =
0;}
                                    | print_stmt {globArgsFloReg = 6; globArgsIntReg = 0;}
                                    | scan_stmt {globArgsFloReg = 6; globArgsIntReg = 0;}

scan_stmt:                    READD var_ID
                                    {
                                            string opr($2);
                                            if(stringType(opr)==3){
                                                    string xx(getArrayParam($2));
                                                    char * zz = getArrayParam($2);
                                                    char * yy = getArrayName($2);
                                                    if(stringType(xx)==0)
                                                            fprintf(user_code,"lw $t3, %s\n",
zz);
                                                    else
                                                            fprintf(user_code,"li $t3, %s\n",
zz);
                                                    fprintf(user_code, "la $t4, %s\n", yy);

                                                    if(yy[0]=='f'){
                                                            fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                            fprintf(user_code, "add $t4, $t4,
$t3\n");


                                                            fprintf(user_code, "li $v0, 6\n");
                                                            fprintf(user_code, "syscall\n");
                                                            fprintf(user_code, "s.s $f0,
0($t4)\n");
```

```
                                                }
                                                else{
                                                        fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                        fprintf(user_code, "add $t4, $t4,
$t3\n");


                                                        fprintf(user_code, "li $v0, 5\n");
                                                        fprintf(user_code, "syscall\n");
                                                        fprintf(user_code, "sw $v0,
0($t4)\n");
                                                }
                                        }
                                        else if(stringType(opr)==0){
                                                checkNewDeclare($2);
                                                if(opr[0]=='f'){
                                                        fprintf(user_code, "li $v0, 6\n");
                                                        fprintf(user_code, "syscall\n");
                                                        fprintf(user_code, "s.s $f0, %s\n",
$2);
                                                }
                                                else{
                                                        fprintf(user_code, "li $v0, 5\n");
                                                        fprintf(user_code, "syscall\n");
                                                        fprintf(user_code, "sw $v0,
%s\n", $2);
                                                }
                                        }
                                }

print_stmt:                     PRINTT id_or_num
                                {
                                        string opr($2);
                                        if(stringType(opr)==3){
                                                string xx(getArrayParam($2));
                                                char * zz = getArrayParam($2);
                                                char * yy = getArrayName($2);
                                                if(stringType(xx)==0)
                                                        fprintf(user_code,"lw $t3, %s\n",
zz);
                                                else
                                                        fprintf(user_code,"li $t3, %s\n",
zz);
                                                fprintf(user_code, "la $t4, %s\n", yy);

                                                if(yy[0]=='f'){
```

```
                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                fprintf(user_code, "l.s $f12,
0($t4)\n");
                                                fprintf(user_code, "li $v0, 2\n");
                                        }
                                        else{
                                                fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                fprintf(user_code, "lw $a0,
0($t4)\n");
                                                fprintf(user_code, "li $v0, 1\n");
                                        }
                                }
                                else if(stringType(opr)==0){
                                        checkNewDeclare($2);
                                        if(opr[0]=='f'){
                                                fprintf(user_code,"l.s $f12, %s\n",
$2);
                                                fprintf(user_code, "li $v0, 2\n");
                                        }
                                        else{
                                                fprintf(user_code,"lw $a0, %s\n",
$2);
                                                fprintf(user_code, "li $v0, 1\n");
                                        }
                                }
                                else{
                                        if(stringType(opr)==2){
                                                fprintf(user_code,"li.s $f12,
%s\n", $2);
                                                fprintf(user_code, "li $v0, 2\n");
                                        }
                                        else{
                                                fprintf(user_code,"li $a0, %s\n",
$2);
                                                fprintf(user_code, "li $v0, 1\n");
                                        }
                                }
                                fprintf(user_code, "syscall\n");
                        }
                        | PRINTT stringgg
```

```
                                        {
                                                fprintf(final_code,"string%d:\t\t.asciiz %s\n",
stringCounter, $2);
                                                fprintf(user_code, "la $a0, string%d\n",
stringCounter);
                                                stringCounter++;
                                                fprintf(user_code, "li $v0, 4\n");
                                                fprintf(user_code, "syscall\n");
                                        }

binary_operation:       var_ID EQ id_or_num arith_rel_ops id_or_num
                                        {
                                                if(strcmp($4,"+")==0)
                                                        add_operation($1, $3, $5);
                                                else if(strcmp($4,"-")==0)
                                                        sub_operation($1, $3, $5);
                                                else if(strcmp($4,"*")==0)
                                                        mul_operation($1, $3, $5);
                                                else if(strcmp($4,"/")==0)
                                                        div_operation($1, $3, $5);
                                                else if(strcmp($4,"<")==0)
                                                        less_than_op($1, $3, $5);
                                                else if(strcmp($4,">")==0)
                                                        great_than_op($1, $3, $5);
                                                else if(strcmp($4,"==")==0)
                                                        equal_op($1, $3, $5);
                                                else if(strcmp($4,"<=")==0)
                                                        less_eq_op($1, $3, $5);
                                                else if(strcmp($4,">=")==0)
                                                        great_eq_op($1, $3, $5);
                                                else if(strcmp($4,"!=")==0)
                                                        not_eq_op($1, $3, $5);

                                        }

id_or_num :             var_ID { $$ = $1;}
                                        | num {$$ = $1;}

arith_rel_ops: ARITH_REL_OPS {$$ = strdup(yytext);}

var_ID:                         ID {$$ = strdup(yytext);}

num:                            INT {$$ = strdup(yytext);}
                                        | FLOAT {$$ = strdup(yytext);}

stringgg:                       STRINGG { $$ = strdup(yytext);}
```

```
assignment:                     var_ID EQ var_ID
                                        {
                                                bool floR=false, floOp=false, resArr=false,
oprArr=false;
                                                string res($1);
                                                string opr($3);

                                                if(res[0]=='f')
                                                        floR=true;
                                                if(opr[0]=='f')
                                                        floOp=true;

                                                checkNewDeclare($3);
                                                checkNewDeclare($1);


                                                if(stringType(opr)==3){
                                                        oprArr=true;
                                                        string xx(getArrayParam($3));
                                                        char * zz = getArrayParam($3);
                                                        char * yy = getArrayName($3);
                                                        if(stringType(xx)==0)
                                                                fprintf(user_code,"lw $t3, %s\n",
zz);
                                                        else
                                                                fprintf(user_code,"li $t3, %s\n",
zz);
                                                        fprintf(user_code, "la $t4, %s\n", yy);

                                                        if(yy[0]=='f'){
                                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "l.s $f0,
0($t4)\n");
                                                        }
                                                        else{
                                                                fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "lw $t0,
0($t4)\n");
                                                        }
                                        }
                                        if(stringType(res)==3){
                                             resArr = true;
                                             string xx(getArrayParam($1));
                                             char * zz = getArrayParam($1);
                                             char * yy = getArrayName($1);
                                             if(stringType(xx)==0)
                                                  fprintf(user_code,"lw $t3, %s\n",
zz);
                                             else
                                                  fprintf(user_code,"li $t3, %s\n",
zz);
                                             fprintf(user_code, "la $t4, %s\n", yy);

                                             if(yy[0]=='f'){
                                                  fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                  fprintf(user_code, "add $t4, $t4,
$t3\n");
                                             }
                                             else{
                                                  fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                  fprintf(user_code, "add $t4, $t4,
$t3\n");
                                             }
                                        }


                                        if(!floR && !floOp){
                                             if(!oprArr)
                                                  fprintf(user_code,"lw $t0, %s\n",
$3);
                                             if(!resArr)
                                                  fprintf(user_code,"sw $t0, %s\n",
$1);
                                             else
                                                  fprintf(user_code,"sw $t0,
0($t4)\n");
                                        }
                                        else if(floR && !floOp){
                                             if(!oprArr){
                                                  fprintf(user_code,"l.s $f0, %s\n",
$3);
                                                  fprintf(user_code,"cvt.s.w $f0,
$f0\n");
```

```
                                        }
                                        else{
                                                fprintf(user_code,"mtc1 $t0,
$f0\n");
                                                fprintf(user_code,"cvt.s.w $f0,
$f0\n");
                                        }
                                        if(!resArr)
                                                fprintf(user_code,"s.s $f0, %s\n",
$1);
                                        else
                                                fprintf(user_code,"s.s $f0,
0($t4)\n");
                                }
                                else if(floR && floOp){
                                        if(!oprArr)
                                                fprintf(user_code,"l.s $f0, %s\n",
$3);
                                        if(!resArr)
                                                fprintf(user_code,"s.s $f0, %s\n",
$1);
                                        else
                                                fprintf(user_code,"s.s $f0,
0($t4)\n");
                                }

                        }
                        | var_ID EQ num
                        {
                                bool floR=false, floOp=false, resArr=false;
                                string res($1);
                                string opr($3);

                                if(res[0]=='f')
                                        floR=true;
                                if(stringType(opr)==2)
                                        floOp=true;

                                checkNewDeclare($1);

                                if(stringType(res)==3){
                                        resArr = true;
                                        string xx(getArrayParam($1));
                                        char * zz = getArrayParam($1);
                                        char * yy = getArrayName($1);
                                        if(stringType(xx)==0)
                                                fprintf(user_code,"lw $t3, %s\n",
zz);
                                        else
                                                fprintf(user_code,"li $t3, %s\n",
zz);
                                        fprintf(user_code, "la $t4, %s\n", yy);

                                        if(yy[0]=='f'){
                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                        }
                                        else{
                                                fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                        }
                                }

                                if(!floR && !floOp){
                                        fprintf(user_code,"li $t0, %s\n", $3);
                                        if(!resArr)
                                                fprintf(user_code,"sw $t0, %s\n",
$1);
                                        else
                                                fprintf(user_code,"sw $t0,
0($t4)\n");
                                }
                                else if(floR && !floOp){
                                        fprintf(user_code,"li.s $f0, %s.0\n", $3);
                                        if(!resArr)
                                                fprintf(user_code,"s.s $f0, %s\n",
$1);
                                        else
                                                fprintf(user_code,"s.s $f0,
0($t4)\n");
                                }
                                else if(floR && floOp){
                                        fprintf(user_code,"li.s $f0, %s\n", $3);
                                        if(!resArr)
                                                fprintf(user_code,"s.s $f0, %s\n",
$1);
                                        else
```

```
                                                        fprintf(user_code,"s.s $f0,
0($t4)\n");
                                            }
                                    }

label :                 LABEL
                                    {
                                            $$ = strdup(yytext);
                                            fprintf(user_code,"%s\n", $$);
                                    }

jump_Cond :             IF var_ID GOTO var_ID
                                    {
                                            fprintf(user_code,"lw $t0 %s\n", $2);
                                            fprintf(user_code,"bne $t0, 0 %s\n",$4);
                                    }
                                    | IF num GOTO var_ID
                                    {
                                            fprintf(user_code,"li $t0 %s\n", $2);
                                            fprintf(user_code,"bne $t0, 0 %s\n",$4);
                                    }

jump_unCond :                   GOTO var_ID
                                    {
                                            fprintf(user_code,"b %s\n",$2);
                                    }

args_stmt       :               ARGS var_ID
                                    {
                                            string a($2);
                                            allVar.push_back(a);
                                            if($2[0]=='f'){
                                                    fprintf(final_code, "%s:\t\t.float 0.0\n",
$2);
                                                    fprintf(user_code, "s.s $f%d, %s\n",
globArgsFloReg, $2);
                                                    globArgsFloReg++;
                                            }
                                            else{
                                                    fprintf(final_code, "%s:\t\t.word 0\n", $2);
                                                    fprintf(user_code, "sw $s%d, %s\n",
globArgsIntReg, $2);
                                                    globArgsIntReg++;
                                            }
                                    }
```

```
arr_decl_stmt:          DECL var_ID
                                        {
                                                string arrName(getArrayName($2));
                                                //string arrSize(getArrayParam($2));
                                                int n = atoi(getArrayParam($2));
                                                string ss="";
                                                if(arrName[0]=='f'){
                                                        for(int i=1;i<n;i++)
                                                                ss += "0.0, ";
                                                        ss += "0.0";
                                                        const char *cstr = ss.c_str();
                                                        fprintf(final_code, "%s:\t\t.float %s\n",
getArrayName($2), cstr);
                                                }
                                                else{
                                                        for(int i=1;i<n;i++)
                                                                ss += "0, ";
                                                        ss += "0";
                                                        const char *cstr = ss.c_str();
                                                        fprintf(final_code, "%s:\t\t.word %s\n",
getArrayName($2), cstr);
                                                }
                                        }

param_stmt      :               PARAM id_or_num
                                        {
                                                string a($2);
                                                if(stringType(a)==3){
                                                        string xx(getArrayParam($2));
                                                        char * zz = getArrayParam($2);
                                                        char * yy = getArrayName($2);
                                                        if(stringType(xx)==0)
                                                                fprintf(user_code,"lw $t3, %s\n",
zz);
                                                        else
                                                                fprintf(user_code,"li $t3, %s\n",
zz);
                                                        fprintf(user_code, "la $t4, %s\n", yy);

                                                        if(yy[0]=='f'){
                                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "l.s $f%d,
0($t4)\n",globArgsFloReg);
```

```
								globArgsFloReg++;
							}
							else{
								fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
								fprintf(user_code, "add $t4, $t4,
$t3\n");
								fprintf(user_code, "lw $s%d,
0($t4)\n",globArgsIntReg);
								globArgsIntReg++;
							}
						}
						else if(stringType(a)==0)
						{
							checkNewDeclare($2);
							if(a[0]=='f'){
								fprintf(user_code, "l.s $f%d,
%s\n", globArgsFloReg, $2);
								globArgsFloReg++;
							}
							else{
								fprintf(user_code, "lw $s%d,
%s\n", globArgsIntReg, $2);
								globArgsIntReg++;
							}
						}
						else if(stringType(a)==1)
						{
							fprintf(user_code, "li $s%d, %s\n",
globArgsIntReg, $2);
							globArgsIntReg++;
						}
						else
						{
							fprintf(user_code, "li.s $f%d, %s\n",
globArgsFloReg, $2);
							globArgsFloReg++;
						}
					}
return_stmt	:		RETURN ret_val
					{
						fprintf(user_code,"jr $ra\n" );
					}

ret_val		:		NULLL {}
					| id_or_num
```

```
                                        {
                                                $$ = $1;
                                                checkNewDeclare($1);
                                                string a($1);

                                                if(stringType(a)==3){
                                                        string xx(getArrayParam($1));
                                                        char * zz = getArrayParam($1);
                                                        char * yy = getArrayName($1);
                                                        if(stringType(xx)==0)
                                                                fprintf(user_code,"lw $t3, %s\n",
zz);
                                                        else
                                                                fprintf(user_code,"li $t3, %s\n",
zz);
                                                        fprintf(user_code, "la $t4, %s\n", yy);

                                                        if(yy[0]=='f'){
                                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "l.s $f20,
0($t4)\n");

                                                        }
                                                        else{
                                                                fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "lw $s7,
0($t4)\n");

                                                        }
                                                }
                                                else if(stringType(a)==0){

                                                        if(a[0]=='f')
                                                                fprintf(user_code, "l.s $f20,
%s\n", $1);
                                                        else
                                                                fprintf(user_code, "lw $s7, %s\n",
$1);
                                                }
                                                else if(stringType(a)==1)
                                                        fprintf(user_code, "li $s7, %s\n", $1);
                                                else
                                                        fprintf(user_code, "li.s $f20, %s\n", $1);
                                        }

refparam_stmt:          REFPARAM var_ID
                                        {
                                                checkNewDeclare($2);
                                                string a($2);
                                                if(stringType(a)==3){
                                                        string xx(getArrayParam($2));
                                                        char * zz = getArrayParam($2);
                                                        char * yy = getArrayName($2);
                                                        if(stringType(xx)==0)
                                                                fprintf(user_code,"lw $t3, %s\n",
zz);
                                                        else
                                                                fprintf(user_code,"li $t3, %s\n",
zz);
                                                        fprintf(user_code, "la $t4, %s\n", yy);

                                                        if(yy[0]=='f'){
                                                                fprintf(user_code, "li $t5, 8\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "s.s $f20,
0($t4)\n");

                                                        }
                                                        else{
                                                                fprintf(user_code, "li $t5, 4\n");
fprintf(user_code, "mul $t3, $t3, $t5\n");
                                                                fprintf(user_code, "add $t4, $t4,
$t3\n");
                                                                fprintf(user_code, "sw $s7,
0($t4)\n");

                                                        }
                                                }
                                                else{
                                                        if($2[0]=='f')
                                                                fprintf(user_code, "s.s $f20,
%s\n", $2);
                                                        else
```

```
                                                                    fprintf(user_code, "sw $s7, 
%s\n", $2);
                                                      }
                                              }

call_stmt                   :           CALL var_ID
                                              {
                                                      fprintf(user_code, "addi $sp, $sp, -4\n");
                                                      fprintf(user_code, "sw $ra, 0($sp)\n");
                                                      fprintf(user_code, "jal %s\n", $2);
                                                      fprintf(user_code, "lw $ra, 0($sp)\n");
                                                      fprintf(user_code, "addi $sp, $sp, 4\n");
                                              }

%%

int stringType(string x){
        for(int i=0; i<x.size();i++){
                if(x[i]=='(')
                        return 3;       // Array Var
        }
        if(x[0]=='f'||x[0]=='i')
                return 0;               // Variable
        for(int i=0; i<x.size();i++){
                if(x[i]=='.')
                        return 2;       // Float
        }
        return 1;                       // Integer
}

char * getArrayName(char *a){
        string s(a);
        string s2 = s.substr(0, s.find("("));
        char *cstr = new char[s2.length() + 1];
        strcpy(cstr, s2.c_str());
        return cstr;
}

char * getArrayParam(char * a){
        string s(a);
        string s2 = s.substr(s.find("(")+1, s.find(")")-s.find("(")-1);
        char *cstr = new char[s2.length() + 1];
        strcpy(cstr, s2.c_str());
        return cstr;
}
```

```
void add_operation(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);
        bool flo1=false;
        bool flo2=false;

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                if(yy[0]=='f'){
                        fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "l.s $f1, 0($t4)\n");
                }
                else{
                        fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "lw $t1, 0($t4)\n");
                }

        }
        else if(stringType(op1)==0){
                checkNewDeclare(a);
                if(op1[0]=='f'){
                        flo1 = true;
                        fprintf(user_code,"l.s $f1, %s\n",a);
                }
                else{
                        fprintf(user_code,"lw $t1, %s\n",a);
                }
        }
        else{
                if(stringType(op1)==2){
                        flo1 = true;
                        fprintf(user_code,"li.s $f1, %s\n",a);
                }
                else{
                        fprintf(user_code,"li $t1, %s\n",a);
                }
        }

        if(stringType(op2)==3){
                string xx(getArrayParam(b));
                char * zz = getArrayParam(b);
                char * yy = getArrayName(b);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                if(yy[0]=='f'){
                        fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "l.s $f2, 0($t4)\n");
                }
                else{
                        fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "lw $t2, 0($t4)\n");
                }

        }
        else if(stringType(op2)==0){
                checkNewDeclare(b);
                if(op2[0]=='f'){
                        flo2 = true;
                        fprintf(user_code,"l.s $f2, %s\n",b);
                }
                else{
                        fprintf(user_code,"lw $t2, %s\n",b);
                }
        }
        else{
                if(stringType(op2)==2){
                        flo2 = true;
                        fprintf(user_code,"li.s $f2, %s\n",b);
                }
                else{
                        fprintf(user_code,"li $t2, %s\n",b);
                }
        }

        if(stringType(res)!=3)
```

```
                checkNewDeclare(r);

        bool resArr = false;
        if(stringType(res)==3){
                resArr = true;
                string xx(getArrayParam(r));
                char * zz = getArrayParam(r);
                char * yy = getArrayName(r);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                if(yy[0]=='f'){
                        fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                }
                else{
                        fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                }
        }
        if(flo1 || flo2){
                if(!flo1){
                        fprintf(user_code,"mtc1 $t1, $f1\n");
                        fprintf(user_code,"cvt.s.w $f1, $f1\n");
                }
                if(!flo2){
                        fprintf(user_code,"mtc1 $t2, $f2\n");
                        fprintf(user_code,"cvt.s.w $f2, $f2\n");
                }
                fprintf(user_code,"add.s $f0, $f1, $f2\n");
                if(!resArr)
                        fprintf(user_code,"s.s $f0, %s\n", r);
                else
                        fprintf(user_code,"s.s $f0, 0($t4)\n");
        }
        else{
                fprintf(user_code,"add $t0, $t1, $t2\n");
                if(!resArr)
                        fprintf(user_code,"sw $t0, %s\n", r);
                else
                        fprintf(user_code,"sw $t0, 0($t4)\n");
        }
}
```

```
void sub_operation(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);
        bool flo1=false;
        bool flo2=false;

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                if(yy[0]=='f'){
                        fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "l.s $f1, 0($t4)\n");
                }
                else{
                        fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "lw $t1, 0($t4)\n");
                }

        }
        else if(stringType(op1)==0){
                checkNewDeclare(a);
                if(op1[0]=='f'){
                        flo1 = true;
                        fprintf(user_code,"l.s $f1, %s\n",a);
                }
                else{
                        fprintf(user_code,"lw $t1, %s\n",a);
                }
        }
        else{
                if(stringType(op1)==2){
                        flo1 = true;
                        fprintf(user_code,"li.s $f1, %s\n",a);
                }
                else{
```

```
                fprintf(user_code,"li $t1, %s\n",a);
        }
}

if(stringType(op2)==3){
        string xx(getArrayParam(b));
        char * zz = getArrayParam(b);
        char * yy = getArrayName(b);
        if(stringType(xx)==0)
                fprintf(user_code,"lw $t3, %s\n", zz);
        else
                fprintf(user_code,"li $t3, %s\n", zz);
        fprintf(user_code, "la $t4, %s\n", yy);

        if(yy[0]=='f'){
                fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "l.s $f2, 0($t4)\n");
        }
        else{
                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }

}
else if(stringType(op2)==0){
        checkNewDeclare(b);
        if(op2[0]=='f'){
                flo2 = true;
                fprintf(user_code,"l.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"lw $t2, %s\n",b);
        }
}
else{
        if(stringType(op2)==2){
                flo2 = true;
                fprintf(user_code,"li.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"li $t2, %s\n",b);
        }
}
```

```
if(stringType(res)!=3)
        checkNewDeclare(r);

bool resArr = false;
if(stringType(res)==3){
        resArr = true;
        string xx(getArrayParam(r));
        char * zz = getArrayParam(r);
        char * yy = getArrayName(r);
        if(stringType(xx)==0)
                fprintf(user_code,"lw $t3, %s\n", zz);
        else
                fprintf(user_code,"li $t3, %s\n", zz);
        fprintf(user_code, "la $t4, %s\n", yy);

        if(yy[0]=='f'){
                fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
        }
        else{
                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
        }
}
if(flo1 || flo2){
        if(!flo1){
                fprintf(user_code,"mtc1 $t1, $f1\n");
                fprintf(user_code,"cvt.s.w $f1, $f1\n");
        }
        if(!flo2){
                fprintf(user_code,"mtc1 $t2, $f2\n");
                fprintf(user_code,"cvt.s.w $f2, $f2\n");
        }
        fprintf(user_code,"sub.s $f0, $f1, $f2\n");
        if(!resArr)
                fprintf(user_code,"s.s $f0, %s\n", r);
        else
                fprintf(user_code,"s.s $f0, 0($t4)\n");
}
else{
        fprintf(user_code,"sub $t0, $t1, $t2\n");
        if(!resArr)
                fprintf(user_code,"sw $t0, %s\n", r);
        else
                fprintf(user_code,"sw $t0, 0($t4)\n");
}
```

```
}

void mul_operation(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);
        bool flo1=false;
        bool flo2=false;

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                if(yy[0]=='f'){
                        fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "l.s $f1, 0($t4)\n");
                }
                else{
                        fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                        fprintf(user_code, "add $t4, $t4, $t3\n");
                        fprintf(user_code, "lw $t1, 0($t4)\n");
                }

        }
        else if(stringType(op1)==0){
                checkNewDeclare(a);
                if(op1[0]=='f'){
                        flo1 = true;
                        fprintf(user_code,"l.s $f1, %s\n",a);
                }
                else{
                        fprintf(user_code,"lw $t1, %s\n",a);
                }
        }
        else{
                if(stringType(op1)==2){
                        flo1 = true;
                        fprintf(user_code,"li.s $f1, %s\n",a);
                }
        else{
                fprintf(user_code,"li $t1, %s\n",a);
        }
}

if(stringType(op2)==3){
        string xx(getArrayParam(b));
        char * zz = getArrayParam(b);
        char * yy = getArrayName(b);
        if(stringType(xx)==0)
                fprintf(user_code,"lw $t3, %s\n", zz);
        else
                fprintf(user_code,"li $t3, %s\n", zz);
        fprintf(user_code, "la $t4, %s\n", yy);

        if(yy[0]=='f'){
                fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "l.s $f2, 0($t4)\n");
        }
        else{
                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }

}
else if(stringType(op2)==0){
        checkNewDeclare(b);
        if(op2[0]=='f'){
                flo2 = true;
                fprintf(user_code,"l.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"lw $t2, %s\n",b);
        }
}
else{
        if(stringType(op2)==2){
                flo2 = true;
                fprintf(user_code,"li.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"li $t2, %s\n",b);
        }
}
```

```
if(stringType(res)!=3)
        checkNewDeclare(r);

bool resArr = false;
if(stringType(res)==3){
        resArr = true;
        string xx(getArrayParam(r));
        char * zz = getArrayParam(r);
        char * yy = getArrayName(r);
        if(stringType(xx)==0)
                fprintf(user_code,"lw $t3, %s\n", zz);
        else
                fprintf(user_code,"li $t3, %s\n", zz);
        fprintf(user_code, "la $t4, %s\n", yy);

        if(yy[0]=='f'){
                fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
        }
        else{
                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
        }
}
if(flo1 || flo2){
        if(!flo1){
                fprintf(user_code,"mtc1 $t1, $f1\n");
                fprintf(user_code,"cvt.s.w $f1, $f1\n");
        }
        if(!flo2){
                fprintf(user_code,"mtc1 $t2, $f2\n");
                fprintf(user_code,"cvt.s.w $f2, $f2\n");
        }
        fprintf(user_code,"mul.s $f0, $f1, $f2\n");
        if(!resArr)
                fprintf(user_code,"s.s $f0, %s\n", r);
        else
                fprintf(user_code,"s.s $f0, 0($t4)\n");
}
else{
        fprintf(user_code,"mul $t0, $t1, $t2\n");
        if(!resArr)
                fprintf(user_code,"sw $t0, %s\n", r);
        else
                fprintf(user_code,"sw $t0, 0($t4)\n");
```

```
	}
}

void div_operation(char *r, char *a, char *b){
	string res(r);
	string op1(a);
	string op2(b);
	bool flo1=false;
	bool flo2=false;

	if(stringType(op1)==3){
		string xx(getArrayParam(a));
		char * zz = getArrayParam(a);
		char * yy = getArrayName(a);
		if(stringType(xx)==0)
			fprintf(user_code,"lw $t3, %s\n", zz);
		else
			fprintf(user_code,"li $t3, %s\n", zz);
		fprintf(user_code, "la $t4, %s\n", yy);

		if(yy[0]=='f'){
			fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
			fprintf(user_code, "add $t4, $t4, $t3\n");
			fprintf(user_code, "l.s $f1, 0($t4)\n");
		}
		else{
			fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
			fprintf(user_code, "add $t4, $t4, $t3\n");
			fprintf(user_code, "lw $t1, 0($t4)\n");
		}

	}
	else if(stringType(op1)==0){
		checkNewDeclare(a);
		if(op1[0]=='f'){
			flo1 = true;
			fprintf(user_code,"l.s $f1, %s\n",a);
		}
		else{
			fprintf(user_code,"lw $t1, %s\n",a);
		}
	}
	else{
		if(stringType(op1)==2){
			flo1 = true;
			fprintf(user_code,"li.s $f1, %s\n",a);
```

```
        }
        else{
                fprintf(user_code,"li $t1, %s\n",a);
        }
}

if(stringType(op2)==3){
        string xx(getArrayParam(b));
        char * zz = getArrayParam(b);
        char * yy = getArrayName(b);
        if(stringType(xx)==0)
                fprintf(user_code,"lw $t3, %s\n", zz);
        else
                fprintf(user_code,"li $t3, %s\n", zz);
        fprintf(user_code, "la $t4, %s\n", yy);

        if(yy[0]=='f'){
                fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "l.s $f2, 0($t4)\n");
        }
        else{
                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }

}
else if(stringType(op2)==0){
        checkNewDeclare(b);
        if(op2[0]=='f'){
                flo2 = true;
                fprintf(user_code,"l.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"lw $t2, %s\n",b);
        }
}
else{
        if(stringType(op2)==2){
                flo2 = true;
                fprintf(user_code,"li.s $f2, %s\n",b);
        }
        else{
                fprintf(user_code,"li $t2, %s\n",b);
        }
```

```
}

if(stringType(res)!=3)
	checkNewDeclare(r);

bool resArr = false;
if(stringType(res)==3){
	resArr = true;
	string xx(getArrayParam(r));
	char * zz = getArrayParam(r);
	char * yy = getArrayName(r);
	if(stringType(xx)==0)
		fprintf(user_code,"lw $t3, %s\n", zz);
	else
		fprintf(user_code,"li $t3, %s\n", zz);
	fprintf(user_code, "la $t4, %s\n", yy);

	if(yy[0]=='f'){
		fprintf(user_code, "li $t5, 8\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
		fprintf(user_code, "add $t4, $t4, $t3\n");
	}
	else{
		fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
		fprintf(user_code, "add $t4, $t4, $t3\n");
	}
}
if(flo1 || flo2){
	if(!flo1){
		fprintf(user_code,"mtc1 $t1, $f1\n");
		fprintf(user_code,"cvt.s.w $f1, $f1\n");
	}
	if(!flo2){
		fprintf(user_code,"mtc1 $t2, $f2\n");
		fprintf(user_code,"cvt.s.w $f2, $f2\n");
	}
	fprintf(user_code,"div.s $f0, $f1, $f2\n");
	if(!resArr)
		fprintf(user_code,"s.s $f0, %s\n", r);
	else
		fprintf(user_code,"s.s $f0, 0($t4)\n");
}
else{
	fprintf(user_code,"div $t0, $t1, $t2\n");
	if(!resArr)
		fprintf(user_code,"sw $t0, %s\n", r);
	else
```

```
                fprintf(user_code,"sw $t0, 0($t4)\n");
        }
}

void less_than_op(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);

        checkNewDeclare(a);
        checkNewDeclare(b);
        checkNewDeclare(r);

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t1, 0($t4)\n");
        }
        else if(stringType(op1)==0)
                fprintf(user_code,"lw $t1, %s\n",a);
        else
                fprintf(user_code,"li $t1, %s\n",a);


        if(stringType(op2)==3){
                string xx(getArrayParam(b));
                char * zz = getArrayParam(b);
                char * yy = getArrayName(b);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }
        else if(stringType(op2)==0)
                fprintf(user_code,"lw $t2, %s\n",b);
        else
                fprintf(user_code,"li $t2, %s\n",b);

        fprintf(user_code,"li $t0, 0\n");
        fprintf(user_code,"slt $t0, $t1, $t2\n");
        fprintf(user_code,"sw $t0, %s\n", r);
}

void great_than_op(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);

        checkNewDeclare(a);
        checkNewDeclare(b);
        checkNewDeclare(r);

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t1, 0($t4)\n");
        }
        else if(stringType(op1)==0)
                fprintf(user_code,"lw $t1, %s\n",a);
        else
                fprintf(user_code,"li $t1, %s\n",a);


        if(stringType(op2)==3){
                string xx(getArrayParam(b));
                char * zz = getArrayParam(b);
                char * yy = getArrayName(b);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }
        else if(stringType(op2)==0)
                fprintf(user_code,"lw $t2, %s\n",b);
        else
                fprintf(user_code,"li $t2, %s\n",b);

        fprintf(user_code,"li $t0, 0\n");
        fprintf(user_code,"sgt $t0, $t1, $t2\n");
        fprintf(user_code,"sw $t0, %s\n", r);
}

void equal_op(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);

        checkNewDeclare(a);
        checkNewDeclare(b);
        checkNewDeclare(r);

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t1, 0($t4)\n");
        }
        else if(stringType(op1)==0)
                fprintf(user_code,"lw $t1, %s\n",a);
        else
                fprintf(user_code,"li $t1, %s\n",a);
```

```
		if(stringType(op2)==3){
			string xx(getArrayParam(b));
			char * zz = getArrayParam(b);
			char * yy = getArrayName(b);
			if(stringType(xx)==0)
				fprintf(user_code,"lw $t3, %s\n", zz);
			else
				fprintf(user_code,"li $t3, %s\n", zz);
			fprintf(user_code, "la $t4, %s\n", yy);

			fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
			fprintf(user_code, "add $t4, $t4, $t3\n");
			fprintf(user_code, "lw $t2, 0($t4)\n");
		}
		else if(stringType(op2)==0)
			fprintf(user_code,"lw $t2, %s\n",b);
		else
			fprintf(user_code,"li $t2, %s\n",b);

		fprintf(user_code,"li $t0, 0\n");
		fprintf(user_code,"seq $t0, $t1, $t2\n");
		fprintf(user_code,"sw $t0, %s\n", r);
	}

	void less_eq_op(char *r, char *a, char *b){
		string res(r);
		string op1(a);
		string op2(b);

		checkNewDeclare(a);
		checkNewDeclare(b);
		checkNewDeclare(r);

		if(stringType(op1)==3){
			string xx(getArrayParam(a));
			char * zz = getArrayParam(a);
			char * yy = getArrayName(a);
			if(stringType(xx)==0)
				fprintf(user_code,"lw $t3, %s\n", zz);
			else
				fprintf(user_code,"li $t3, %s\n", zz);
			fprintf(user_code, "la $t4, %s\n", yy);

			fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
			fprintf(user_code, "add $t4, $t4, $t3\n");
			fprintf(user_code, "lw $t1, 0($t4)\n");
		}
		else if(stringType(op1)==0)
			fprintf(user_code,"lw $t1, %s\n",a);
		else
			fprintf(user_code,"li $t1, %s\n",a);


		if(stringType(op2)==3){
			string xx(getArrayParam(b));
			char * zz = getArrayParam(b);
			char * yy = getArrayName(b);
			if(stringType(xx)==0)
				fprintf(user_code,"lw $t3, %s\n", zz);
			else
				fprintf(user_code,"li $t3, %s\n", zz);
			fprintf(user_code, "la $t4, %s\n", yy);

			fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
			fprintf(user_code, "add $t4, $t4, $t3\n");
			fprintf(user_code, "lw $t2, 0($t4)\n");
		}
		else if(stringType(op2)==0)
			fprintf(user_code,"lw $t2, %s\n",b);
		else
			fprintf(user_code,"li $t2, %s\n",b);

		fprintf(user_code,"li $t0, 0\n");
		fprintf(user_code,"sle $t0, $t1, $t2\n");
		fprintf(user_code,"sw $t0, %s\n", r);
}

void great_eq_op(char *r, char *a, char *b){
		string res(r);
		string op1(a);
		string op2(b);

		checkNewDeclare(a);
		checkNewDeclare(b);
		checkNewDeclare(r);

		if(stringType(op1)==3){
			string xx(getArrayParam(a));
			char * zz = getArrayParam(a);
			char * yy = getArrayName(a);
			if(stringType(xx)==0)
```

```
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t1, 0($t4)\n");
        }
        else if(stringType(op1)==0)
                fprintf(user_code,"lw $t1, %s\n",a);
        else
                fprintf(user_code,"li $t1, %s\n",a);


        if(stringType(op2)==3){
                string xx(getArrayParam(b));
                char * zz = getArrayParam(b);
                char * yy = getArrayName(b);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }
        else if(stringType(op2)==0)
                fprintf(user_code,"lw $t2, %s\n",b);
        else
                fprintf(user_code,"li $t2, %s\n",b);

        fprintf(user_code,"li $t0, 0\n");
        fprintf(user_code,"sge $t0, $t1, $t2\n");
        fprintf(user_code,"sw $t0, %s\n", r);
}

void not_eq_op(char *r, char *a, char *b){
        string res(r);
        string op1(a);
        string op2(b);

        checkNewDeclare(a);
        checkNewDeclare(b);
        checkNewDeclare(r);

        if(stringType(op1)==3){
                string xx(getArrayParam(a));
                char * zz = getArrayParam(a);
                char * yy = getArrayName(a);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t1, 0($t4)\n");
        }
        else if(stringType(op1)==0)
                fprintf(user_code,"lw $t1, %s\n",a);
        else
                fprintf(user_code,"li $t1, %s\n",a);


        if(stringType(op2)==3){
                string xx(getArrayParam(b));
                char * zz = getArrayParam(b);
                char * yy = getArrayName(b);
                if(stringType(xx)==0)
                        fprintf(user_code,"lw $t3, %s\n", zz);
                else
                        fprintf(user_code,"li $t3, %s\n", zz);
                fprintf(user_code, "la $t4, %s\n", yy);

                fprintf(user_code, "li $t5, 4\n"); fprintf(user_code, "mul $t3, $t3, $t5\n");
                fprintf(user_code, "add $t4, $t4, $t3\n");
                fprintf(user_code, "lw $t2, 0($t4)\n");
        }
        else if(stringType(op2)==0)
                fprintf(user_code,"lw $t2, %s\n",b);
        else
                fprintf(user_code,"li $t2, %s\n",b);

        fprintf(user_code,"li $t0, 0\n");
        fprintf(user_code,"sne $t0, $t1, $t2\n");
        fprintf(user_code,"sw $t0, %s\n", r);
}
```

```
void checkNewDeclare(char * s){
	string ss(s);
	if(stringType(ss)!=0)
		return;

	if(find(allVar.begin(), allVar.end(), ss) == allVar.end()){
		allVar.push_back(ss);
		if(s[0]=='f')
			fprintf(final_code,"%s:\t\t .float 0.0\n", s);
		else
			fprintf(final_code,"%s:\t\t .word 0\n", s);

	}
}

int main (void) {
	char a[1000];

	user_code=fopen("temp_mips.s","w");
	final_code=fopen("mips.s","w");

	fprintf(final_code,".data\n");
	fprintf(final_code,"newLine:\t\t.asciiz \"\\n\"\n");

	yyparse ();

	fprintf(final_code,"\n.text\n" );

	fclose(user_code);
	fclose(final_code);

	std::ifstream in("temp_mips.s");
	std::ofstream out("mips.s", std::ios::app);
	out << in.rdbuf();

	return 0;
}

int yyerror (char *s){
	fprintf (stderr, "%s\n", s);
}
```

Makefile

```
all:
    bison -d -v parser1.y
    flex lexer1.lex
    g++ -g -std=c++11 lex.yy.c parser1.tab.c parser1.tab.h -o main1
-lfl
    ./main1 < input.c
    bison -d -v -t parser2.y
    flex lexer2.lex
    g++ lex.yy.c parser2.tab.c parser2.tab.h -o main2 -lfl
    ./main2 < intermediate.txt
```