Lex Part:

```
"#include" { return HINCLUDE;}
stdio.h|stdlib.h return LIBNAME;


; return SEMI;
\{ return CBO;
\} return CBC;
\( return SBO;
\) return SBC;
\&  return AMP;

"int"       {installID(TYPE_KEYWORD, yytext); return INT;}
"float"           {installID(TYPE_KEYWORD, yytext); return FLOAT;}
"char"            {installID(TYPE_KEYWORD, yytext);return CHAR;}
"while"           {installID(TYPE_KEYWORD, yytext);return WHILE;}
"main"            {installID(TYPE_KEYWORD, yytext);return MAIN;}
, return COMMA;

[+-]?[0-9]+\.?[0-9]*  {installID(TYPE_DIGIT, yytext);return NUM; }
"++"              {installID(TYPE_OPERATOR, yytext); return INCOP; }
"+"   {installID(TYPE_OPERATOR, yytext);return PLUS;}
"--"  {installID(TYPE_OPERATOR, yytext);return DECOP;}
"-"   {installID(TYPE_OPERATOR, yytext);return MINUS;}
{id}  {installID(TYPE_IDENTIFIER, yytext); return ID;}
"<"   {installID(TYPE_OPERATOR, yytext); return LE;}
"<="  {installID(TYPE_OPERATOR, yytext); return LEQ; }
">"   {installID(TYPE_OPERATOR, yytext);return GE; }
">="  {installID(TYPE_OPERATOR, yytext);return GEQ; }
"=="  {installID(TYPE_OPERATOR, yytext);return DEQ; }
"="   {installID(TYPE_OPERATOR, yytext);return EQ; }

\".*\" {return STRING; }
\"  {inDouble = 1 - inDouble;}
"//".* { if (inDouble && !inComment) printf("Unexpected1: %s\n", yytext); else
{inDoublecom++;}}
"/*" { BEGIN(C_COMMENT); if(!inDoublecom) inComment=1;}
<C_COMMENT>"*/"     { BEGIN(INITIAL);if (inComment) {inComment=0;}
            else
                  printf("Unexpected2: %s\n", yytext);
        }
<C_COMMENT>\n {if(inDoublecom) inDoublecom=0; line_number++;}
<C_COMMENT>[ \t] {}
<C_COMMENT>. {if (!inComment) {printf("Unexpected3: %s\n", yytext); exit(-1);}}

\n {if(inDoublecom) inDoublecom=0; line_number++;}
[ \t] {}
. {if (!inComment) {printf("Unexpected3: %s\n", yytext); exit(-1);}}
```

Grammar part:-

```
start: header  main ;

header: HINCLUDE LE LIBNAME GE ;

main: INT MAIN SBO SBC CBO body CBC;

body: stmt  body
     | ;

stmt: decl SEMI | assgn SEMI |ctrlstmt | pstmt SEMI |sstmt SEMI ;
```

```
pstmt: PRINTF SBO STRING COMMA ID SBC {check_type($3, $5);};

sstmt: SCANF SBO STRING COMMA  AMP ID SBC {check_type($3,$6);};

decl : type names {set_type($1);};

type : INT {$$=0;} | FLOAT {$$=1;} | DOUBLE {$$=2;} | CHAR {$$=3;};

names : name COMMA names | name  ;

name : ID  {add_id($1);} | ID EQ NUM {add_id($1);};

assgn : ID EQ NUM | ID EQ ID | ID INCOP |ID DECOP;

ctrlstmt : WHILE SBO relstmt SBC CBO body CBC |
      WHILE SBO relstmt SBC stmt ;

relstmt: ID relop ID| ID relop NUM  ;

relop : LE |LEQ | GE |GEQ |NEQ |EQ |DEQ ;
```

Initial grammar for first and follow set is:

```
['start', ' header  main ']
['header', ' HINCLUDE LE LIBNAME GE ']
['main', ' INT MAIN SBO SBC CBO body CBC | INT MAIN SBO SBC CBO CBC']
['body', ' stmt  body | stmt']
['stmt', ' decl SEMI | assgn SEMI |ctrlstmt']
['decl ', ' type names ']
['type ', ' INT | FLOAT |DOUBLE |CHAR ']
['names ', ' name COMMA names | name  ']
['name ', ' ID | ID EQ NUM ']
['assgn ', ' ID EQ NUM | ID EQ ID | ID INCOP |ID DECOP']
['ctrlstmt ', ' WHILE SBO relstmt SBC CBO body CBC | WHILE SBO relstmt SBC stmt
']
['relstmt', ' ID relop ID| ID relop NUM  ']
['relop ', ' LE |LEQ | GE |GEQ |NEQ |EQ |DEQ']
```

First for th given grammar is:

```
First (start) :  set(['HINCLUDE'])
First (header) :  set(['HINCLUDE'])
First (main) :  set(['INT'])
First (body) :  set(['INT', 'DOUBLE', 'FLOAT', 'CHAR', 'WHILE', 'ID'])
First (stmt) :  set(['CHAR', 'WHILE', 'INT', 'DOUBLE', 'FLOAT', 'ID'])
First (decl) :  set(['INT', 'DOUBLE', 'FLOAT', 'CHAR'])
First (type) :  set(['INT', 'DOUBLE', 'FLOAT', 'CHAR'])
First (names) :  set(['ID'])
First (name) :  set(['ID'])
First (assgn) :  set(['ID'])
First (ctrlstmt) :  set(['WHILE'])
First (relstmt) :  set(['ID'])
First (relop) :  set(['GEQ', 'LE', 'DEQ', 'LEQ', 'GE', 'EQ', 'NEQ'])
```

Follow set for the given grammar is:

```
Follow (start) :  set(['$'])
Follow (header) :  set(['INT'])
Follow (main) :  set(['$'])
Follow (body) :  set(['CBC'])
Follow (stmt) :  set(['CBC', 'INT', 'DOUBLE', 'FLOAT', 'CHAR', 'WHILE', 'ID'])
Follow (decl) :  set(['SEMI'])
Follow (type) :  set(['ID'])
Follow (names) :  set(['SEMI'])
Follow (name) :  set(['COMMA', 'SEMI'])
Follow (assgn) :  set(['SEMI'])
Follow (ctrlstmt) :  set(['CBC', 'INT', 'DOUBLE', 'FLOAT', 'CHAR', 'WHILE', 'ID'])
Follow (relstmt) :  set(['SBC'])
Follow (relop) :  set(['NUM', 'ID'])
```

Type Checking:-

```
void check_type(int s_id, int i_id)
{
      //printf("%d %d\n", s_id, i_id);
      if ((table[i_id].type == 0) &&
            (strcmp("\"%d\"", table[s_id].value) != 0)) {
            printf("expecting %%d but got %s at line %d\n", table[s_id].value, table[s_id].line_number);
            exit(-1);
      }
      if ((table[i_id].type == 1) &&
            (strcmp("\"%f\"", table[s_id].value) != 0)) {
            printf("expecting %%f but got %s at line %d\n", table[s_id].value, table[s_id].line_number);
            exit(-1);
      }
      if ((table[i_id].type == 2) &&
            (strcmp("\"%e\"", table[s_id].value) != 0)) {
            printf("expecting %%e but got %s at line %d\n", table[s_id].value, table[s_id].line_number);
            exit(-1);
      }
      if ((table[i_id].type == 3) &&
            (strcmp("\"%s\"", table[s_id].value) != 0)) {
            printf("expecting %%s but got %s at line %d\n", table[s_id].value, table[s_id].line_number);
            exit(-1);
      }
}

void set_type(int type)
{
      //printf("setting type:%d\n", type);
      int i;

      for (i=0; i < iDIndex; i++) {
            table[iDs[i]].type = type;
      }
      iDIndex=0;
}
```