```
<expr> = number
         | identifier
         | ( <expr> operator <expr> )

<stmt> = print <expr>
         | when <expr> <stmt>
         | var identifier = <expr>
         | identifier = <expr>
         | { <stmt> ( ; <stmt> )* }

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <vector>

using namespace std;

const char ctrl_D = (char)('D' - 64);

class iosystem
{
protected:
    istream & fin;
    string line;
    int pos, linenum, linelen;

public:
  iosystem(istream & f): fin(f)
  { line = "";
    linelen = 0;
    pos = 0;
    linenum = 0; }

  char nextch()
  { if (pos > linelen)
    { getline(fin, line);
      if (fin.fail())
      { line = "";
        line += ctrl_D; }
      linelen = line.length();
      linenum += 1;
      pos = 0; }
    if (pos == linelen)
    { pos += 1;
      return '\n'; }
    pos += 1;
    return line[pos-1]; }

  void backch()
  { if (pos > 0)
      pos -= 1; }

  void error(string msg)
  { cerr << "Error, line " << linenum << ", char " << pos
         << ": " << msg << "\n";
    cerr << line << "\n";
    for (int i = 1; i < pos; i += 1)
      cerr << " ";
    cerr << "^^\n";
    exit(1); } };
```

```
enum lextype
{ LX_ERROR, LX_EOF, LX_number, LX_OP_plus, LX_OP_minus, LX_OP_times,
  LX_OP_divide, LX_OP_eq, LX_OP_noteq, LX_OP_less, LX_OP_lesseq, LX_OP_more,
  LX_OP_moreeq, LX_OP_assign, LX_RW_var, LX_RW_when, LX_RW_print,
  LX_variable, LX_semicolon, LX_opencurly, LX_closecurly, LX_openround,
  LX_closeround };

bool isoperator(lextype t)
{ switch (t)
  { case LX_OP_plus:    case LX_OP_minus:
    case LX_OP_times:   case LX_OP_divide:
    case LX_OP_eq:      case LX_OP_noteq:
    case LX_OP_less:    case LX_OP_lesseq:
    case LX_OP_more:    case LX_OP_moreeq:
      return true;
    default:
      return false; } }

struct varinfo
{ int value;
  varinfo * prev;

  varinfo(int v = 0, varinfo * p = NULL)
  { value = v;
    prev = p; }
};

struct symbol
{ string form;
  lextype kind;
  varinfo * declaration;
  symbol * next;

  symbol(string f, lextype k, varinfo * v = NULL)
  { form = f;
    kind = k;
    declaration = v;
    next = NULL; }
};


class symboltable
{
  protected:
    static const int size = 1000;
    symbol * table[size];

public:

  symboltable()
  { for (int i = 0; i < size; i += 1)
      table[i] = NULL; }

  unsigned int hash(string s)
  { unsigned int h = 9283741;
    for (int i = 0; i < s.length(); i += 1)
      h = h * 691 + s[i];
    return h; }
  symbol * lookup(string name)
  { int h = hash(name) % size;
    symbol * sym = table[h];
    while (sym != NULL)
    { if (sym->form == name)
        return sym;
      sym = sym->next; }
    return NULL; }

  symbol * enter(string name, lextype kind, varinfo * vi = NULL)
  { int h = hash(name) % size;
    symbol * sym = new symbol(name, kind, vi);
    sym->next = table[h];
    table[h] = sym;
    return sym; }
};




class lexan
{
  protected:
    iosystem & IO;
    symboltable & ST;

  public:
    lextype kind;
    string form;
    int intvalue;
    symbol * syminfo;
    bool reuse;

  void initlex()
  { kind = LX_ERROR;
    form = "";
    intvalue = 0;
    syminfo = NULL; }

  lexan(iosystem & i, symboltable & s): IO(i), ST(s)
  { reuse = false;
    initlex(); }

  void error(string s)
  { IO.error(s); }

  void nextlex()
  { if (reuse)
    { reuse = false;
      return; }
    initlex();
    char c = IO.nextch();
    while (c == ' ' || c == '\n' || c == '\t')
      c = IO.nextch();
    switch (c)
    { case ctrl_D:
      { kind = LX_EOF;
        form = "ctrl-D";
        return; }
```

```
case '0': case '1': case '2': case '3': case '4':
case '5': case '6': case '7': case '8': case '9':
{ kind = LX_number;
  while (c >= '0' && c <= '9')
  { intvalue = intvalue * 10 + c - '0';
    form = form + c;
    c = IO.nextch(); }
  IO.backch();
  return; }

case ';':
{ kind = LX_semicolon;
  form = ";";
  return; }

case '{':
{ kind = LX_opencurly;
  form = "{";
  return; }

case '}':
{ kind = LX_closecurly;
  form = "}";
  return; }

case '(':
{ kind = LX_openround;
  form = "(";
  return; }

case ')':
{ kind = LX_closeround;
  form = ")";
  return; }

case '+':
{ kind = LX_OP_plus;
  form = "+";
  return; }

case '-':
{ kind = LX_OP_minus;
  form = "-";
  return; }

case '*':
{ kind = LX_OP_times;
  form = "*";
  return; }

case '/':
{ kind = LX_OP_divide;
  form = "/";
  return; }

case '!':
{ c = IO.nextch();
  if (c == '=')
  { kind = LX_OP_noteq;
    form = "!=";
    return; }
  else
    IO.error("Illegal operator !"); }
```

```
case '=':
{ c = IO.nextch();
  if (c == '=')
  { kind = LX_OP_eq;
    form = "==";
    return; }
  else
  { IO.backch();
    kind = LX_OP_assign;
    form = "=";
    return; } }

case '<':
{ c = IO.nextch();
  if (c == '=')
  { kind = LX_OP_lesseq;
    form = "<=";
    return; }
  else
  { IO.backch();
    kind = LX_OP_less;
    form = "<";
    return; } }

case '>':
{ c = IO.nextch();
  if (c == '=')
  { kind = LX_OP_moreeq;
    form = ">=";
    return; }
  else
  { IO.backch();
    kind = LX_OP_more;
    form = ">";
    return; } }

case 'a': case 'b': case 'c': case 'd': case 'e': case 'f':
case 'g': case 'h': case 'i': case 'j': case 'k': case 'l':
case 'm': case 'n': case 'o': case 'p': case 'q': case 'r':
case 's': case 't': case 'u': case 'v': case 'w': case 'x':
case 'y': case 'z': case 'A': case 'B': case 'C': case 'D':
case 'E': case 'F': case 'G': case 'H': case 'I': case 'J':
case 'K': case 'L': case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T': case 'U': case 'V':
case 'W': case 'X': case 'Y': case 'Z':
{ while (c >= 'a' && c <= 'z' ||
         c <= 'A' && c >= 'Z' ||
         c >= '0' && c <= '9')
  { c = tolower(c);
    form = form + c;
    c = IO.nextch(); }
  IO.backch();
  symbol * sym = ST.lookup(form);
  if (sym == NULL)
  { kind = LX_variable;
    syminfo = ST.enter(form, kind); }
  else
  { kind = sym->kind;
    syminfo = sym; }
  return; }
```

```
      default:
        IO.error("Illegal character " + c); } }


  void printlex()
  { cout << "lexeme kind = " << kind << ", form = \"" << form
         << "\", intvalue = " << intvalue
         << ", syminfo = " << syminfo << "\n"; }

  void backlex()
  { reuse = true; }
};


int main()
{ iosystem IO(cin);
  symboltable ST;
  lexan LEX(IO, ST);
  ST.enter("var", LX_RW_var);
  ST.enter("when", LX_RW_when);
  ST.enter("print", LX_RW_print);
  while (true)
  { LEX.nextlex();
    LEX.printlex();
    if (LEX.kind == LX_EOF)
      break; } }
```

```
$ lang0
one two cat 34dog one when two three;6+7
lexeme kind = 17, form = "one", intvalue = 0, syminfo = 0x804d080
lexeme kind = 17, form = "two", intvalue = 0, syminfo = 0x804d090
lexeme kind = 17, form = "cat", intvalue = 0, syminfo = 0x804d0b0
lexeme kind = 2, form = "34", intvalue = 34, syminfo = 0
lexeme kind = 17, form = "dog", intvalue = 0, syminfo = 0x804d0d0
lexeme kind = 17, form = "one", intvalue = 0, syminfo = 0x804d080
lexeme kind = 15, form = "when", intvalue = 0, syminfo = 0x804d050
lexeme kind = 17, form = "two", intvalue = 0, syminfo = 0x804d090
lexeme kind = 17, form = "three", intvalue = 0, syminfo = 0x804d0f0
lexeme kind = 18, form = ";", intvalue = 0, syminfo = 0
lexeme kind = 2, form = "6", intvalue = 6, syminfo = 0
lexeme kind = 3, form = "+", intvalue = 0, syminfo = 0
lexeme kind = 2, form = "7", intvalue = 7, syminfo = 0
^D
lexeme kind = 1, form = "ctrl-D", intvalue = 0, syminfo = 0
$
```