```
enum nodetype
{ N_vardecl, N_whenstmt, N_printstmt, N_integer, N_variable, N_binaryexp,
  N_sequence };

string tostring(nodetype n)
{ switch (n)
  { case N_vardecl: return "vardecl";
    case N_whenstmt: return "whenstmt";
    case N_printstmt: return "printstmt";
    case N_integer: return "integer";
    case N_variable: return "variable";
    case N_binaryexp: return "binaryexp";
    case N_sequence: return "sequence";
    default: return "ERROR???"; } }

struct node
{ nodetype kind;
  int intvalue;
  symbol * syminfo;
  vector<node *> subtree;

  node(nodetype k)
  { kind = k;
    intvalue = 0;
    syminfo = NULL; }

  node(nodetype k, int v)
  { kind = k;
    intvalue = v;
    syminfo = NULL; }

  node(nodetype k, symbol * s)
  { kind = k;
    intvalue = 0;
    syminfo = s; }
};


void print(node * n, int indent = 0)
{ cout << setw(indent * 3) << "";
  if (n == NULL)
  { cout << "NULL\n";
    return; }

  cout << tostring(n->kind) << " ";
  if (n->intvalue != 0)
    cout << "intvalue=" << n->intvalue << " ";
  if (n->syminfo != NULL)
    cout << "syminfo=" << n->syminfo << " ";
  cout << "\n";

  for (int i = 0; i < n->subtree.size(); i += 1)
    print(n->subtree[i], indent+1); }
```

```
node * parse_expression(lexan & LEX)
{ LEX.nextlex();
  if (LEX.kind == LX_number)
    return new node(N_integer, LEX.intvalue);
  else if (LEX.kind == LX_variable)
    return new node(N_variable, LEX.syminfo);
  else
    LEX.error("expecting expression, found " + LEX.form); }


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);
  node * r = parse_expression(LEX);
  print(r); }


$ lang1
123
integer intvalue=123
$ lang1
cat
variable syminfo=0x804e090
```