Step 1

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