Lexikalischer Analysator *next()*

```
public static Token next() {
  Token t = new Token();
  while (ch <=' ') nextCh();        // skip white space
  t.line = line; t.col = col;       // set position (for error-msgs)
  switch (ch) {
    // ----- ident or keyword
    case 'a': ... case 'z': case 'A': ... case 'Z':
      readName(t);                  // identifier or keyword!!
      break;
    // ----- number
    case '0': ... case '9':
      readNumber(t);
      break;
    // ----- character constant
    case '\'':
      t.kind = Token.charConst;
      nextCh();
      if (ch == '\'') {
        error ("empty character constant");
        nextCh();
      } else {
        if (ch >= ' ' && ch <= '~')// printable chars
          t.val = (int) ch;
        else error("invalid character constant '" + ch + "'");
        nextCh();
        if (ch == '\'') nextCh();
        else error ("missing ' at end of character constant");
      }
      break;
    // ----- simple tokens
    case ';': t.kind = Token.semicolon; nextCh(); break;
    ...
    case eofCh: t.kind = Token.endFile; break;
    // ----- compound tokens
    case '=':
      nextCh();
      if (ch == '=') { t.kind = Token.eql; nextCh(); }
      else t.kind = Token.assign;
      break;
    case '/':
      nextCh();
      if (ch == '/') {              // skip eol comment
        do nextCh(); while (ch != eol && ch != eofCh);
        t = next();                 // recursion!!
      } else t.kind = Token.slash;
      break;
    case '&':
      nextCh();
      if (ch == '&') { t.kind = Token.and; nextCh(); }
      else t.kind = Token.none;
      break;
    ...
  }
  if (t.kind = Token.none) error("invalid symbol");
  return t;
}
```

StudentList.mic

```
class StudentList
  final int MAXLEN = 100;

  class Student {
    int matrNr;
    char name;
  }

  Student list[];
  int stCnt;

{

  void init () {
    list = new Student[MAXLEN];
    stCnt = 0;
  }

  void add (Student s)
    int i;
  {
    // insert sorted by matrNr
    i = stCnt-1;
    while (i >= 0 && s.matrNr < list[i].matrNr) {
      list[i+1] = list[i];
      i--;
    }
    list[i+1] = s;
    stCnt++;
  }

  int find (int matrNr)
    int l, r, x;
  {
    // do binary search
    l = 0; r = stCnt-1;
    while (l <= r && matrNr != list[x].matrNr) {
      x = (l+r)/2;
      if (matrNr < list[x].matrNr) r = x-1;
      else l = x+1;
    }
    if (matrNr == list[x].matrNr) return x;

    return -1;
  }

  void printStudent (int i) {
    print('m'); print('['); print(i); print(']'); print('=');
    print(list[i].matrNr); print(','); print(list[i].name);
    print(eol);
  }

  void main() { … }
}
```