Construction of a Recursive Descent Parser

Based on?

- The construction is made directly from the grammar
- The grammar must be LL(1), which means:
- No left recursion
- Different rules for a non-terminal must have different starters

Non-terminals

N ::= …

```
private void parseN()
{
   …
}

// public if N is the start symbol

// responsible for removing ”as many terminals as possible”
```

Rules

N ::= rule 1 | rule 2 | …

```
private void parseN()
{
   switch( currentToken ) {
      case X1: case Y1: … // starters of rule 1
         // rule 1 code
      break;
      case X2: case Y2: … // starters of rule 2
         // rule 2 code
      break;
…
   }
}
```

Sequences

N1 N2 T1 N3 T2 T3 …

```
parseN1();
parseN2();
accept(T1);
parseN3();
accept(T2);
accept(T3);
…
```

Repetitions

(sequence)*

```
while( currentToken is in starters for sequence ) {
   code for sequence
}
```

Selections

(s1 | s2 | …)

```
if( currentToken is a starter of s1 )
   code for s1
else if( currentToken is a starter of s2 )
   code for s2
…

// a switch statement could also be used
```