

KOI - Lab 5

Anton Söderlund, DT501G

2019-01-06

Part A-B

Jag tänkte det vore intressant att sätta ihop de tidigare labbarna med `lexer.l` och `parser.ypp`, istället för att använda `lexer.cpp`. Det krävde en del små omskrivningar som t.ex. jag har definierat `lexan()` i både `lexer.l` och `parser.ypp` i tidigare labbarna, så de konflikterna är lösta. Det lades även till tilldelningar i `lexer.l`, och sådan att `yyval` också definieras eftersom den används i bison's parser. Programmet använder fortfarande `token_value` och `yyval`, eftersom `.cpp`-filerna använder `token_value`, och det blev enklare än att ändra i hela programmet.

Jag lade också till en funktion `lookup_helper` i `lexer.l` så att variabler tittas efter i symboltabellen ordentligt.

Syntaxträdet utgår ifrån ledningen som gavs i labben och jag valde att bygga trädet i c++ istället då exemplet utgick ifrån det, så jag använder lab3-D för parsern och en modifierad version av `lexer.l` från lab 4. Printfunktionen är också byggd på ledningen, men jag böt ut satserna till en case/switch istället.

`parsetree.cpp` innehåller definitionen för parseträdet och funktionerna för att bygga noder/löv. För att förhindra shift/reduce-konflikter i samband med att kunna skriva flera uttryck på samma rad delade jag upp grammatiken sådan att ett träd består av *lists*, och *lists* kan vara antingen ett uttryck följt av flera uttryck, eller ett uttryck endast. Då finns det alltså ett syntaxträd för ett helt program, och programmet exekveras genom kommandot `exec` (utan semikolon). Trädet blir alltså en top-down parser, så för uttrycket "`2+5; x+7+y;`" beräknas `2+5` först, och `x+7+y` efteråt (om programmet gjorde beräkningar).

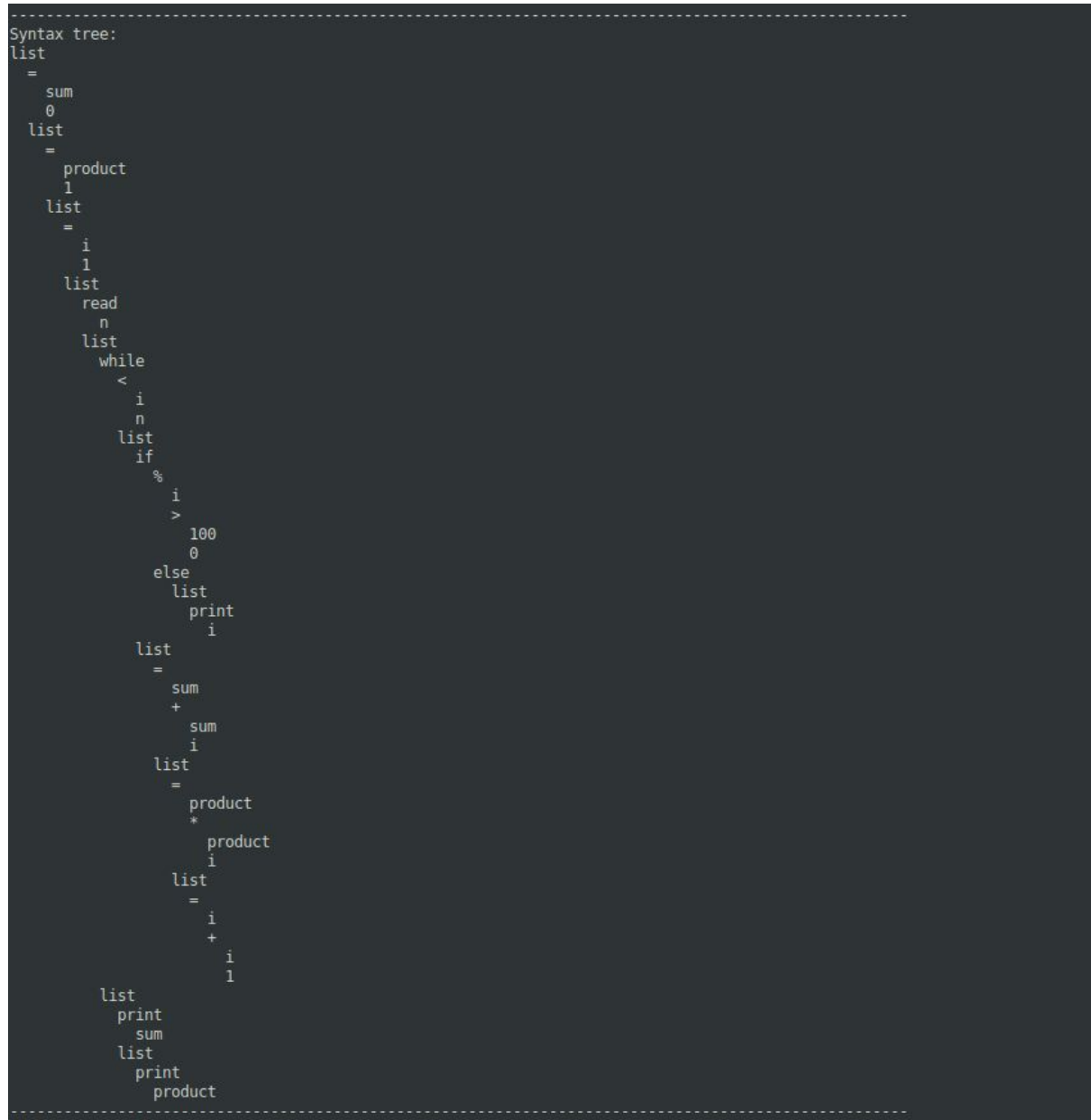
```
anton@anton:~/Documents/compiler-interpretors/anton_soderlund_dt501a/lab5/lab5_A-B$ ./infix2postfix
2+5;x+7+y;
Syntax tree:
LIST
+
  2
  5
  LIST
  +
    +
    x
    7
    y
```

Figur 1, exempel på syntaxträd

Part C

De olika slags uttrycken läggs först till i `lexer.l` sådan att vi kan returnera typerna till parsern. Sedan läggs typerna till som tokens i parsern, och nu behöver vi definiera vad varje token ska göra i grammatiken. If/else är också givna *%nonassoc* prioriteten, dvs vi kan inte säga *if(x < y < z)*. För att göra det enklare att hantera kodblock kan vi definiera ett nytt uttryck där ett kodblock kan antingen vara en rad, eller flera rader inom "{ }". Eftersom variabelnamn är

definierade som en sträng där första tecknet ska vara en bokstav från a-zA-Z så kommer t.ex. "print" hamna under den regeln, så därför definierade jag de reglerna först i *lexer.l*. Efter implementering av de nya påståenden kan vi nu skriva program sådana att



Figur 2, syntaxträd från ett testprogram

Testprogrammet använder alla nya påståenden.