Koi - Lab 2

Anton Söderlund, DT501G

Part A

I filen grammatik finns den modifierade grammatiken. Jag har lagt till sådan att,

```
list ->     assignment ; list
          | expr ; list
          | /* empty */
```

vilket betyder att inputen kan antingen vara ett uttryck eller en tilldelning.

Part B

Först ändrar jag i *parser.c* sådan att funktionen list() kallar assignment() istället för expr(), och lägger till funktionen assignment() som tittar att inputen är på formen “id = expr”.
Sen läggs ett case för “=” till i *emitter.c* som printar ut ID = expr

Part C

En stack och dess funktioner finns deklarerad i *symbol.c* som består av symentries, och symentries har en ny datatyp *symentry.value* för att spara numeriska värden för variabler. *Pop(int from_symtable)* tar ett argument för om man vill ta ett värde från symboltabellen, alltså för variabler. Det behövs för att kunna referera till variabeln själv och variabelns värde senare i emittern.

I *emitter.c* pushas inputen till stacken i varje case, och om tokenet är en operator så kallas funktionen *calc*. Den poppar de två operander från stacken, argumentet är till för att titta om en operand är en variabel så tar man dess numeriska värde från symboltabellen (problemet jag hade på labben). Sen beroende på vilken operator som skickas med dvs “token_type” så utför den motsvarande operation. I emit() poppas sedan de två sista elementen i stacken vilket är ett numeriskt värde och den variabel som blir tilldelad.

Part D

I *emitter.c* finns en ny case-sats för tecknet ‘^’ i emit() och calc() och en power(op1,op2) funktion som gör beräkningen för exponenter. I *parser.c* behöver vi nu förlänga sådan istället för att uttryck består av termer som består av faktorer, så består nu även faktorer av exponenter. Det betyder att exponenter kommer ha högst prioritet och beräknas först, enligt aritmetiska regler.