Compilatori

1) Eliminare la ricorsione a sinistra nella seguente grammatica:

$$A \rightarrow A + B \mid B$$
$$B \rightarrow int \mid (A)$$

2) Fattorizzare a sinistra la seguente grammatica:

$$E \rightarrow int \mid int + E \mid int - E \mid E - (E)$$

3) effettuare la traduzione in codice intermedio di

```
while(x < 3) z=1;
```

4) Data la seguente grammatica (**a,b** sono terminali):

$$S \rightarrow X\mathbf{a}$$
$$X \rightarrow \mathbf{a} \mid \mathbf{a}X\mathbf{b}$$

i) costruire l'automa LR(0) (indicare sono i viable prefix)
ii) usando le regole SLR(1), individuare eventuali conflitti **shift-reduce**
iii) risolvendo tutti gli eventuali conflitti **shift-reduce** scegliendo **shift**, mostrare tutti i passi del parser SLR(1) assumendo di avere in ingresso la stringa **aaba**

5) Data la seguente grammatica (**id** , **(** , **)** , **[** , **]** , **;** , sono terminali)

$$E \rightarrow \mathbf{id}\ X$$
$$X \rightarrow \varepsilon \mid (A) \mid [E]$$
$$A \rightarrow EY$$
$$Y \rightarrow \varepsilon \mid ; A$$

i) calcolare **FIRST** e **FOLLOW** per ogni non-terminale
ii) costruire la tabella LL(1)
iii) mostrare tutti i passi del parser LL(1) assumendo di avere in ingresso la stringa **id(id[id];id)**

6) considerare il seguente blocco di base:

```
a := b + c
z := a ** 2
x := 0 * b
y := b + c
w := y * y
u := x + 3
v := u + w
```

effettuare, nell'ordine:
i) semplificazioni algebriche;
ii) eliminazioni sottoespressioni comuni
iii) copy propagation
iv) constant folding