Domanda 1. (8 punti)

Usando le tabelle allegate, costruisci il codice a tre indirizzi del seguente frammento di programma:

```
 x = 5;
while ( x <= 10 )
     x = x + 1;
```

Spiegare i vari passi del procedimento a partire dalla costruzione del parse-tree.

Domanda 2. (8 punti)

Data la grammatica G:

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow a \\ B &\rightarrow b \\ C &\rightarrow AB \mid a \end{aligned}$$

(a) stabilire se G è LL(1) applicando la definizione;

(b) in caso di risposta positiva alla prima domanda, analizzare la stringa aab simulando il parser costruito con pila e tabella.

Domanda 3. (8 punti)

Data la grammatica G:

$$\begin{aligned} S &\rightarrow NV \mid I \\ N &\rightarrow L \mid LR \\ V &\rightarrow B \mid BN \\ I &\rightarrow V \\ R &\rightarrow NB \\ L &\rightarrow u \\ B &\rightarrow u \end{aligned}$$

(a) Dimostrare che G non è SLR, mentre la grammatica G' ottenuta da G modificando le produzioni per L e per B come segue, è SLR

$$\begin{aligned} L &\rightarrow u \\ B &\rightarrow v \end{aligned}$$

(b) Simulare il parser SLR su un input appropriato per dimostrare che G e G' non generano lo stesso linguaggio.

Domanda 4. (6 punti)

(a) Discutere l'implementazione del type checking nel compilatore per il linguaggio Fasto studiato in laboratorio.

(b) Scegliere una delle espressioni del linguaggio (definite in SML nel file `Fasto.sml` del progetto sviluppato in laboratorio) tra quelle riportate in Figura 1 e definire per quella espressione

- l'Interprete `evalExp`;
- il Type Checker.

```
datatype Exp =
...

| Iota of Exp * pos                                          (* iota(n) *)
| Map of FunArg * Exp * T.TypeAnnot * T.TypeAnnot * pos   (* map(f, arr) *)
| Reduce of FunArg * Exp * Exp * T.TypeAnnot * pos  (* reduce(f, 0, arr) *)
```

Figure 1: Frammento di `Fasto.sml`