Domanda 1. (8 punti)

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

```
z=10;
while (z > 0)
    z = z - x
else
    z=x;
```

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

Domanda 2. (8 punti)

Data la grammatica G:

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

(a) dimostrare che G non è LL(1);

(b) costruire la tabella di parsing SLR;

(c) individuare eventuali conflitti shift-reduce;

(d) simulare il parser SLR sulla stringa *bab*.

Domanda 3. (8 punti)

Considera la grammatica G:

$$\begin{aligned} E &\rightarrow -E \mid (E) \mid VF \\ F &\rightarrow -E \mid \varepsilon \\ V &\rightarrow a\,W \\ W &\rightarrow (E) \mid \varepsilon \end{aligned}$$

(a) Dimostrare, applicando la definizione, che G è LL(1);

(b) simulare il parser LL(1) su input `a(-a)`.

Domanda 4. (6 punti)

Considera il seguente frammento dell'implementazione del parser per Fasto incluso nel file `Parser.grm` del progetto sviluppato in laboratorio:

```
%token <(int*int)> PLUS TIMES LET IN EQ
%token <(int*int)> IF THEN ELSE LET IN INT
%token <string*(int*int)> ID

%nonassoc letprec
%left OR
%left AND
%left DEQ LTH
%left PLUS MINUS

%type <Fasto.UnknownTypes.Exp> Exp

Exp :        NUM            { Constant (IntVal (#1 $1), #2 $1) }
           | CHARLIT        { Constant (CharVal (#1 $1), #2 $1) }
           | TRUE           { Constant (BoolVal true, $1) }
           | FALSE          { Constant (BoolVal false, $1) }
           | ID                  { Var $1 }
           | Exp PLUS  Exp  { Plus ($1, $3, $2) }
           | Exp MINUS Exp  { Minus($1, $3, $2) }

            ...

          | IF Exp THEN Exp ELSE Exp %prec ifprec
                        { If ($2, $4, $6, $1) }
```

(a) Spiega l'uso di `%prec letprec`

(b) Completa lo schema con le altre espressioni:

```
| Exp TIMES Exp
| Exp DIV Exp
| Exp AND Exp
| Exp OR Exp
| Exp DEQ   Exp
| Exp LTH   Exp
| NOT Exp
| NEGATE Exp
| LET ID EQ Exp IN Exp
```