6 Primo riepilogo

Soluzioni

Soluzione dell'esercizio 6.1

```
int p, u; /* indice di piano nelledificio e di ufficio nel piano */

for (p = 0; p < 20; p++)
    for (u = 0; u < 40; u++)
        if ((torre[p][u].esp == sudEst || torre[p][u].esp == sud) &&
            (torre[p][u].superficie >=20 && torre[p][u].superficie<=30)) {
            printf("\n il Signor %s è impiegato di categoria %d",
                   torre[p][u].occupante.cognome,
                   torre[p][u].occupante.cat);
            printf("e ha uno stipendio pari a %d euro \n",
              torre[p][u].occupante.stipendio);
        }
```

```
int uffNord; /* uffNord fa da flag*/

for (p = 0; p < 20; p++) {
    uffNord = 0; //per ogni piano assumo 0
    for (u = 0; u < 40 && !uffNord; u++)
        if(torre[p][u].esposizione == nord)
            uffNord = 1;

    /* se qui vale ancora 0 vuol dire che non ci sono uffici a nord*/
    if (!uffNord)
        printf("il piano %d  non ha edifici esposti a nord", p);
 }

/* è corretto anche senza !uffNord nel for(), anche se non è efficiente. */
```

```
for(i = 0; i < 20; i++) { //scorre i piani
    noUfficiNord = 1; //versione con flag inversa

    for(j = 0; j < 40; j++) //senza flag arriva fino a 39
        if(torre[i][j].esposizione == nord)
            noUfficiNord = 0;

    if(noUfficiNord) {
        printf("il piano %d non ha uffici a nord", i);
        stipendioMedio = 0;
        cnt = 0;
        for(j = 0; j < 40; j++)
            if(strcmp(torre[i][j].occupante.nome, "Giacomo") == 0){
                stipendioMedio += torre[i][j].occupante.stipendio;
                cnt++;
            }
        printf("Lo stipendio medio dei Giacomo nel "
               "piano è %f ", stipendioMedio / cnt);
    }
 }
```

Soluzione dell'esercizio 6.2

```
/* prima variante */
```

```
s = 0;
p = 1;

for(i = 0; i < DIM ; i++) {
    s += m[i][i] * m[i][DIM - i -1];
    p *= m[i][i] + m[i][DIM - i -1];
 }

/* seconda variante con (due indici) */
s = 0;
p = 1;

for(i = 0, j = DIM-1; i < DIM, j > 0; i++, j--) {
    s += m[i][k] * m[i][j];
    p *= m[i][j] + m[i][j];
 }
```

Soluzione dell'esercizio 6.3

1. È vera perché è vero il secondo disgiunto (di cui è vero sia il primo congiunto, in quanto `c1` vale `'e'`, che il secondo congiunto, in quanto `c2` vale `'m'`). L'espressione, essendo vera, ovviamente non è sempre falsa. Inoltre non è sempre vera, per esempio sarebbe falsa se `c2` valesse `'k'` o qualsiasi altro valore diverso da `'m'`.

2. È sempre vera perché, in base alla legge di De Morgan, è equivalente a

   ```
   c1<'g' || c1>'g' || c1=='g'
   ```

   quindi la formula non è sempre falsa.

3. È sempre vera perchÃ¨ equivalente alla formula

   ```
   (c1 <= 'm') || (c2 > 'm') || c2 <= c1
   ```

   che è identicamente vera: infatti sarebbe falsa solo se fossero falsi tutti e tre i suoi disgiunti, cosa che non può essere, perché se sono falsi i primi due (cioè se `c1>'m'` e `c2<='m'`) allora `c2<c1`, quindi il terzo è vero. Quindi la formula non è sempre falsa.

Soluzione dell'esercizio 6.4

```
int i, j, s;
s = 1;
i = 3;

while (i >= 0) {
  j = 3;
```

```
  while (j >= 0) {
    if ((i+j) % 2 != 0 )
      s = s * 2;

    j--;
  }

  i--;
}
```

Il programma raddoppia il valore di `s` per un numero di volte pari alla quantità di coppie di numeri tra 0 a 3 la cui somma è dispari. Questa condizione è verificata per 8 coppie ($s^8 = 256$).

La stessa conclusione si può trarre calcolando i valori

i	j	s
3	3	1
3	2	2
3	1	2
3	0	4
2	3	8
2	2	8
2	1	16
2	0	16
1	3	16
1	2	32
1	1	32
1	0	64
0	3	128
0	2	128
0	1	256

Soluzione dell'esercizio 6.5

```
simmetrica = 1;

for(i = 0; i < D && simmetrica; i++)
    for(j = 0; j < i && simmetrica; j++)
        if(m[i][j] != m[j][i])
            simmetrica = 0;

if(!simmetrica)
    printf("NON ");
```

```
printf(" simmetrica");
```

La condizione `j < i` serve per far scorrere l'indice `i` solo nella prima metà della matrice.