Corso di Linguaggi di Programmazione
Prova scritta del 6 Febbraio 2024.

Tempo a disposizione: ore 2.
Svolgere gli esercizi 1–4, 5–6 e 7–8 su tre fogli separati.
Scrivere nome, cognome e matricola su ogni foglio consegnato.

FOGLIO 1 ▷ 1. Si consideri la grammatica G

$$C \rightarrow \quad a \,|\, b \,|\, C;C \,|\, (C)$$

che esprime comandi composti sequenzialmente a partire istruzioni elementari a e b, non meglio specificate. Si dimostri che la grammatica G è ambigua. Definire le regole di semantica operazionale strutturata (SOS) per il costrutto $C_1;C_2$.

FOGLIO 1 ▷ 2. Dati due DFA $M_1 = (\Sigma, Q_1, \delta_1, q_{01}, F_1)$ e $M_2 = (\Sigma, Q_2, \delta_2, q_{02}, F_2)$ tali che $Q_1 \cap Q_2 = \emptyset$, sia $M = (\Sigma, Q_1 \times Q_2, \delta, (q_{01}, q_{02}), F_1 \times F_2)$ con $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$. Dire se M è un DFA. Qual è il linguaggio riconosciuto da M se $L_1 = L[M_1]$ e $L_2 = L[M_2]$?

FOGLIO 1 ▷ 3. Si costruisca il più semplice automa che riconosca il linguaggio $L = \{a^n b^m \,|\, n \geq 1, m \geq 0\}$. Ricavare da tale automa una grammatica equivalente.

FOGLIO 1 ▷ 4. Si costruisca un parser LL(1) che riconosca il linguaggio $L = \{a^n c b^n \,|\, n \geq 1\}$ e si mostri il suo funzionamento su input acb.

FOGLIO 2 ▷ 5. Si dica cosa è l'aliasing e perchè può rendere difficoltosa la verifica della correttezza dei programmi. È possibile inserire nel compilatore di un ipotetico linguaggio un controllo che permetta di identificare tutte le situazioni di aliasing? Perchè?

FOGLIO 2 ▷ 6. Si dica cosa viene stampato dal seguente frammento di codice scritto in uno pseudo-linguaggio che usa scoping dinamico e passaggio di parametri per nome. La primitiva `write(x,y,z)` permette di stampare i valori dei tre argomenti.

```
{int x = 2;
 int y = 5;
 int z = 10;
void pippo(name int v, name int w){
     int x = 1000;
     w = v;
     v = v+ w + (z++) + z;
     z = 1000;
     }
   { int x = 20;
     int y = 50;
     int z = 100;
     pippo(x, y);
     write(x,y,z);
   }
write(x,y,z) }
```

FOGLIO 3 ▷ 7. Si assuma di avere uno linguaggio che usa la tecnica locks and keys. Dato `OGG` generico oggetto nello heap, indichiamo con `OGG.lock` il suo lock (nascosto). Data `p` variable contente il valore di un generico puntatore (sulla pila o nello heap), indichiamo con `p.key` la sua key (nascosta). Il linguaggio supporta la gestione manuale della memoria (ad es., `free` libera la memoria). Si consideri il seguente frammento di codice:

```
class C { C next; }
C p = new C(); // nuovo OGG (OGG1)
for( int i = 0, i < 2, i++ ){
 C q = new C(); // nuovo OGG
 q.next = new C(); // nuovo OGG
 p.next = q.next;
}
```

Si rappresentino graficamente gli oggetti (`OGG1`, `OGG2`, `OGG3`, ...) generati in memoria e lo schema di lock e key stabilito tra di essi dopo l'esecuzione del frammento (spiegare brevemente il ragionamento seguito).

FOGLIO 3 ▷ 8. Si assuma un linguaggi di programmazione a oggetti, con tipi nominali e passaggio per riferimento. Le classi `A`, `B`, e `C` sono tali che `B` è sottoclasse di `A` e `C` è sottoclasse di `A`. Nel linguaggio, il tipo `T[]` indica un array di oggetti della classe `T` con scritture e letture covarianti rispetto ai sottotipi. Indicare quali istruzioni verrebbero segnate come *non* corrette dal controllore dei tipi (e indicare brevemente perché).

```
void f( A a, B b, C c, A[] aa, B[] bb, C[] cc ) {
 b = c;              // I1
 c = b;              // I2
 a = c;              // I3
 a = bb[ 0 ];        // I4
 aa = bb;            // I5
 cc = aa;            // I6
 bb = bb;            // I7
 aa[ 0 ] = cc;       // I8
 aa[ 0 ] = cc[ 0 ]; // I9
 bb[ 0 ] = a;        // I10
 c = bb[ 0 ];        // I11
}
```