Corso di Linguaggi di Programmazione
Prova scritta del 27 maggio 2022 .

Tempo a disposizione: ore 2.
Svolgere gli esercizi 1-4 e 5-8 su due fogli differenti.

1. Si consideri l'iterazione determinata, realizzata mediante un comando della forma

   ```
   for I = inizio to fine by passo do C
   ```

   dove l'indice `I` è una variabile che non può essere modificata (nè esplicitamente, nè implicitamente) nel corso dell'esecuzione del corpo `C`, mentre `inizio`, `fine` e `passo` sono delle costanti.

   Si dica, motivando la risposta, se un tale comando può essere espresso mediante un comando `while` e, vice versa, se un comando `while` può essere espresso da un comando di iterazione determinata nel contesto un linguaggio imperativo che abbia anche i comandi di assegnamento, sequenza e condizionale.

2. È dato il seguente frammento di codice in uno pseudolinguaggio con `goto`, scope dinamico e blocchi annidati etichettati (indicati con `A :{...}`):

   ```
   A: { int x = 5;
        int y = 4;
        goto B;
           B: {int x = 4;
               int z = 3;
               goto C;
               }
           C: {int x = 3;
               D: {int x = 2;
                   }
               goto E;
               }
            E: {int x = 1; // (**)
                }

        }
   ```

   Lo scope dinamico è gestito mediante CRT. Si illustri graficamente la situazione della CRT nel momento in cui l'esecuzione raggiunge il punto segnato con il commento `(**)`.

3. La definizione di un certo linguaggio di programmazione specifica che la valutazione procede da sinistra a destra, ma nel valutare un'espressione complessa eventuali sottoespressioni che vi compaiono più di una volta devono essere valutate una sola volta, usando il valore così calcolato anche per le altre occorrenze della stessa sottoespressione. Un assegnamento è una particolare forma di espressione complessa e il passaggio dei parametri avviene per nome. Si consideri il seguente frammento di codice:

   ```
   int x = 1;
   int A[5];
   for (int i=0; i<5; i++)
      A[i] = i;

   void f(int name x, int name y){
      x= y + x + x;
      }
      f(A[x++],x)
   ```

   Qual'è lo stato del vettore `A` dopo la chiamata della funzione `f`?

4. Si consideri il seguente frammento in uno pseudolinguaggio con parametri di ordine superiore:

```
{int m = 1;
 int n = 2;
 void foo (int f(), int n){
        int m = 3;
        int g(){
          write(n,m);
         }
    if (n==0) { f();
                g();
               }
    else      {  int m = 5;
                 int n = 6;
                 f();
                 foo(g,0);
                }
    }

 {int m = 10;
  int n = 11;
  int g(){
          write(n,m);
         }
  foo(g,1);
  }
}
```

Si dica cosa stampa il frammento con scope statico e deep binding, motivando brevemente la risposta.

5. Il frammento di codice sotto è scritto in un linguaggio che usa passaggio per valore e dove `T*` è il tipo "puntatore generico a un valore di tipo `T`"; `T[]` è il tipo "puntatore ad un array di elementi di tipo T"; `print` è una funzione che stampa il valore dei parametri dati in input, separati da virgole; `&var` e `*var` corrispondono rispettivamente a referenziamento e dereferenziamento di una `var`iable.

   Dire se nel frammento di codice indicato l'utilizzo fatto dei valori di tipo riferimento (reference types) introduce violazioni della memoria. Nel caso ve ne siano, indicare una possibile correzione minima del codice e riportare cosa stampa tale frammento, spiegando brevemente i passaggi del ragionamento seguito.

```
void foo( int* a, int* b, int* j ){
  a[ *j ] = *j + 1;
  a++;
  a[ *j ] = *j;
  *j = *j + 1;
}

int[] a[ 3 ];
int* j;
{
  int i = 0;
  j = &i;
  a[ 0 ] = 1;
  a[ *j ] = 0;
}
foo( a, &a[ *j ], j );
print( a[ 0 ], a[ *j ] );
```

6. Si consideri un linguaggio con passaggio per valore nel quale le eccezioni: devono essere dichiarate con la sintassi `exception E` con `E` nome dell'eccezione; vengono sollevate con l'istruzione `throw E`; vengono gestite coi blocchi `try{ ... } catch E { ... }`. Il linguaggio ha scoping statico per tutti i nomi, eccezioni comprese. Cosa stampa (tramite l'operazione `print`) il seguente frammento? Spiegare brevemente il ragionamento dietro la risposta.

```
exception Y;
void a(){ throw Y; }
exception X;
void b( int x ){
 if( x < 4 ){ a(); } else { throw X; }
}
int c( int y ){
 try { b( y ); } catch X { return y - 2; }
}
int d( int z ){
 try { return c( z ); }
 catch Y { return 10 + d( z + 1 ); }
}
print( d( 0 ) );
```

7. Si assuma di avere uno linguaggio che adotti 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). Si consideri il seguente frammento di codice:

```
class C {
 C next;
}
C p = new C();         // oggetto OGG1
C q = new C();         // oggetto OGG2
{ p.next = new C(); } // oggetto OGG3
q.next = p.next;
```

   Si diano possibili valori di `OGG1.lock`, `OGG2.lock`, `OGG3.lock`, `p.key`, `p.next.key`, `q.key`, `q.next.key` dopo l'esecuzione del frammento (spiegare brevemente il ragionamento seguito).

8. Si rappresenti e descriva la struttura delle *vtable* corrispondenti alle seguenti dichiarazioni di classi, assumendo ereditarietà singola e che `class B extends A` indica un rapporto di ereditarità dalla classe `A` verso la classe `B`.

```
class A {
 int x = 1;
 int h( int n ){ return n + 1; }
}
class B extends A {
 int x = 1;
 int g(){ return f( 3 ); }
}
```

   Inoltre, dopo aver compilato entrambi le classi `A` e `B`, supponiamo di modificare la classe `A` come segue

```
class A {
 int x = 1;
 int h(){ return 4; }
 int f( int n ){ return n + 1; }
}
```

   possiamo ricompilare solo `A` e usare con sicurezza il compilato ottenuto in precedenza della classe `B`? Motivare la risposta.