Corso di Linguaggi di Programmazione
Prova scritta del 27 Maggio 2025.

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

FOGLIO 1 ▷ 1. Si dica cosa stampa il seguente frammento in uno pseudolinguaggio con passaggio per nome e scope dinamico

```
int x = 10;
void foo(name int y){
    int z;
    z = y++;
    z = y++;
    x = x+10;
    write(x);
}
{int x = 50;
 foo(x);
 write(x); }
```

FOGLIO 1 ▷ 2. Supponiamo che il linguaggio imperativo `Ric` non permetta comandi di allocazione (e deallocazione) esplicita della memoria e, oltre agli usuali comandi, permetta solo la ricorsione in coda. Si dica, motivando la risposta, qual'è la più semplice forma di gestione della memoria utilizzabile nell'implementazione di `Ric`.

FOGLIO 1 ▷ 3. L'esecuzione del seguente frammento di codice su una certa implementazione risulta nella stampa del valore 13.

```
int V[10];
int x = 5;
for (int i=0, i<10, i++) V[i]=i;
void foo(int y){
    V[x] = V[x] + V[y];
    V[x] = V[x] + V[y];
}
 foo(x++);
 write(x);
}
```

Si fornisca una possibile spiegazione, specificando che tipo di passaggio di parametri è stato usato.

FOGLIO 1 ▷ 4. Si consideri il seguente frammento in uno pseudolinguaggio con parametri di ordine superiore:

```
{void foo (int f(), int n){
    int m = 10;
    int fie(){
      write(n,m);
    }
    if (n==0) {int n = 20;
              f();
              }
    else {foo(fie,0);
      }
 }
 int g(){
    write(10);
 }
 int m = 100;
 foo(g,1);
}
```

Si dica cosa stampa il frammento con scope dinamico e shallow binding.

FOGLIO 2 ▷ 5. Descrivere brevemente i possibili problemi di gestione della memoria tramite puntatori nel codice C sotto.

```
1  int* d = malloc( sizeof( int ) );
2  int* c;
3  { int e = 1;
4   *d = 3;
5   free( d );
6   int* b = malloc( sizeof( int ) );
7   c = &e; }
8  int* a;
9  *d = 4;
10 free( d );
11 ...
```

FOGLIO 2 ▷ 6. Un tipo di dato chiamato "PolyForest" rappresenta una collezione di alberi interconnessi tramite rami. In PolyForest, un ramo può portare ad un altro albero, così che un valore PolyForest corrisponde ad un albero "radice" che da accesso ad altri PolyForest tramite i propri rami. Gli alberi in PolyForest sono di tre sottospecie: "Binary", con due rami che portano a due PolyForest, "Multi" che hanno un numero illimitato di rami, ognuno che porta ad un PolyForest, e "Strange" con un numero illimitato di rami ma non tutti questi portano ad un PolyForest. Scrivere, usando tipi unità, somma e prodotto, la definizione di PolyForest, descrivendo brevemente il ragionamento seguito.

FOGLIO 2 ▷ 7. In un pseudolinguaggio che supporta polimorfismo di sovraccarico, la notazione `a[i]` indica l'accesso al valore `i`-esimo del vettore `a` e `[1,2,3]` definisce un vettore contente gli elementi 1, 2 e 3. Abbiamo inoltre:

- `a[i,j]` permette di ottenere un vettore "tagliando" un vettore `a` dall'indice `i`, incluso, all'indice `j`, escluso, o una stringa "tagliando" una stringa `a` dalla posizione `i`, inclusa, alla posizione `j`, esclusa;
- `length` ritorna la lunghezza rispettivamente di una stringa o di un vettore passati come parametro;
- `+` somma due interi o concatena due stringhe (spec. quella a sinistra seguita da quella a destra).

Indicare, riportando brevemente il ragionamento seguito, cosa stampa (`print`) il codice seguente.

```
a = "trams"
b = [ 0, 1, 2, 3, 4 ]
print( a( a, b ) )

string a( string a, int b){ return a[ b, b+1 ] }
int a( string a, string b){ return length( a ) }
int a( int[] a, int b ) { return length( a ) }
int a( int[] a, int[] b){ return a[ a( b, a[0] ) - 1 ] }
string a(string a, int[] b) {
  if ( a( b, a( a, a ) ) > 0 ){
    return a( a, a( b, b ) ) + a( a, a( b, a ) )
  } else { return "" }
}
int[] a( int[] a, string b ) { return a[ 0, a( a, a ) ] }
```

FOGLIO 2 ▷ 8. Si 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 f( int n ){ return x; }
}
```

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

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

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

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