Corso di Linguaggi di Programmazione
Prova scritta del 3 Febbraio 2026.

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. Descrivere le regole di semantica operazionale strutturata per l'espressione booleana b_0 `xor` b_1, secondo la disciplina di valutazione esterna-sinistra (ES). Per espressioni di questo tipo, la valutazione ES e quella IS (interna-sinistra) danno sempre lo stesso risultato? Motivare la risposta.

FOGLIO 1 ▷ 2. Fornire una definizione regolare per la categoria sintattica `Ide`, dove un identificatore è una qualunque sequenza su alfabeto $A = \{a,...,z,A,...,Z\} \cup \{0,1,...,9\} \cup \{!,?\}$ tale che comincia con una lettera minuscola, contiene almeno una lettera maiuscola o una cifra, e termina con un simbolo non alfanumerico.

FOGLIO 1 ▷ 3. Sia $L_1 = \{a^{2n}b^{2m} \mid n,m \geq 0\}$ e $L_1 \cap L_2 = \{a^{4n}b^{2n} \mid n \geq 0\}$, come è definito L_2? A quali classi appartengono i linguaggi L_1,L_2 e $L_1 \cap L_2$?

FOGLIO 1 ▷ 4. Si consideri la grammatica G con simbolo iniziale S:

$$\begin{array}{lcl} S & \rightarrow & \texttt{b}A\texttt{d} \\ A & \rightarrow & \texttt{a} \mid \texttt{a}B\texttt{a} \\ B & \rightarrow & \epsilon \end{array}$$

(i) Quale linguaggio genera G? (ii) Costruire un parser di classe LL(1) per il linguaggio $L(G)$.

FOGLIO 2 ▷ 5. Si consideri il seguente frammento di codice, assumendo venga usata la regola di scope dinamico e si permetta il passaggio di funzioni come parametro.

```
x := 1
procedure PrintX()
    print(x)
procedure ChangeAndCall(f)
    x := 3
    f()
procedure Store(f)
    x := 5
    g := f
    return g
procedure Main()
    x := 10
    h := Store(PrintX)
    ChangeAndCall(h)

Main()
```

Cosa viene stampato con shallow binding? Cosa viene stampato con deep binding? Motivare brevemente le risposte.

FOGLIO 2 ▷ 6. Si consideri il seguente frammento di codice scritto in un linguaggio imperativo che usa scope statico, passaggio dei parametri per valore e pila dei RDA classica (senza ottimizzazioni automatiche e senza trasformazioni della tail recursion)

```
function F(n, acc)
    if n == 0 then
        return acc
    else
        return F(n - 1, acc + n)


function G(n)
    acc := 0
    while n > 0 do
        acc := acc + n
        n := n - 1
    return acc
```

Dal punto di vista semantico, le due funzioni sono sempre equivalenti? Se si, specificare sotto quali ipotesi sul linguaggio. Se no, fornire un controesempio.

Dal punto di vista della gestione e dell'uso della memoria ci sono differenze fra le due funzioni? Discutere brevemente.

FOGLIO 3 ▷ 7. È dato il seguente frammento di codice in uno pseudolinguaggio con passaggio per riferimento e garbage collection con contatore dei riferimenti; nel linguaggio, **new** alloca dinamicamente memoria nello heap.

```
type A = struct { A next; }
A f(){
  A p = new A();
  A q = new A();
  p.next = q;
  return p;
}
A g( A t ){
  A a = new A();
  a.next = t;
  A out = t;
  return out;
}
A x = g( f() );
x = g( x.next );
```

(i) Quanti oggetti di tipo A sono creati sullo heap in totale? (ii) Si rappresentino le configurazioni della memoria ai punti di chiamata e assegnamento significativi (ad es., `f`, `g`, assegnamento di `x`), mostrando uno schema degli oggetti in memoria e il loro valore del contatore, in modo da illustrare l'evoluzione della memoria e il risultato finale ottenuto al termine del frammento.

FOGLIO 3 ▷ 8. Viene dato uno pseudolinguaggio con tipaggio nominale, tipi base come `String` e `Int`, prodotto `A x B` e somma `A + B` e passaggio per riferimento senza riferimenti nulli. Si codifichino i due tipi `Team` e `Risorsa`. `Team` ha un nome, può avere una lista di `sottoteam`, di tipo `Team`, e può avere una lista di risorse allocate di tipo `Risorsa`. `Risorsa` ha un nome e può avere una lista di assegnatari, di tipo `Team`. Oltre alla definizione dei tipi, scrivere il codice che genera correttamente (anche per il type checker) i valori: `Team` "Backend" che ha come `sottoteam` "API" e "Database", dove "API" e "Database" utilizzano entrambi la stessa `Risorsa` "Server", di cui sono assegnatari. Per la creazione dei valori è possibile usare la notazione `Tipo v = new Tipo { a: new Tipo ..., ..., z: e }` e `v.a = ....`