

Corso di Laurea in Informatica

Linguaggi

Esame del 27 Giugno 2025

Si ricorda che ogni risposta va giustificata ed ogni concetto spiegato nel corso che viene citato va definito. Per ogni esercizio si indica tra parentesi il suo valore in 30-esimi.

1. ~~(*)~~ Definire intuitivamente e formalmente (mediante definizione semantica) cosa è un interprete. Scrivere la struttura di un interprete e spiegarne il funzionamento.
2. ~~(*)~~ Si consideri l'insieme E definito induttivamente: $n \in E$, e se $e_1, e_2 \in E$ allora $e_1 \wedge e_2 \in E$, $e_1 \vee e_2 \in E$. Si definiscano sugli elementi di tale insieme le seguenti funzioni: $v(n) = 1$, $v(e_1 \wedge e_2) = v(e_1 \vee e_2) = v(e_1) + v(e_2)$, e $o(n) = 0$, $o(e_1 \wedge e_2) = o(e_1 \vee e_2) = o(e_1) + o(e_2) + 1$. Si dimostri per induzione strutturale che $\forall e \in E. v(e) = o(e) + 1$.
3. ~~(*)~~ Si consideri il programma sulla destra. Si dica cosa viene calcolato dall'assegnamento in caso di scoping statico (mostrando come vengono calcolati i link statici e come vengono risolti riferimenti non locali) e in caso di scoping dinamico (mostrando l'evoluzione della tabella centrale dei riferimenti (CRT) e come vengono risolti riferimenti non locali).

```
{int a = 2; int b = 3;
void fun(){
    int z = a * b;}
void fun2(){
    int a = 4; int c = 5;
    void fun3(){
        int a = c - b;
        fun();}
    b = a + b;
    fun3();}
fun2();}
```

RICA 1
RICA 2
RICA 3
RICA 4
RICA 5
RICA 6
RICA 7
RICA 8
RICA 9
RICA 10
RICA 11

4. ~~(*)~~ Si consideri lo schema del seguente codice, nel quale vi sono due buchi indicati rispettivamente con (*) e (**). Si dia il codice da inserire al posto di (*) e (**) in modo tale che:
 - Se il linguaggio usato adotta lo scope statico, le due iterazioni assegnino a x lo stesso valore;
 - Se il linguaggio usato adotta lo scope dinamico, le due iterazioni assegnino a x valori diversi.

Si descrivano dettagliatamente le scelte fatte e si descriva cosa avviene a tempo di esecuzione in entrambi i casi e perché.

```
{ int i=0; int x:=3;
(*) void pippo(){
    int y;
    (**);
    x:=pluto(); i:=i+2;
}
while(i≤ 2){pippo();}}
```

5. ~~(*)~~ Definire brevemente il concetto di ricorsione e di ricorsione in coda. Si consideri il programma ricorsivo a destra e si descriva che cosa calcola mostrando anche il calcolo sulla lista (2, 4, 6). Trasformare quindi tale programma in un programma ricorsivo in coda. Mostrare poi il funzionamento dell'algoritmo fornito sulla lista (2, 4, 6).

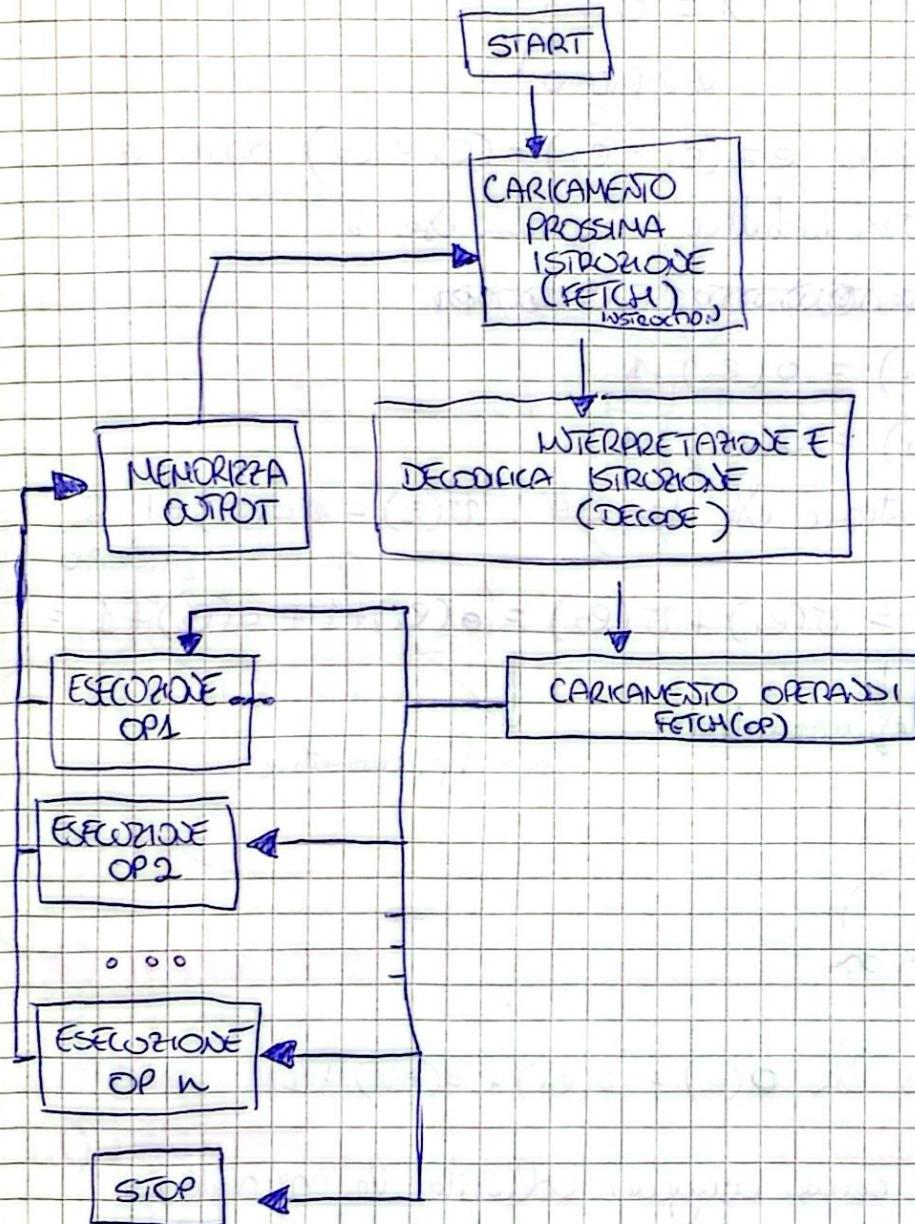
```
lista function (lista list){
    if (length(list) = 0) then return ();
    else return concat(car(list) * 3, function(cdr(list)));
}
```

NOTA: Data una lista $list$, $length(list)$ restituisce la sua lunghezza, $car(list)$ restituisce il primo elemento, $cdr(list)$ restituisce tutta la lista tranne il primo elemento, $concat$ concatena due liste (se un elemento non è una lista, $concat$ lo converte prima in lista). () è la lista vuota.

6. ~~(*)~~ Siano $\rho = [x \mapsto 3, y \mapsto l_y]$ un ambiente dinamico con l_y locazione di tipo int, e $\sigma = [l_y \mapsto 2]$ una memoria. Descrivere le derivazioni della semantica dinamica per il comando $y := 3x + y$ nell'ambiente ρ a partire dalla memoria σ ($\rho \vdash (y := 3x + y, \sigma) \rightarrow \dots$).

Esercizio 1) Un interprete è un programma scritto in L₀, che esegue un altro programma, scritto in un linguaggio L, attraverso un input fornito, usando solo le primitive fornite da L₀. Formalmente è → $I^{L_0} = (\text{Prog } L \times \text{Dato}) \rightarrow \text{Dato}$.

Flow-chart :



ES 2) ~~2.0072~~ ~~Passare~~
~~alla prossima~~
~~riga~~

Dimostriamo che $\forall e \in E. v(e) = o(e) + 1$, tramite induzione strutturale.
caso base: prendiamo $e = n$ e abbiamo

$$\begin{aligned} v(n) &= 1 \\ o(n) &= 0 \end{aligned} \quad \xrightarrow{\text{quindi,}} \quad v(n) = o(n) + 1$$
$$1 = 0 + 1 \quad \checkmark$$

VERIFICATO!

Passo Induttivo: Prendendo $e = (e_1 \wedge e_2) = (e_1 \vee e_2)$ abbiamo
~~che~~ che, come ipotesi induttive, sappiamo che:

- 1) $v(e_1) = o(e_1) + 1$
- 2) $v(e_2) = o(e_2) + 1$

Portiamo con il dimostrare che $\forall e \in E. v(e) = o(e) + 1$:

$$v(e) = v(e_1 \wedge e_2) = v(e_1) + v(e_2) = o(e_1) + 1 + o(e_2) + 1 =$$

\downarrow definizione di v \downarrow per hyp. induttive

$$= o(e) + 1 = \boxed{v(e)}$$

\downarrow siamo partiti da

\downarrow per definizione di o dato che $o(e) = o(e_1) + o(e_2) + 1$

ES 3) Lo scopo è il come vengono valutate le variabili in programmi dove vi sono contesti non locali. Si necessita quindi definire l'ambiente associato per risolvere i riferimenti, come detto, non locali (non presenti nella funzione in cui vengono usate). Per fare ciò, abbiamo due metodi/approcci:

- SCOPING STATICO: si valutano le variabili nel momento in cui sono definite, ossia a tempo di compilazione. Utilizzando la cattedra statica, collegamento delle stesse i contesti lessicali statici, si trova l'ambiente di definizione corretto.

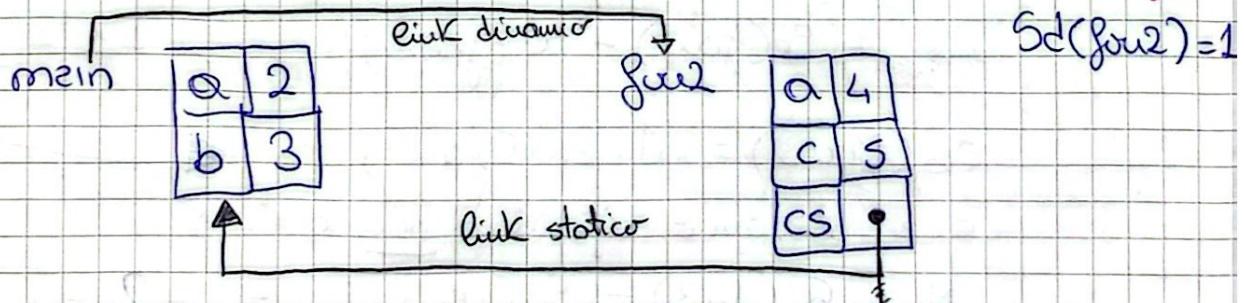
- SCOPING DINAMICO: si valutano le variabili nel momento delle chiamate o fruizioni, ossia a tempo di esecuzione (run-time). Si utilizza la CRT, tabella centrale dei riferimenti, che tiene traccia dei riferimenti fatti durante l'esecuzione alle varie variabili, partendo dalla più recente.

VERSIONE SCOPING STATICO:

1) MAIN INIZIALIZZA (RIGA 1): $Sd(\text{main}) = 0$

main	a	2
	b	3

2) MAIN CHIAMA FUNZIONE FUN2, CHE INIZIALIZZA ALTRE VARIABILI (RIGA 4 e 5):



$K_{\text{fun2}} = Sd(\text{chiamante}) - Sd(\text{chiamato}) + 1$

$= Sd(\text{main}) - Sd(\text{fun2}) + 1 = 0 - 1 + 1 = 0$

$CS(\text{fun2}) = \text{indirizzo}(\text{main})$

3) FUN2 ESEGUE $b = a + b$: (RICA10)

Per fun2, 'b' non è locale. Calcoliamo quindi

$$Nb = Sd(\text{chiamata}) - Sd(\text{definizione}) = Sd(\text{fun2}) - Sd(\text{main}) =$$

\downarrow
chi utilizza la variabile

\downarrow
che è stata
definita la variabile

$$= 1 - 0 = 1$$

$$b = a + b$$

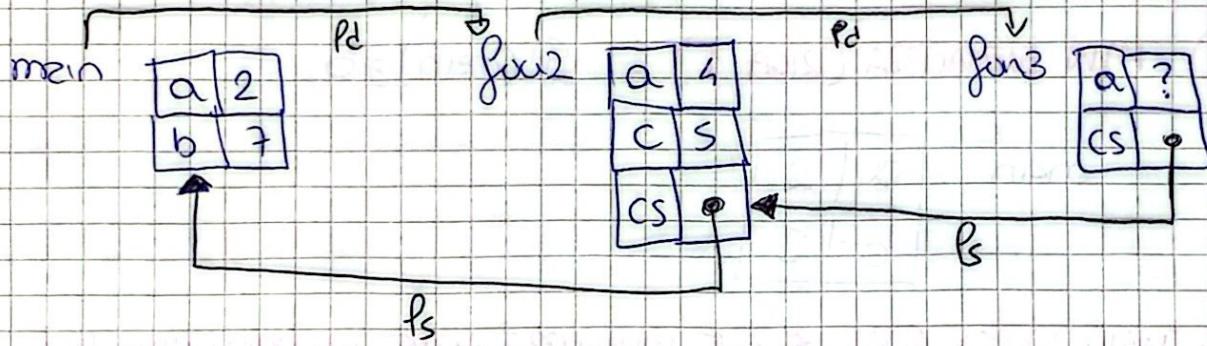
\downarrow
LOCALE (VALE 4)
 $\xrightarrow{\text{da main, code 3}}$

$$b = 4 + 3 = 7$$

(RICA10, 6 e 7)

main	a	2	ooo (resto inviato)
	b	3	7

4) FUN2 CHIAMA FUN3, DEFINISCE ~~COMUNO~~ ESEGUE $a = b \circ c - b$:



$$K_{fun3} = Sd(fun3) - Sd(fun2) \stackrel{+1}{=} 1 - 2 + 1 = 0$$

CS (fun3) = indirizzo (fun2)

'c' e 'b' fanno riferimenti non locali :

$$Nb = Sd(fun3) - Sd(main) = 2 - 0 = 2 \rightarrow \text{trovi in main}$$

$$Nc = Sd(fun3) - Sd(fun2) = 2 - 1 = 1 \rightarrow \text{trovi in fun2}$$

$$a = c - b$$

\downarrow

$$a = 5 - 7 = -2$$

ooo
(invia
resto)

fun3	a	-2
	c	s

MARTIN PEDRON GIUSEPPE

RICA 2 e 3

VR471448

LAVORAGGI - FOGIO 2

5) FUNZIONE INVOCA FUNZIONE ESECUDE $[z = a * b]$:

$$Sd(f_{fun}) = 1$$

'a' e 'b' ambe var locali.

$$Na = Sd(f_{fun}) - Sd(\text{main}) = 1 - 0 = 1$$

$$Nb = Sd(f_{fun}) - Sd(\text{main}) = 1 - 0 = 1$$

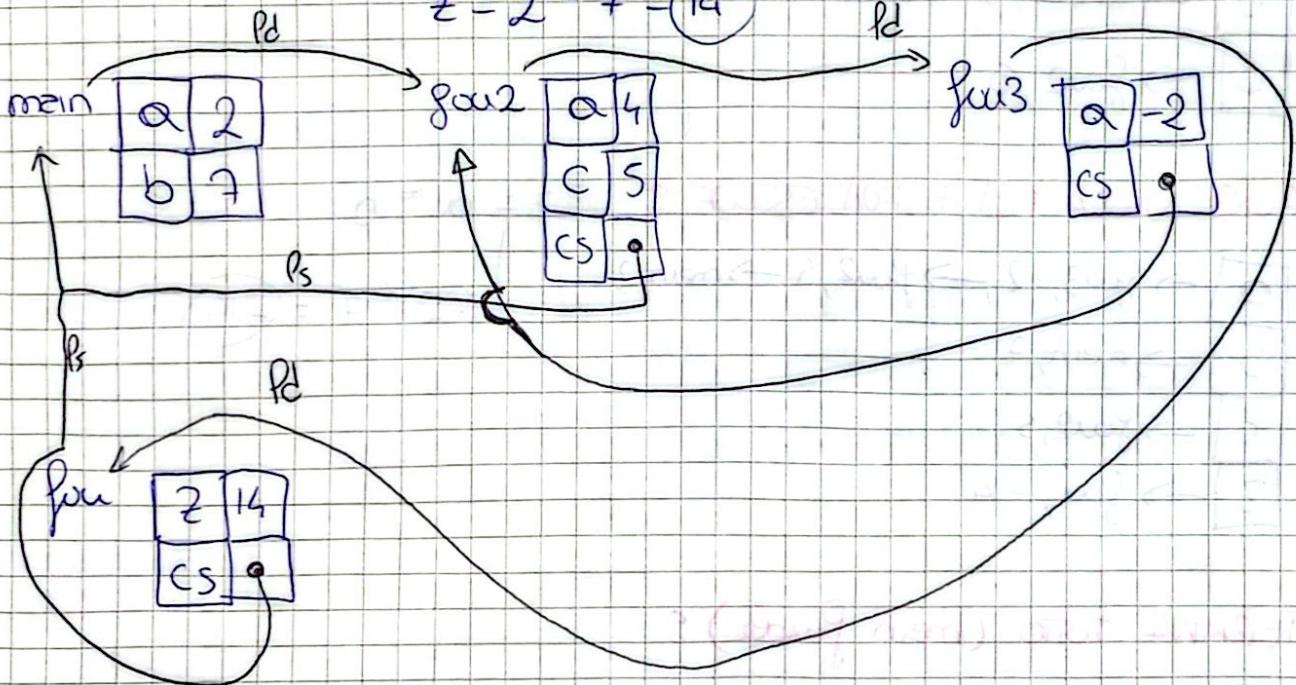
$$K_{fun} = Sd(f_{fun3}) - Sd(f_{fun}) + 1 = 2 - 1 + 1 = 2$$

$$CS(f_{fun}) = CS(CS(f_{fun3})) = CS(\text{Indirizzo}(f_{fun2})) = \text{Indirizzo}(\text{main})$$

quindi

$$z = a * b$$

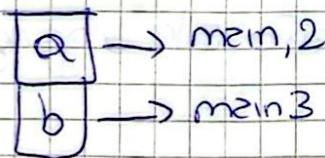
$$z = 2 * 7 = 14$$



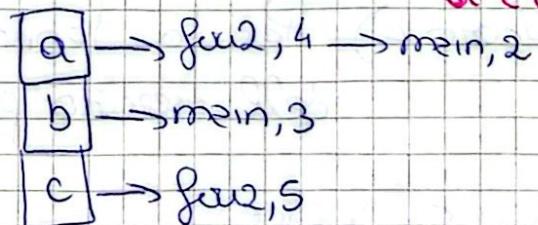
6) TERMINA TUTTO

VERSIONE SCOPING DINAMICO:

1) MAIN DEFINISCE a e b :



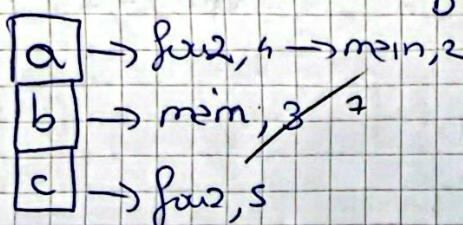
2) MAIN INVOCA FUN2 + FUN2 DEFINISCE a e c



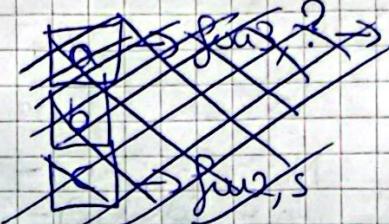
3) FUN2 ESEGUE:

$$b = a + b$$

4) FUN2 NONA FUN3, ~~NON~~



$$b = 4 + 3 = 7$$

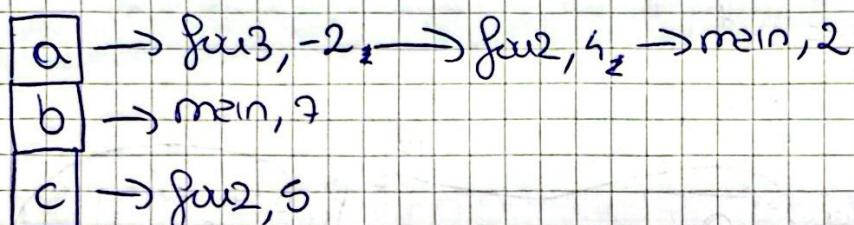


CRT USUALE A PRECEDENTE!

5) FUN3 ESEGUE:

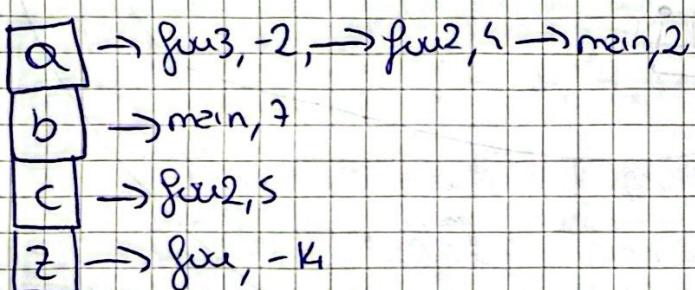
$$a = c - b$$

$$a = 5 - 7 = -2$$



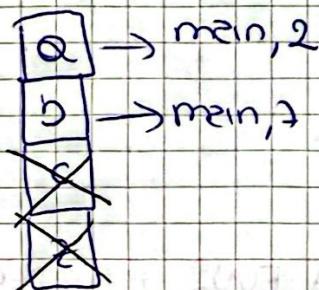
6) FUN3 NONA FUN + FUN ESEGUE:

$$z = a * b$$



$$= (-2) * 7 = \boxed{-14}$$

7) TERMINA TUTTO (main finale):



Esercizio 4) Per la teoria si faccia riferimento allo scoping dell'esercizio precedente!

Nuovo codice:



```

    ↓
{ int i=0;
  int x = 3;      (*)
  int y = i;
  int pluto() {
    return y;
  }
}

```

NB: cerchiamo in verde le parti aggiunte!

```

void pippo() {
  int y;
  y = i;          (**)
  x = pluto();
  i = i+2;
}
while (i <= 2) {
  pippo();
}

```

SCOPING STATICO: l'ambiente in scoping statico a cui 'pluto' fa riferimento è:

$$Sc(pippo) - Sc(pluto) + 1 = 1 - 1 + 1 = 1$$

$$Cs(pluto) = \text{main} \quad Cs(pippo) = \text{main}$$

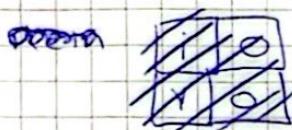
Per cui il 'return y' di pluto fa riferimento alla 'y' del main che è statica ' $\neq 0$ ', dato che viene eseguito (l'assegnamento $y = i$) solo una volta e quando appunto $i = 0$. Per cui anche iterando e modificando i , la y a cui pluto fa riferimento è quella aggiunta globale e soprattutto statica (non modificata)!

SCOPING DINAMICO: con lo scoping dinamico, l'ambiente a cui pluto fa riferimento entrambe le volte in cui viene eseguito è quello della sua funzione chiamante, pippo. Anche qui la y viene

assegnata = i, ma tuttavia questa assegnazione verrà eseguita 2 volte (chiamata da parte dell'while). Nella prima iterazione i vale 0 (è sempre quello globale, ma pippo viene chiamato prima che i venga modificata). Nella 2^a iterazione, i vale 2 ($i = i + 2$, quindi anche a y viene dato quel valore. $i = 0 + 2 = 2$),

Le 2 esecuzioni di pippo quindi, hanno una x con valori differenti rispetto a pippo.

ALLA 1^a CHIAMATA DI PIPPO:



$i \rightarrow \text{min}, 0$

$y \rightarrow \text{pippo}, 0 \rightarrow \text{min}, 0$

quindi $x = 0$ alla fine!

AUT 2^a CHIAMATA DI PIPPO

$i \rightarrow \text{min}, 2$

$y \rightarrow \text{pippo}, 2 \rightarrow \text{min}, 0$

quindi $x = 2$ alla fine!

questa sarebbe quella di cui si parla nello scoping STANDO di prima

ESECUZIONE 5) La ricorsione è una funzione che chiama se stessa per risolvere un problema/calcolo, suddividendo in versioni sempre più piccole del problema fino ad arrivare ad un caso base, dove la ricorsione termina e il calcolo può essere eseguito a ritroso.

La ricorsione in coda altro non è che una versione computazionalmente migliore della ricorsione base.

Sfrutta il passaggio di parametro del risultato intermedio all'iterazione per now creare i record di attivazione che invece la ricorsione normale alloca (può portare a overflow infatti, quella in coda correttamente NO).

ESECUZIONE PROGRAMMA :

Input :	RIS. IF :	CONSEGUENZA :
(2,4,6)	length ≠ 0	return (concat(2 * 3, (4,6)) →

↓

Input

(4, 6)

RISULTATO IF

length ≠ 0

CONSEGUENZA

return (concat(4 * 3), (6))

(6)

length ≠ 0

return (concat(6 * 3, ()))

()

length = 0

return ()

Torna retro:

return ((18)) → return (concat((12), (18)))

→ return (concat((6), (12, 18)))

→ return finale: (6, 12, 18)

SPIEGAZIONE: ritorna le liste dei numeri passati come argomento dentro ad una lista, moltiplicati *3 e nello stesso ordine in cui sono arrivati.

RICORSIONE IN

Coda:

void main() {

list² res = function((2, 4, 6), ());

}

list² function(list² list, list² res) {

if (length(list) = 0) then

return res;

else

return function(cdr(list), concat(res, ^{3*}car(list)));

}

ESECUZIONE FUNCTION:

Input	IF length ≠ 0	conseg.
((2, 4, 6), (5))	length ≠ 0	return (function((4, 6), concat((5), 3 * 2))
((4, 6), (6))	length ≠ 0	" " " ((6), concat((6), 3 * 4))
((6), (6, 12))	" "	" " " ((6, 12), 3 * 6)
((), (6, 12, 18))	length = 0	return res. <div style="text-align: center;"> } ↓ (6, 12, 18)! </div>

ESERCIZIO 6) La semantica è definita da ~~due~~ stati/passi che "catturano" l'esecuzione di un programma. Uno stato è rappresentato principalmente da :

- AMBIENTE : è una mappatura tra identificatori (σ) (nomi/var) e locazioni di memoria;
- MEMORIA : è una mappatura tra locazioni di memoria e valori effettivi;

1) REGOLA INIZIALE :

$$\frac{J \vdash \langle 3x + y, \sigma \rangle \longrightarrow \langle k_0, \sigma \rangle}{J \vdash \langle y := 3x + y, \sigma \rangle \longrightarrow \langle y := k_0, \sigma \rangle}$$

2) VALUTANO k_0 :

$$\frac{J \vdash \langle 3x, \sigma \rangle \longrightarrow \langle k_1, \sigma \rangle}{J \vdash \langle 3x + y, \sigma \rangle \longrightarrow \langle k_1 + y, \sigma \rangle}$$

3) VALUTIAMO x :

$$\frac{P \vdash \langle x, \sigma \rangle \rightarrow \langle 3\sigma \rangle}{P \vdash \langle 3x, \sigma \rangle \rightarrow \langle 3 \cdot 3, \sigma \rangle}$$

$$f(x) = 3$$

4) VALUTIAMO CALCOLO:

$$f(x) = 3 \quad \text{quindi} \quad 3 \cdot 3 = 9 \Rightarrow K_1 = 9$$

5) TORNIAMO IN DISSERVO: (REGOLA ③)

$$\frac{P \vdash \langle 3x, \sigma \rangle \rightarrow \langle 9, \sigma \rangle}{P \vdash \langle 9 + y, \sigma \rangle \rightarrow \langle 9 + y, \sigma \rangle}$$

6) VALUTIAMO y :

$$f(y) = p_y \rightarrow \sigma(p_y) = 2$$

7) TORNIAMO A REGOLA 2:

$$\frac{P \vdash \langle y, \sigma \rangle \rightarrow \langle 2\sigma \rangle}{P \vdash \langle 9 + y, \sigma \rangle \rightarrow \langle 9 + 2, \sigma \rangle}$$

8) RISULTATO:

$$P \vdash \langle 9 + 2, \sigma \rangle \rightarrow \langle M, \sigma \rangle$$

STATO FINALE:

$$P = \left[x \mapsto 3, y \mapsto b_y \right] \sim \sigma = \left[b_y \mapsto M \right].$$