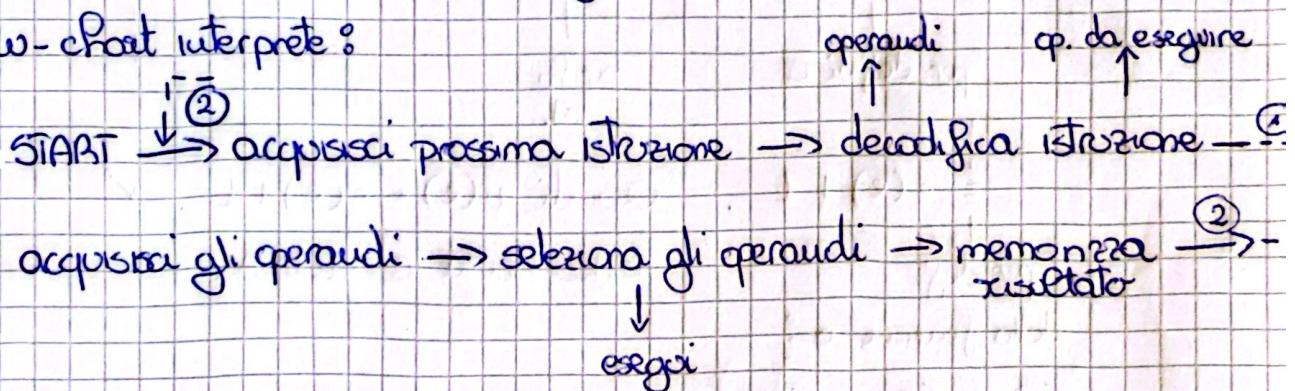


Esercizio 1) Definire intuitivamente e formalmente (mediante definizione semantica) cos'è un interprete. Descrivere la struttura di un interprete e spiegarne il funzionamento. ① scritto in Lo

Intuitivamente, un interprete è un programma che esegue direttamente un altro programma, scritto in un determinato linguaggio sorgente, senza prima tradurlo in codice macchina. Formalmente, un interprete è un programma che ne riceve un altro, scritto in L, insieme ad un input. L'interprete esegue il programma tramite l'input fornito, utilizzando schematicamente le primitive messe a disposizione dalla macchina o struttura di lo, ritornando l'output risultante.

$$I^{LoL} = (\text{Prog}^L \times \text{Data}) \rightarrow \text{Data}$$

Flow-chart interpréte:



Esercizio 2) Si consideri l'insieme E definito induttivamente: $n \in E$,

• 3pt. , se $'e_1, e_2' \in E$ allora $e_1 \wedge e_2 \in E$, $e_1 \vee e_2 \in E$.

Si definiscono sugli elementi di tale insieme le seguenti

für zwei: $v(n) = 1$, $v(e_1 \wedge e_2) = v(e_1 \vee e_2) = v(e_1) + v(e_2)$ & $\sigma(n) = 0$, $\sigma(e_1 \wedge e_2) = \sigma(e_1 \vee e_2) =$

$\circ(e_1) + \circ(e_2) + 1$. Si dimostri per induzione che

$$\forall e \in E. \nu(e) = o(e) + 1.$$

Abbiamo quindi : $v = \begin{cases} v(n) = 1 \\ v(e_1 \wedge e_2) = v(e_1 \vee e_2) = v(e_1) + v(e_2) \end{cases}$

$$o = \begin{cases} o(n) = 0 \\ o(e_1 \wedge e_2) = o(e_1 \vee e_2) = o(e_1) + o(e_2) + 1 \end{cases}$$

CASO BASE: prendiamo $e = n$, abbiamo che $v(e) = 1$ e $\circ(e) = 0$.

Per cui $v(e) = \circ(e) + 1$

$$1 = 0 + 1$$

$$1 = 1 \quad \text{VERIFICATO!}$$

PASSO INDUTTIVO: dobbiamo dimostrare che $\forall e \in E \cdot v(e) = \circ(e) + 1$.

Assumiamo come ipotesi induttiva, con $e = e_1 \wedge e_2$ (analogo per $e_1 \vee e_2$ visto le fazioni), che:

$$1) v(e_1) = \circ(e_1) + 1$$

$$2) v(e_2) = \circ(e_2) + 1$$

Allora per dimostrare che $v(e) = \circ(e) + 1$ costruiamo

$$v(e) = v(e_1) + v(e_2) = \circ(e_1) + 1 + \circ(e_2) + 1 =$$

per def di v

per Pp
ind.

$$= \circ(e) + 1 \quad \text{Quindi } v(e) = \circ(e) + 1 \quad \checkmark$$

per def

DIMOSTRAZIONE COMPLETATA.

della funzione \circ :

$$\circ(e) = \circ(e_1) + \circ(e_2) + 1$$

Esercizio 3) Si consideri il seguente programma. Si dica cosa viene

(Spt.) calcolato dall'assegnamento in caso di scoping statico (mostrando come vengano calcolati i link statici e come vengano risolti i riferimenti non locali) e in caso di scoping dinamico (mostrando l'esecuzione della CRT):

1. $\{ \text{int } a = 2;$

7. $\text{int } c = 5;$

13. $\text{fun2}();$

2. $\text{int } b = 3;$

8. $\text{void } \text{fun3}();$

3. $\text{void } \text{fun1}();$

9. $\text{int } a = c - b;$

4. $\text{int } z = a * b, \}$

10. $\text{fun1();};$

5. $\text{void } \text{fun2}();$

11. $b = a + b;$

6. $\text{int } a = 4;$

12. $\text{fun3();};$

Lo scoping è una tecnica che permette di determinare l'ambito di visibilità di una variabile. Esistono due strategie di scoping, e variano appunto nel

- caso una variabile viene considerata all'interno di un certo contesto di programmazione:

- SCOPING STATICO: si osserva dove le variabili sono definite nel codice, o a tempo di compilazione;
- SCOPING DINAMICO: si osserva la successione di chiamate a funzione nell'ambiente di esecuzione (run-time);

Per calcolare passo passo l'evoluzione di entrambe si utilizzano reciprocamente:

- LINK STATICO (CS): puntatore/collegamento che risale i contesti lessicali, calcolato come

$$K = Sd(\text{chiamante}) - Sd(\text{chiamato}) + 1,$$

dove Sd è la profondità statica e di modifica della funzione in esame. Se $K=0$, il link statico punta direttamente all'indirizzo del chiamante, altrimenti si risale la catena di $K=n$ passi;

- TABELLA CRT: tabella centrale dei riferimenti, che contiene una lista di elementi e informazioni necessarie ad accedere ad un eventuale ambiente di riferimento.

VERSIONE SCOPING STATICO:

- 1) main inizializza 'a' e 'b', RKA 1 e 2 :

a	2
b	3

- 2) main invoca fxz2 , RKA 13, 5, 6, e 7 :

$$K_{fxz2} = Sd(\text{main}) - Sd(\text{fxz2}) + 1 = 0 - 1 + 1 = 0 ;$$

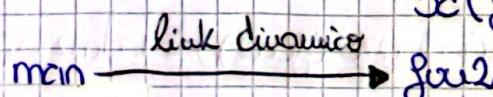
$$CS(\text{fxz2}) = \text{indirizzo}(\text{main})$$

Variabili : - 'a' viene creata localmente in fxz2;

- 'c' viene creata localmente in fxz2;

- 'b' fa riferimento non locale :

$$Sd(\text{fxz2}) - Sd(\text{main}) = 1 - 0 = 1$$



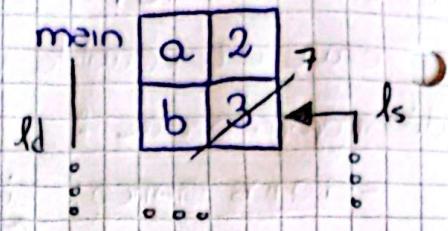
a	b
2	3

cs	a	c
4	5	

3) fun2 esegue operazione $b = a + b$, RICA 11:

$$b = a + b = 4 + 3 = 7$$

↓ ↓
locale non locale



4) fun2 invoca fun3, RICA 12:

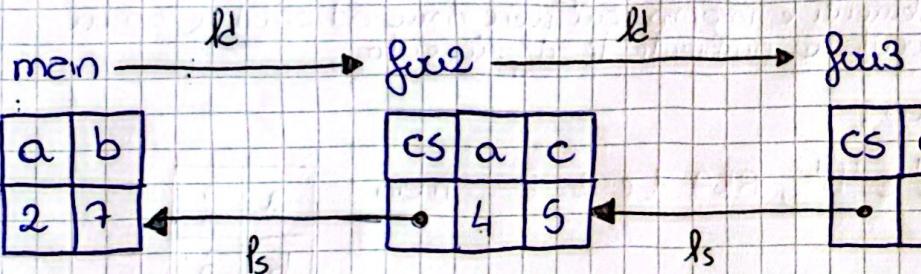
$$K_{fun3} = Sc(fun2) - Sc(fun3) + 1 = 1 - 2 + 1 = 0$$

$$CS(fun3) = \text{indirizzo}(fun2)$$

- Variabili:
- 'a' viene ricevuta localmente;
 - 'b' fa riferimento non locale;
 - 'c' fa riferimento non locale;

$$Nb = Sc(fun3) - Sc(main) = 2 - 0 = 2;$$

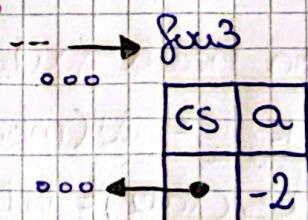
$$Nc = Sc(fun3) - Sc(fun2) = 2 - 1 = 1;$$



5) fun3 esegue operazione $a = c - b$, RICA 9:

$$a = c - b = 5 - 7 = -2$$

↓ ↓
locale non locale,
da main
non locale,
da fun2



6) fun3 invoca fun, RICA 10:

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

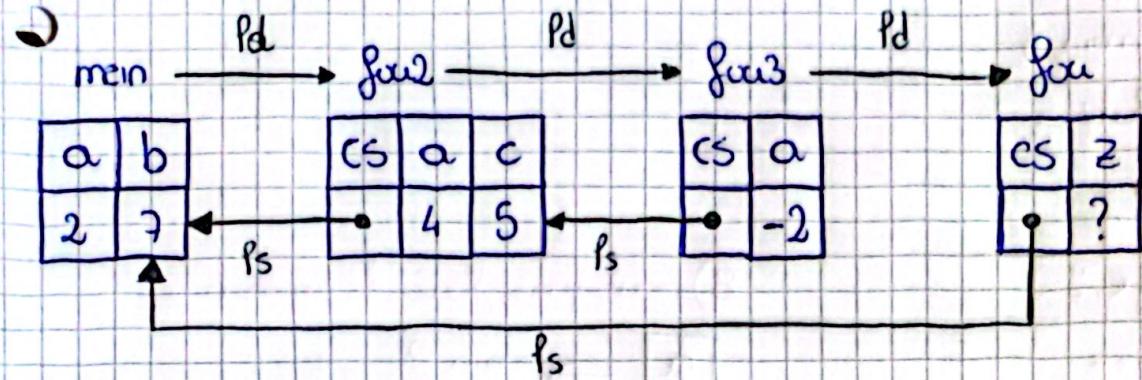
$$CS(fun) = CS(CS(fun3)) = CS(\text{indirizzo}(fun2)) = \text{indirizzo}(main)$$

- Variabili:
- '2' è locale;

- 'a' non è locale;
- 'h' non è locale;

$$Na = Sd(gvu) - Sd(men) = 1 - 0 = 1,$$

$$Nb = Sd(gm) - Sd(mn) = 1-0=1;$$



7) fai eseguire operazione $z = a^* b$, RIGA 4:

$$z = a * b = 2 * 7 = \overline{14}$$

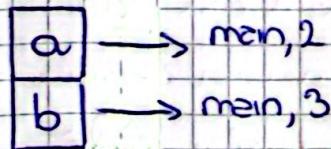
↓ ↓ ↓
 lade von lade,
 da mein
 non lade,
 da mein



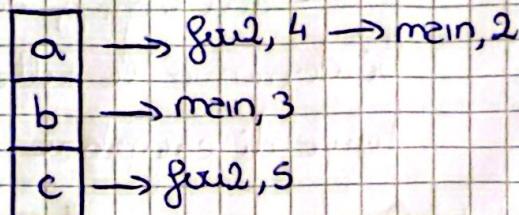
8) Termina tutti a catena;

VERSIONE SCOPING DINAMICO:

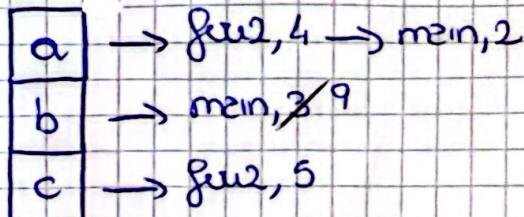
1) mein નિર્ણયાદીનાં :



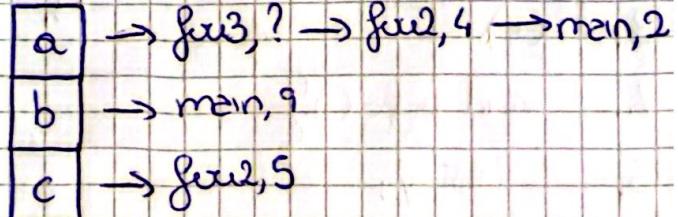
2) man invoca fürz?



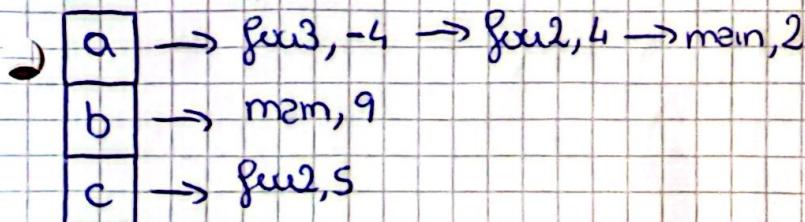
3) Für welche:



4) $\lim_{x \rightarrow 0} \ln(x)$ ist ca. $\ln(3)$:



5) fülv elege:



6) fun3 inca fun:

a	→ fun3, -4 → fun2, 4 → main, 2
b	→ main, 9
c	→ fun2, 5
z	→ X

7) fun esegue:

a	→ ...
...	
...	
z	→ fun, -36

8) fun, fun3, fun terminano:

a	→ main, 2
b	→ main, 9
c	X
z	X

Esercizio 4) Si consideri lo schema del seguente codice, nei quali ci sono buchi indicati rispettivamente con (*) e (**).

6pt. Si inserisca opportuno codice nei buchi in modo che:

- con scopo statico, l'assegnamento a X è sempre lo stesso, ad ogni iterazione;
- con scopo dinamico, la X è diversa all'iterazione;

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

1. { int i = 0;
2. int x = 3;
3. (*)
4. void pippo() {
5. int y;
6. (**)
7. x = pluto();
8. { i = i + 2;
9. }
10. while (i <= 2) {

Nuovo codice:

```
1. { int i = 0;
2.   int x = 3;
3.   int y = i;
4.   int pluto() {
5.     return y;
6.   }
7.   void pippo() {
8.     int y;
9.     y = i;
10.    x = pluto();
11.    i = i + 2;
12.  }
13.  while (i <= 2) {
14.    pippo();
15.  }
16. }
```

Spiegazione scoping statico:

La 'y' globale viene posta ' $=i$ ' (RICA 3), ed essendo codice eseguito una volta sola, non cambierà il valore di 'y' anche al variazione di 'i'. Nell'esecuzione di pippo, durante entrambe le chiamate a pluto, viene quindi ritornato il valore di 'y' che, dato lo stesso sd di pippo e pluto, fa riferimento alla variabile globale per entrambi i casi di iterazione. Viene quindi ritornato '0' entrambe le esecuzioni di pluto ($K_{pluto} = Sd(pippo) - Sd(pluto) + 1 = 1 - 1 + 1 = 1$)

quindi $CS(pluto) = CS(pippo) = \underline{\text{indirizzo(main)}}$),

Spiegazione scoping dinamico:

Per lo scoping dinamico invece, la var. 'y' a cui fa riferimento pluto nel ritorno del valore, è l'ultima (o + recente) presente nella CRT, ossia quella di pippo, impostata ' $=i$ ' a RICA 9. A differenza dell'assegnamento di prima tuttavia, questo viene ripetuto 2 volte ($\text{while}(i \leq 2) = 2 \text{ iter}$); nella 1^a ha valore $i = 0$, per tanto anche l'"y" ritornata da pluto, nella 2^a ha valore $i = 2$. L'assegnamento a 'x' è quindi differente per entrambe le iterazioni.

ESEMPIO 5) Definire brevemente il concetto di ricorsione, standard e in coda.

(sp)

Si consideri il programma nella destra e si descriva che cosa calcola mostrando anche il cicalo sulla lista $(2, 4, 6)$. Trasformare quindi il programma in ricorsione in coda, e mostrare il funzionamento fornito sulla lista $(2, 4, 6)$.

Nota: $\text{length}(\text{list}) \rightarrow$ lunghezza lista,
 $\text{car}(\text{list}) \rightarrow$ primo elemento
 $\text{cdr}(\text{list}) \rightarrow$ tutta la lista fatta il 1^{o} elemento
 $\text{caucat} \rightarrow$ concatenare due liste;

Pista $\text{function}(\text{list}) \{$

if ($\text{length}(\text{list}) == 0$) then

return();

else

{ return caucat(car(list) * 3, function(cdr(list))); }

Una funzione è ricorsiva quando chiama se stessa per risolvere una versione più piccola dello stesso problema a cui è sottoposta. Ogni chiamata si avvicina quindi ad un caso base, definito da condizione di terminazione.

Ogni chiamata crea un nuovo record di attivazione, per cui le chiamate si "impilano", e il cicalo avviene a ritroso, una volta raggiunto il caso base.

Nella ricorsione in coda invece, non ci sono operazioni in sospeso al momento della chiamata ricorsiva. Il cicalo avviene quindi "in avanti", accumulando il risultato nei parametri passati, trasformando la ricorsione in un ciclo while / for in termini di efficienza.

ESEMPI DI PROGRAMMA: l'ordine è: INPUT - RISULTATO IF - CONSEGUENZA;

I1) $(2, 4, 6) \rightarrow \text{length} \neq 0 \rightarrow \text{caucat}(2 * 3, \text{function}((4, 6)))$;

I2) $(4, 6) \rightarrow \text{length} \neq 0 \rightarrow \text{caucat}(4 * 3, \text{function}((6)))$;

I3) $(6) \rightarrow \text{length} \neq 0 \rightarrow \text{caucat}(6 * 3, \text{function}(()))$;

I4) $() \rightarrow \text{length} = 0 \rightarrow \text{return}();$

Torna I3) caucat((18), ()); $\rightarrow \text{return}(18);$

trasformata in lista da coda

TORNA A IT2) concat((12), (18)); \rightarrow return (12, 18);

TORNA A IT1) concat((6), (12, 18)); \rightarrow return (6, 12, 18);

- Intuitivamente, prende ogni elemento della lista e lo moltiplica $\times 3$, mantenendone l'ordine.

PROGRAMMA CON RICORSIONE IN COPIA:

vor main() {

list nsotto = function((2, 4, 6), ());
}

lista function (list list, list res) {

if (length(list) = 0)

return res;

else

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

}

ESECUZIONE:

IT1) ((2, 4, 6), ()) \rightarrow length(list) $\neq 0 \rightarrow$ return ((4, 6), concat((), 2 * 3)),

IT2) ((4, 6), (6)) \rightarrow length(list) $\neq 0 \rightarrow$ return ((6), concat((6), $\underbrace{4 * 3}_{(12)})$),

IT3) ((6), (6, 12)) \rightarrow length(list) $\neq 0 \rightarrow$ return ((), concat((6, 12), $\underbrace{6 * 3}_{(18)})$),

IT4) ((), (6, 12, 18)) \rightarrow length(list) = 0 \rightarrow return ((6, 12, 18)).

Esercizio 6) Siano $\rho = [x \mapsto 3, y \mapsto p_y]$ un ambiente dinamico con
5pt. la locazione di tipo int, e $\sigma = [p_y \mapsto 2]$ una memoria.

Descrivere le derivazioni della semantica dinamica per il
caso quando $y := 3x + y$ nell'ambiente ρ a partire dalla
memoria σ ($\rho \vdash y := 3x + y, \sigma \rightarrow \dots$).

Per la teoria, desidero ambiente dinamico, memoria e scrittura per assegnamento e valore.

ESEMPIO:

1) REGOLA INIZIALE:

$$\frac{\rho \vdash \langle (3x+y, \sigma) \rangle \rightarrow^* \langle (K_0, \sigma) \rangle}{\rho \vdash \langle y := 3x + y, \sigma \rangle \rightarrow \langle y := K_0, \sigma \rangle}$$

2) VALUTANO L'ESPRESSIONE $3x+y$:

$$\frac{\rho \vdash \langle (3x), \sigma \rangle \rightarrow^* \langle K_1, \sigma \rangle}{\rho \vdash \langle 3x+y, \sigma \rangle \rightarrow \langle K_1+y, \sigma \rangle}$$

3) VALUTANO L'ESPRESSIONE $3x$:

$$\frac{\rho \vdash \langle 3(x), \sigma \rangle \rightarrow \langle 3, \sigma \rangle}{\rho \vdash \langle 3x, \sigma \rangle \rightarrow \langle 3^* 3, \sigma \rangle}$$

4) VALUTANO E CALCOLANO $3^* 3$:

$$\rho \vdash \langle 3^* 3, \sigma \rangle \rightarrow \langle 9, \sigma \rangle$$

quindi

$$\rho \vdash \langle 3x, \sigma \rangle \rightarrow^* \langle 9, \sigma \rangle \Rightarrow K_1 = 9$$

5) TORNANO INDIETRO: valutiamo K_1+y

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

$$\rho(y) = p_y \sim \sigma(p_y) = 2$$

6) TORNANO INDIETRO: valutiamo $9+2$

$$\rho \vdash \langle 9+2, \sigma \rangle \rightarrow \langle \underbrace{11}_{K_0}, \sigma \rangle \implies$$

$$\rho \mapsto \langle 3x+y, \sigma \rangle \rightarrow^* \langle u, \sigma \rangle$$

$$\rho \mapsto \langle y := (3x+y), \sigma \rangle \rightarrow \langle y = u, \sigma \rangle$$

7) ESAMINIAMO $y = u$:

$$\rho \mapsto \langle y = u, \sigma \rangle \rightarrow \sigma = [ly \mapsto u], \rho(y) = p_u.$$

-