giovedì 19 settembre 2019 16:14

$$MCD : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

1) $MCD(n, n) = n$
2) $MCD(n, m) = MCD(n-m, m)$ se $n > m$
3) $MCD(n, m) = MCD(n, m-n)$ se $m > n$

$MCD(21, 6)$
$= \{$ proprietà 2$\}$
$MCD(15, 6)$
$= \{$ prop. 2$\}$
$MCD(9, 6)$
$= \{$ prop. 2$\}$
$MCD(3, 6)$
$= \{$ prop. 3$\}$
$MCD(3, 3)$
$= \{$ prop. 1$\}$
3

```
int x = 21;
int y = 6;
while (x != y)
    if (x > y) x = x - y;
        else y = y - x;
```

ho il risultato come valore di x e y

inclusioni di librerie (programmi predefiniti)

standard input/output

```
#include <stdio.h>
Dichiarazioni
main()
{
    Blocco
}
```

{ comando1 comando2 .... comandon }

Un unico comando

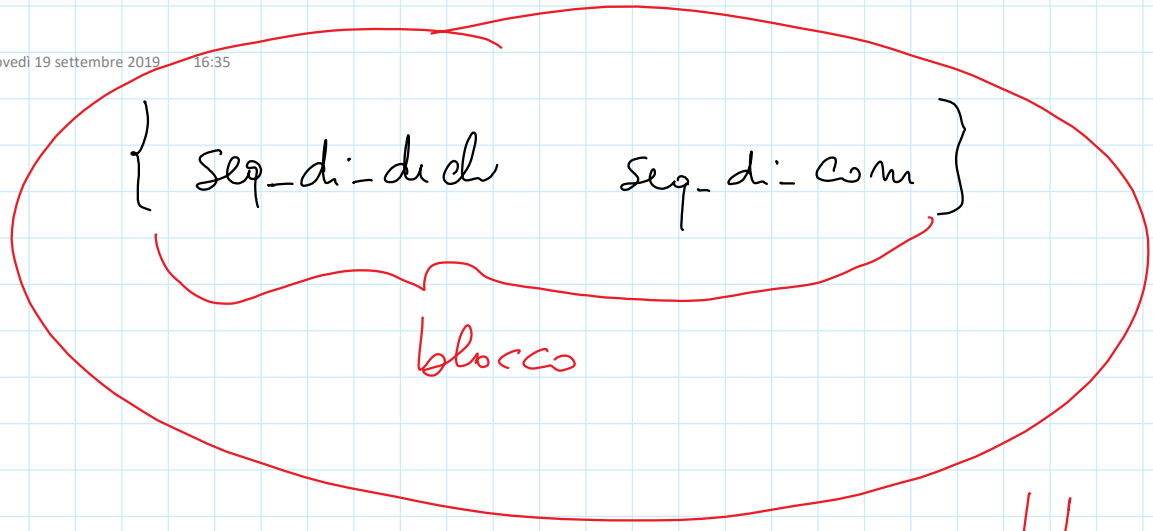
{ sequenza_di_dichiarazioni sequenza_di_comandi }

blocco

```
int numerodielementi = 10;      | corretta
int NumeroDiElement = 10;
int Numero Di Elementi=10;      | non è corretta
```

3 nomi

```
int numero_di_element =10;
```



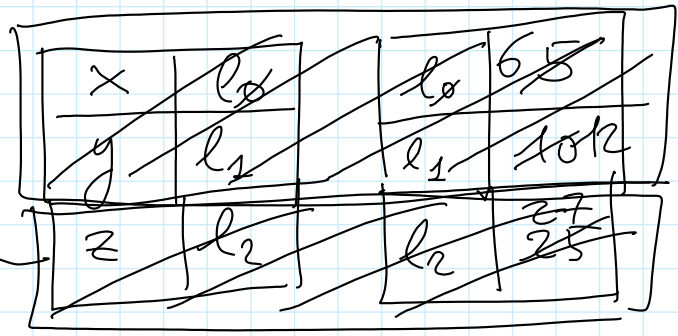
è un comando!!

Tutte le variabili create durante l'esecuzione di un blocco alla fine della sua esecuzione (del blocco) vengono cancellate (sia dall'ambiente che dalla memoria)

giovedì 19 settembre 2019 16:38

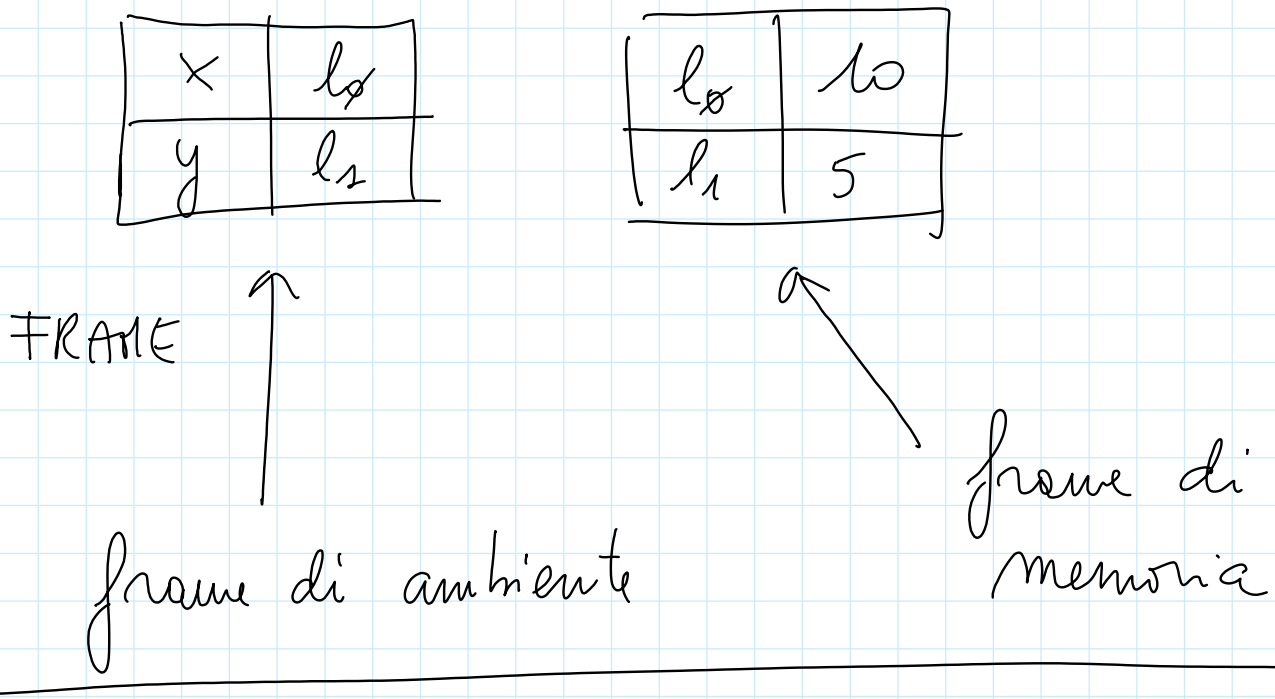
ambienti memoria

```
{ int x = 5;
  int y = 10;
  x = x + 1;
  { int z = 25;
    z = z + 2;
  }
  y = y + 2;
}
```



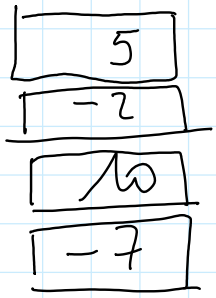
errore

```
y = y + z;
```



PILA (STACK)

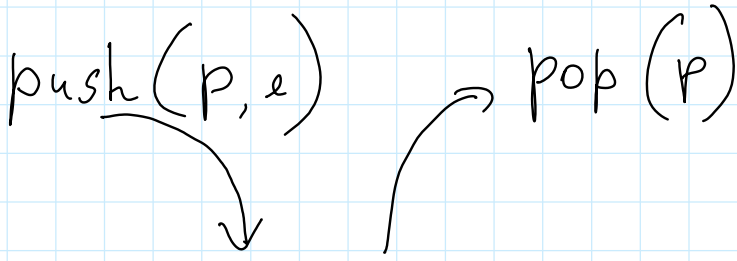
Struttura dati (collezione di valori con una loro gestione)



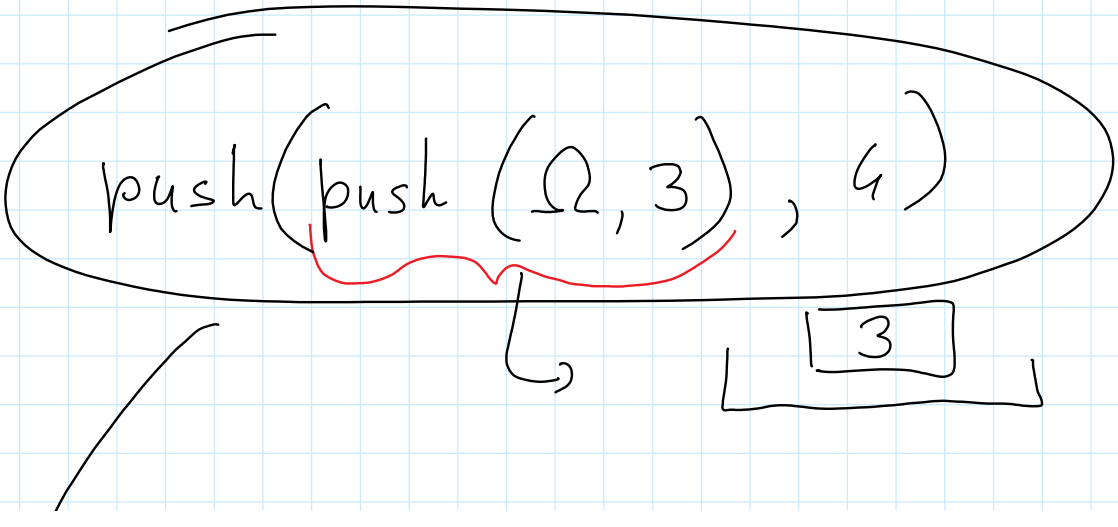
questa collezione di valori si chiama PILA se è gestita in modo

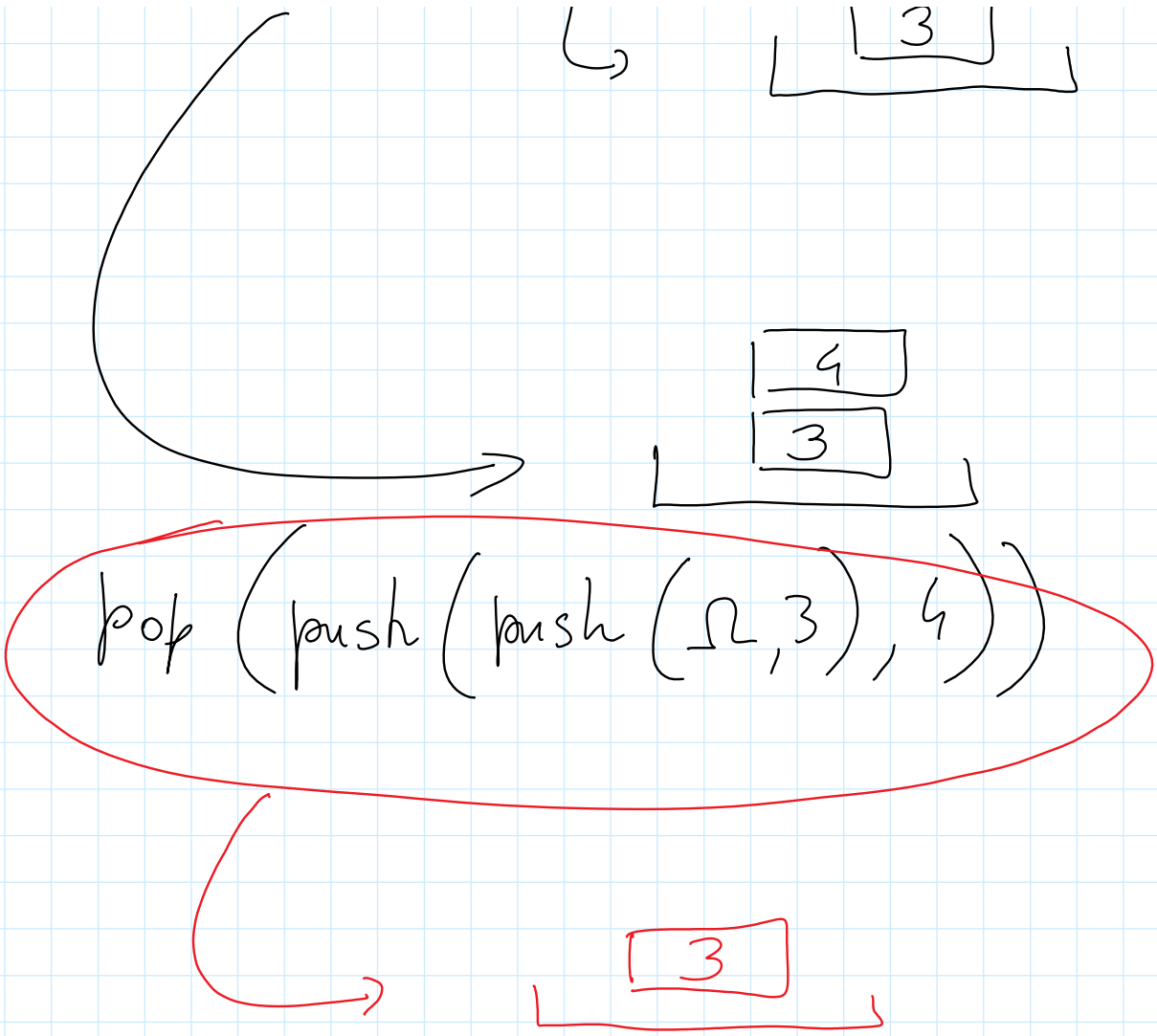
Last In First Out

(LIFO)



pile vuote

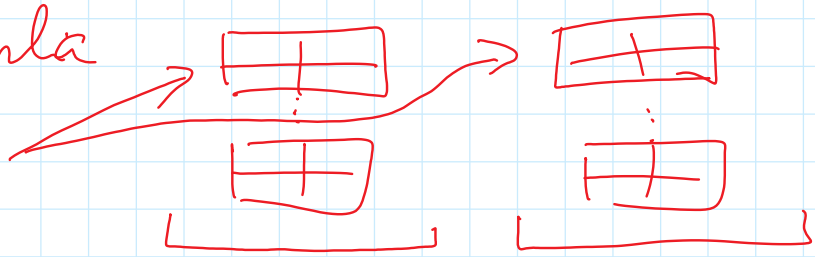



3
4
3
pop (push (push (Ω, 3), 4))
3

Lo stato è rappresentato da una

PILA di FRAME

- Quando si inizia l'esecuzione di un blocco si aggiunge un frame vuoto sulla pila ambiente e sulla pila memoria (frame vuoto ——)
- Ogni dichiarazione crea una associazione sui frame di ambiente e di memoria che stanno in testa alle pile

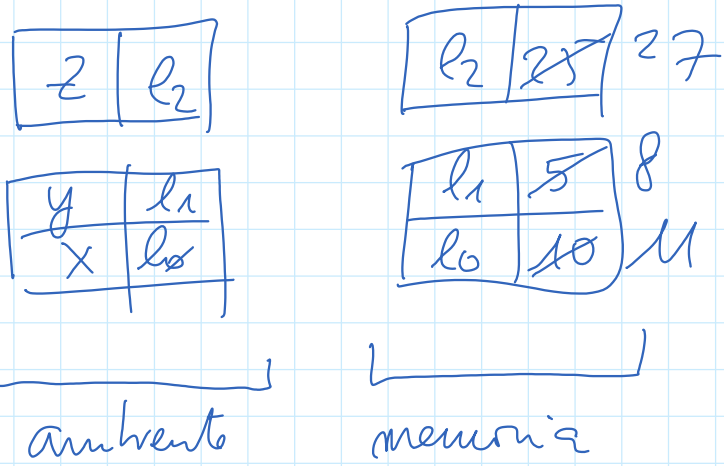


- Tutte le volte che si esce da un blocco (ovvero si esegue }) si fa un "pop" sulle pile di ambiente e sulle pile di memoria.

```
{ int x = 10;
  int y = 5;
  x = x + 1;
  { int z = 25;
    y = y + 3;
    z = z + 2;
  }
  y = y + 2;
}
```

variabili globali

" locali



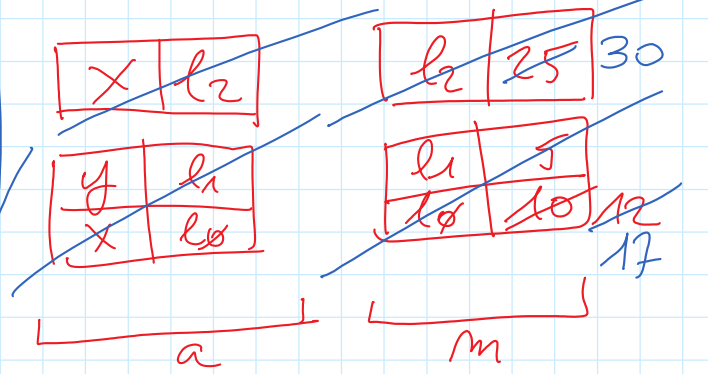
Visibilità delle variabili (nomi, identificatori)

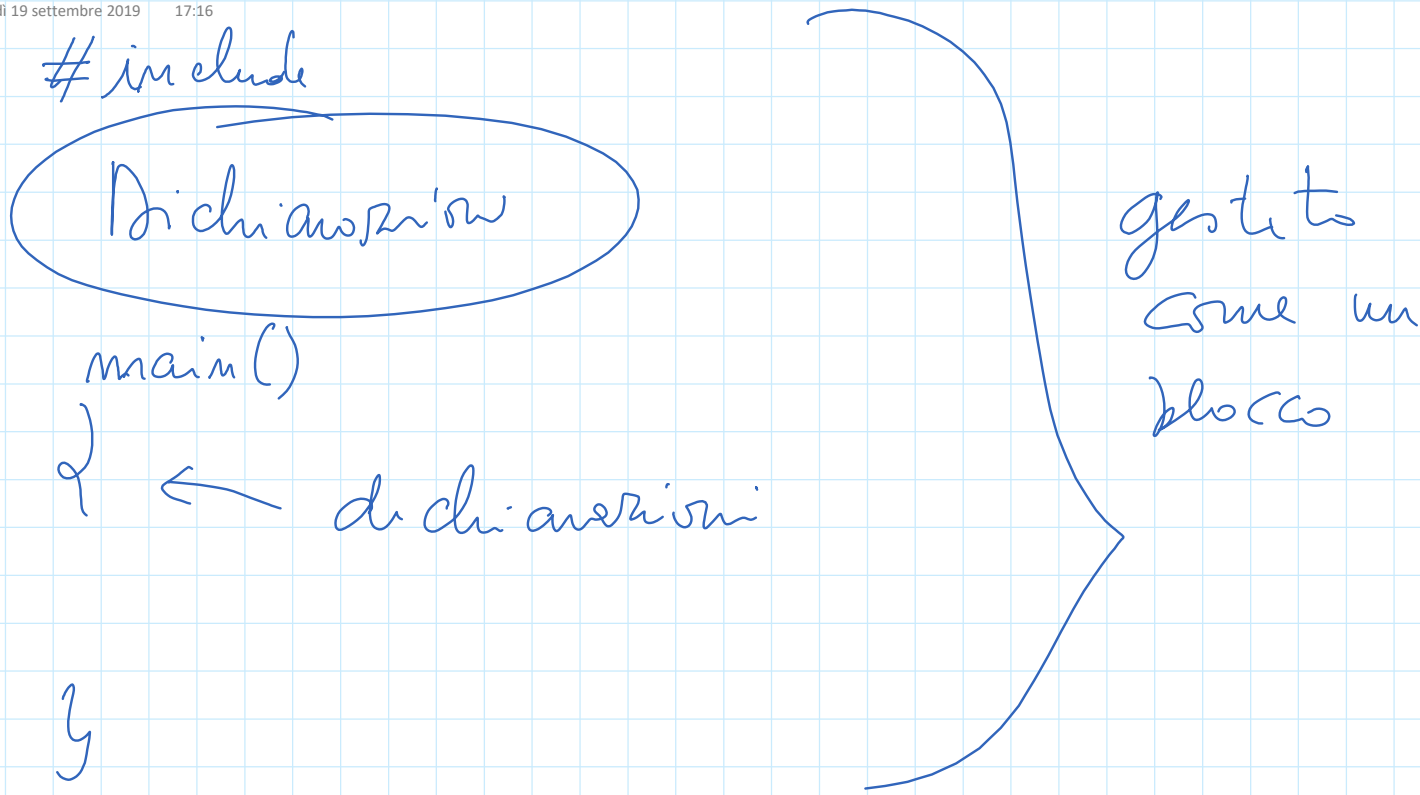
Ogni variabile dichiarata in un blocco è VISIBILE in tutti i blocchi interni, a meno che non venga ridichiarata in un blocco interno!

```
{ int x = 10;
  int y = 5;
  x = x + 2
  { int x = 25;
    x = x + y;
  }
  x = x + y;
}
```

int x = 25; ← errore

Corretto




#include
Dichiarazioni
main()
{
dichiarazioni
}
gestito come un blocco

```
#include <stdio.h>

int x;
int y;

main()
{
  scanf("%d,%d", &x, &y);
  while (x != y)
      if (x > y) x = x - y;
      else y = y - x;
  printf("il valore mcd e' %d", x);
}
```

devi leggere un valore intero

```
#include <stdio.h>
int x;
int y;
int z;
int w;

main()
    scanf("%d,%d", &x, &y);
    scanf("%d,%d", &z, &w);
    while (x != y)
    ...
    while (z != w)
    ...
```

2 volte lo stesso programma ma variabili diverse

giovedì 19 settembre 2019 17:28

Nei linguaggi di programmazione imperativi esiste la possibilità di scrivere

FUNZIONI (pezzi di programma parametrici che, in più, restituiscono un valore)

e

PROCEDURE (pezzi di programma parametrici)

```
# include <stdio.h>

int MCD ( int n, int m )
{ while (n != m)
    if (n > m) n = n - m;
    else       m = m - n;
  return n;
}
```

il tipo e il nome dei parametri

tipo del risultato
nome della funzione

21 = 21 - 3

non ha significato

```
int x;
int y;
int z;
int w;

main()
{ scanf("%d, %d", &x, &y);
  //    "      //  , &z, &w);

  printf("%d", MCD(x, y));
  printf("%d", MCD(z, w));
```