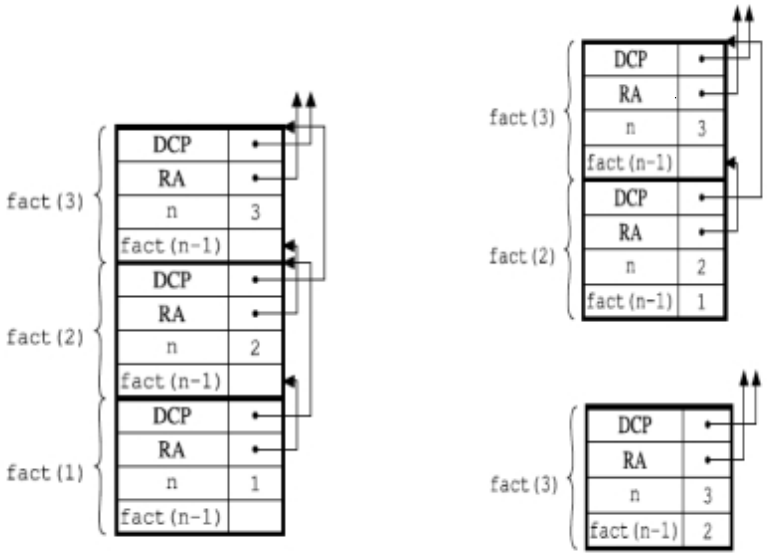
Costrutti non strutturati

- Costrutto Strutturato: un costrutto non atomico che definisce una sequenza atomica "chiusa", ovvero ha ha un unico punto di ingresso (il primo atomo) ed un unico punto di uscita (l'ultimo atomo).
- Linguaggio/Programma Strutturato: Le funzioni calcolabili possono essere espresse con programmi che utilizzano solo costrutti strutturati (Bohm-Jacopini, 1966)
- Rispondiamo:
  - Quali dei seguenti costrutti sono strutturati e perchè?
    (a) Goto; (b) break; (c) continue; (d) return
  - Il costrutto Switch del C è un costrutto strutturato?

Ricorsione

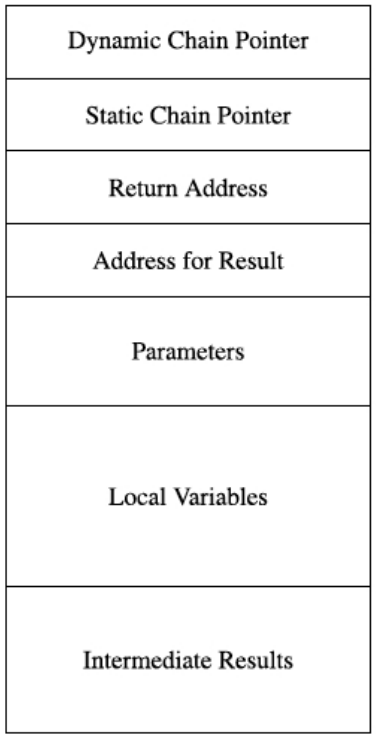
- Algoritmi definiti in modo induttivo
- Introdotta attraverso Procedure/Funzioni
- Possono richiedere l'allocazione di una grande quantità di AR

```
int fact(int n){
        if (n<0) return 0;
         else if (n==0) return 1;
             else return n * fact(n − 1);
 }
```



- Dire chi sono i campi: DCP, RA presenti negli AR e di quale campo fa parte l'entry "fact(n-1)..."

AR per Memoria a Stack



- Ricordiamo la struttura generale vista

Ricorsione: Funzioni Parziali e Gestione delle Eccezioni

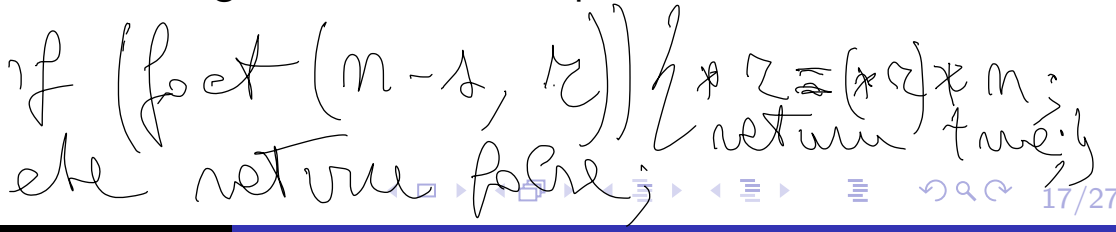
- Possiamo ridurre l'allocazione di AR al solo AR dell'invocazione corrente
- Ricorsione di coda: È una tecnica che lo permette
- Prima però commentiamo la definizione data per `fact`.

```
int fact(int n){** fact calcola 0 se n è negativo, altrimenti n! **
        if (n<0) return 0;
          else if (n==0) return 1;
            else return n * fact(n - 1);
    }
```

  - C non ha un meccanismo di eccezioni che sarebbe stato preferibile a questo uso arbitrario (e oscuro) del valore calcolato 0

```
bool fact(int n, int  * r){** Convenzione in C per le parziali**
        if (n<0) return false;
          else if (n==0){*r = 1; return true; }
              else{*r = n * fact(n - 1); return true; }
    }
```

  - Il commento inserito è d'obbligo ma non risolve il problema



Ricorsione: Funzioni Parziali e Gestione delle Eccezioni - 2

- Il commento inserito è d'obbligo ma non risolve il problema delle parziali

```
int fact(int n){** fact calcola 0 se n è negativo, altrimenti n! **
        if (n<0) return 0;
          else if (n==0) return 1;
            else return n * fact(n - 1);
    }

bool fact(int n, int  * r){** Convenzione in C per le parziali**
       if (n<0) return false;
         else if (n==0){*r = 1; return true; }
              else{*r = n * fact(n - 1); return true; }
  }
```

- la definizione sotto è una corretta definizione della funzione **parziale** n!.

```
int fact(int n){** fact calcola 0 se n è negativo, altrimenti n! **
        if (n==0) return 1;
        else return n * fact(n - 1);
    }
```

  - Noi vogliamo usare proprio quella parziale nei nostri programmi?
  - Nella quasi totalità dei casi NO
  - Esprimere una definizione che ci dica quando le cose non vanno:
    senza "falsare" il valore calcolato
    ovvero, faccia uso di un *meccanismo di eccezioni*.

Ricorsione: Ricorsione di Coda

- Possiamo ridurre l'allocazione di AR al solo AR dell'invocazione corrente
- Ricorsione di coda: È una tecnica che lo permette

```
int fact(int n){** fact calcola 0 se n è negativo, altrimenti n! **
        if (n<0) return 0;
          else if (n==0) return 1;
             else return n * fact(n - 1);
  }
```

Definition (Invocazione di Coda e Ricorsione di coda)

Sia `F` una definizione di funzione che contiene un'invocazione di una funzione `g`. Tale invocazione è detta invocazione di coda se, quando applicata, `F` restituisce il valore calcolato da tale invocazione senza ulteriore calcolo. Una definizione ricorsiva `F` è con ricorsione di coda se ogni sua invocazione contenuta in `F` é una invocazione di coda.

- la definizione di `fact` non è con ricorsione di coda
- questa sotto lo è

```
int factT(int n, int acc){** acc produttoria argomenti n di invocazioni precedenti **
          if (n<0) return 0;
            else if (n==0) return acc;
               else return fact(n - 1, n * acc);
   }
```

- Per ogni intero `n`, `factT(n,1)` calcola `n!`.

Ricorsione: Macchine Astratte per Ricorsione di Coda

AR nell'invocazione di funzione con Ricorsione di Coda.

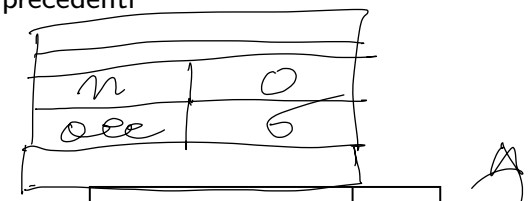
- Sia $g(e_0)$ l'invocazione di una tale funzione, con e_0 (lista degli) argomenti dell'applicazione.
- L'invocazione può condurre a due casi possibili:
  - Calcola un valore senza richiedere ulteriori invocazioni di g. Caso finale
  - Conduce ad un'invocazione di coda $g(e_1)$. Allora $g(e_1)$ è tutto ciò che serve per calcolare e_0
    - Possiamo rimpiazzare l'invocazione $g(e_0)$ con $g(e_1)$, ovvero
    - Possiamo, rimpiazzare il contenuto dell'AR prodotto per calcolare $g(e_0)$ con quello per $g(e_1)$

Ricorsione di Coda: Activation Record di factT

- Sia $g(e_0)$ l'invocazione di ... con e_0 (lista degli) argomenti dell'applicazione.
- L'invocazione può condurre a due casi possibili:
  - Calcola un valore senza richiedere ulteriori invocazioni di g. Caso finale
  - Conduce ad un'invocazione di coda $g(e_1)$. Allora $g(e_1)$ è tutto ciò che serve per calcolare e_0
    - Possiamo rimpiazzare l'invocazione $g(e_0)$ con $g(e_1)$, ovvero
    - Possiamo, rimpiazzare il contenuto dell'AR prodotto per calcolare $g(e_0)$ con quello per $g(e_1)$

Applichiamolo a factT(3,1), definita sotto

```
int factT(int n, int acc){** acc produttoria argomenti n di invocazioni precedenti **
        if (n<0) return 0;
          else if (n==0) return acc;
              else return fact(n − 1, n ∗ acc);
}
```



...	...
CD	...
RA	...
n	3
acc	1
fact(n-1,n*acc)	

...	...
CD	...
RA	...
n	2
acc	3
fact(n-1,n*acc)	

...	...
CD	...
RA	...
n	1
acc	6
fact(n-1,n*acc)	

Ricorsione: Memoization e Memoria Statica nelle Procedure

- Possiamo ridurre l'allocazione di AR al solo AR dell'invocazione corrente
- Ricorsione Memoized: È una tecnica che lo permette *parzialmente*

Definition (Funzione Memoized)

Una definizione di funzione è Memoized se memorizza tutte le coppie (invocazione, valore-calcolato) durante l'intera esecuzione del programma, fornendo il valore calcolato memorizzato se già invocata sull'argomento, calcolando il nuovo valore altrimenti.

```
int fact(int n){/* diverge quando n<0, altrimenti n! */
    if (n==0) return 1;
    return n * fact(n-1);
}
int memoizedFact(int n){/* diverge quando n<0, altrimenti n! */
                        /* definizione memoized */
    static int Tab[100];
    static int index=0;
    Tab[0]=1;
    if ((n<=index)&!(n<0)) return Tab[n];
    Tab[n] = n * memoizedFact(n-1);
    return Tab[index=n];
}
```

- la definizione di `memoizedFact` è ~~con~~ Memoized
- la definizione utilizza Memoria Statica nell'Activation Record di `memoizedFact`

Ricorsione: Ricorsione di Coda, Memoization

- Due tecniche completamente diverse
- Ricorsione di Coda richiede:
  - Una definizione con ricorsione di coda (fornito da utente);
  - Un meccanismo per riuso dell'AR (fornito dal Linguaggio);
  - Presente nei linguaggi tipo Scheme, Miranda (tutti funzionali);
- Memoization richiede:
  - Un meccanismo per ricordare le invocazioni (Tabella hash);
  - C non lo possiede e noi lo abbiamo emulato nell'esempio;
  - La nostra emulazione ha usato Memoria Statica,
  - Ma potevamo usare anche Memoria Dinamica,
  - ed anche Permanente (un file del File System).
  - Presente in Common Lisp, Perl, Python (manipolazione simbolica)

- Concludiamo:
  - Ricorsione di coda applicabile solo a funzioni definite ricorsive e utilizzabile solo in Linguaggi dotati di meccanismo per il riuso dell'AR.
  - Memoization utilizzabile in tutti i linguaggi e applicabile a tutte le funzioni, richiede che non siano presenti effetti laterali.