CORRISPONDENZA C++ ASSEMBLER

Funzioni C++ (1)

- **Funzione C/C++:**
  - **può avere parametri formali;**
  - **può avere variabili locali;**
  - **può restituire un risultato.**
- **Parametri formali e variabili locali**
  - **detti entrambi *entità locali*, che hanno lo stesso tempo di vita, legato ad una esecuzione della funzione (*istanza*), che avviene in seguito a una chiamata, e non alla funzione stessa.**
- **Funzione:**
  - **può dar luogo a diverse istanze;**
  - **all'inizio di ogni istanza nasce una copia delle entità locali, e alla fine tale copia muore;**
  - **le varie copie delle entità locali, pur essendo diverse da istanza a istanza, hanno lo stesso identificatore C++ per tutte le istanze;**
  - **il tempo di vita di una copia delle entità locali (dalla nascita alla morte) è uguale per tutte le variabili locali definite nei vari blocchi di una medesima funzione.**

Funzioni C++ (2)

- **Funzioni ricorsive (direttamente o mutuamente):**
  - **una nuova istanza comporta all'inizio la nascita di una nuova copia delle entità locali, senza che avvenga la morte della copia relativa all'istanza precedente, non ancora terminata;**
  - **le morti della copia delle entità locali dell'istanza precedente avvengono alla fine di questa, dopo che è morta la copia delle entità locali della nuova istanza (muoiono per prime le copie delle entità locali nate per ultime);**
  - **le copie delle entità locali devono quindi utilizzare locazioni di memoria diverse da istanza a istanza, e utilizzare la regola della pila (LIFO).**
- **Chiamata di una funzione:**
  - **vengono specificati i cosiddetti parametri attuali, che sono grandezze utilizzate per inizializzare la copia dei parametri formali.**
- **Una funzione può restituire un risultato, oppure non restituirne ed essere quindi di tipo *void*.**

Tempo di vita delle variabili C++ (1)

- **Variabili globali (definite all'esterno di tutte le funzioni):**
  - **hanno lo stesso tempo di vita del programma (*variabili statiche*).**
- **Variabili locali di una funzione (definite all'interno di una funzione):**
  - **la copia relativa a un'istanza ha un tempo di vita che coincide con il tempo di un'esecuzione della funzione (*variabili automatiche*):**
    - **come detto, le variabili locali definite nei vari blocchi di una medesima funzione hanno tutte lo *stesso tempo di vita*.**
- **Entità locali di una funzione:**
  - **I parametri formali hanno lo stesso tempo di vita delle variabili locali;**
  - **la copia dei parametri formali relativa a un'istanza, quando nasce, viene inizializzata con i parametri attuali, diversi da istanza a istanza;**
- **Variabili create e distrutte dinamicamente:**
  - **non vengono prese in considerazione.**

Livello dinamico di un'istanza

- **In un programma, una funzione può essere eseguita una o più volte (può dar luogo a diverse *istanze*).**
- **Istanze:**
  - **si riferiscono a tutte le funzioni**
- **Livello dinamico di una istanza:**
  - **numero di istanze non ancora terminate (di quella funzione o di altre funzioni), a partire dalla prima ed unica istanza della funzione *main()*, alla quale si associa il livello dinamico 0.**
- **Istanze (di una stessa funzione o di funzioni differenti):**
  - **una istanza inizia (necessariamente) dopo l'istanza che la manda in esecuzione e termina prima di quest'ultima;**
  - **le istanze vanno in esecuzione con la stessa regola LIFO vaista per il tempo di vita della copia delle entità locali.**

Ambiente di un'istanza

- **Ambiente di una istanza:**
  - **si compone di due parti:**
    - ***ambiente globale:***
      - **insieme delle funzioni (compresa se stessa) e insieme delle variabili globali;**
    - ***ambiente locale:***
      - **copia inizializzata dei parametri formali e copia delle variabili locali.**
- **Ambiente globale:**
  - **uguale per tutte le istanze di funzione;**
  - **ambiente *statico* (determinato dal compilatore).**
- **Ambiente locale di una funzione:**
  - **varia da istanza a istanza di una stessa funzione;**
  - **ambiente *dinamico* (creato in esecuzione).**
- **Forma dell'ambiente locale di una funzione:**
  - **uguale il numero delle entità locali e uguale il tipo di ognuna di esse;**
  - **forma *statica (*determinata dal compilatore).**
- **Chiamata di una funzione:**
  - **nasce una copia dell'ambiente locale (la cui forma è statica): i parametri formali vengono inizializzati con i parametri attuali e le variabili locali con le eventuali quantità indicate nella forma dell'ambiente, copia che poi muore quando l'esecuzione della funzione termina.**

Entità statiche ed entità dinamiche

- **Entità statiche:**
  - **ambiente globale;**
  - **forma dell'ambiente locale;**

  vengono definite in fase di traduzione
- **Entità dinamiche:**
  - **ambiente locale.**

  vengono definite in fase di esecuzione.
- **Regole di visibilità:**
  - **possono imporre delle restrizioni alla possibilità di indirizzare le variabili locali:**
    - **in un blocco di una funzione, possono essere indirizzate solo un sottoinsieme delle variabili locali della funzione stessa;**
  - **agiscono in base alla natura testuale del programma;**
  - **sono regole *statiche;***

  hanno effetto in fase di traduzione.

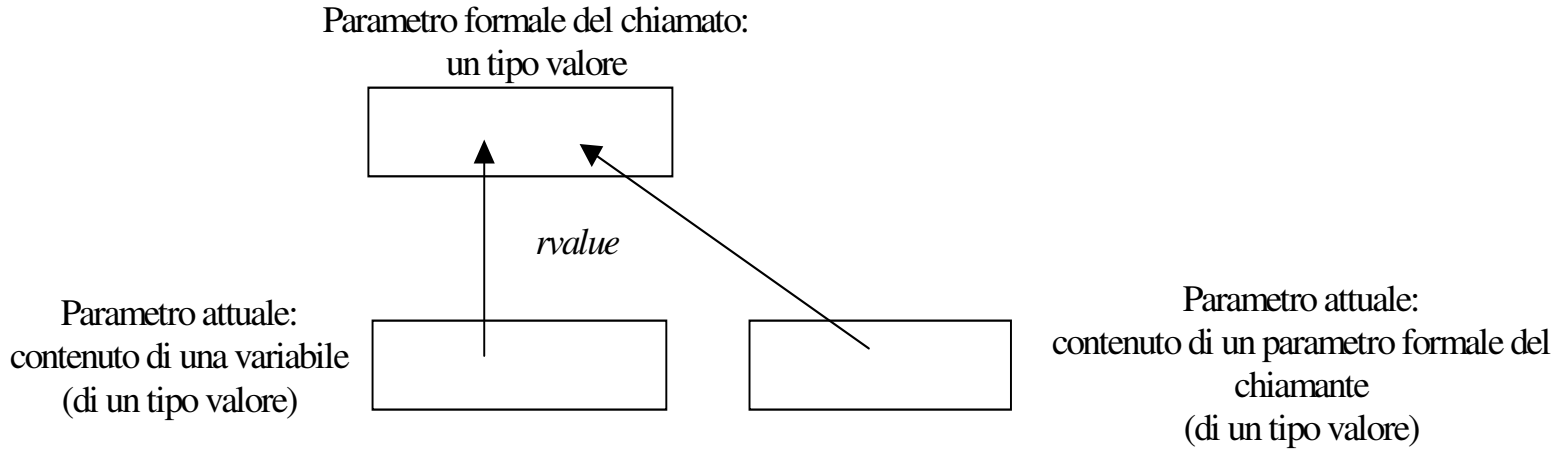
Tipi dei parametri

I parametri formali possono essere di due categorie:

- **di un tipo valore;**
- **di un tipo riferimento;**
- **parametri formali di un tipo valore: inizializzati con il valore del parametro attuale (*rvalue* in C++);**
- **parametri formali di un tipo riferimento: inizializzati con l'indirizzo del parametro attuale (*lvalue* in C++).**

Parametri valore (1)

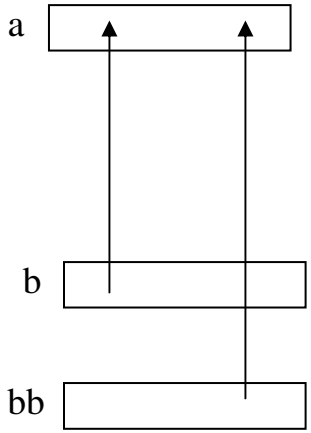
- **Parametri di un tipo valore:**
  - **alla chiamata, assume un valore iniziale dato da un'espressione che produce un *rvalue* (parametro attuale).**
  - **le elaborazioni specificate nel corpo della funzione avvengono sul parametro formale inizializzato;**
  - **casi tipici di inizializzazione:**
    - **con il valore di una variabile (globale o locale del chiamante) di un tipo valore;**
    - **con il valore di un parametro formale del chiamante di un tipo valore, certamente già inizializzato.**



Parametri valore (2)

- **Esempio precedente:**

```
void secondo (int a )
{       int aa; ...
        aa = 1; a = a + 10; ...
}
void primo (int b)
{       int bb;  ...
        secondo(b); …
        secondo(bb); ...
 }
```



- **Istanze di *secondo()*:**
  - **ambiente locale della prima istanza:**
    - **variabile locale *aa* e parametro formale *a* inizializzato col *valore di b*,**
  - **ambiente locale della seconda istanza:**
    - **variabile locale *aa* e parametro formale *a* inizializzato col *valore di bb;***
  - **la variabile locale *aa* e il parametro formale *a* di secondo(), essendo entità locali, sono diversi da istanza a istanza (anche se in C++ hanno lo stesso identificatore).**

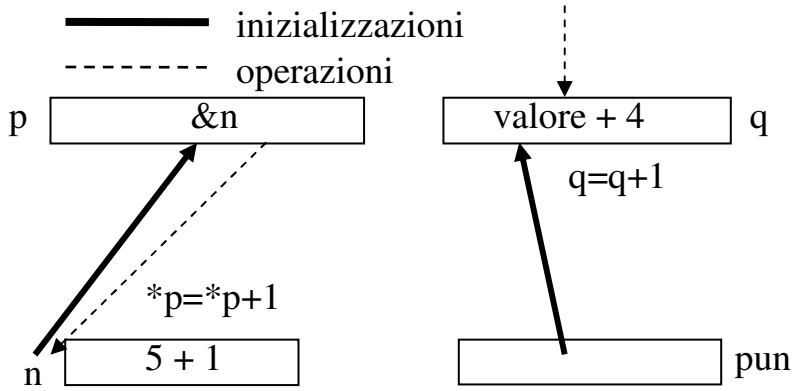
Parametri di un tipo puntatore (1)

- **Parametri di un tipo puntatore (caso particolare di un tipo valore):**
  - **alla chiamata, un parametro formale di un tipo puntatore assume come valore iniziale un' espressione che produce un valore-indirizzo (particolare *rvalue)*;**
  - **casi tipici di inizializzazione:**
    - **il valore di di un parametro formale del chiamante o di una variabile di un tipo puntatore;**
    - **l'indirizzo di una variabile, ossia il suo identificatore C++ preceduto dall'operatore &;**
    - **l'indirizzo di un parametro formale del chiamante di un tipo valore, ossia il suo identificatore C++ preceduto dall'operatore di indirizzo C++ &.**
  - **l'operatore di indirizzo C++ & trasforma un indirizzo (*lvalue*) in un valore-indirizzo (*rvalue*).**
  - **sui valori-indirizzo, contenuti anche nei puntatori, è definita una specifica aritmetica, per cui gli operatori + e – moltiplicano l'operando di destra per il numero di byte del tipo del puntatore.**
- **Dereferenziazioni esplicite:**
  - **nel corpo della funzione, le elaborazioni possono avvenire sia sui parametri formali inizializzati, sia (mediante dereferenziazioni esplicite) sulle entità indirizzate dai parametri attuali (ossia indirizzate dai valori attuali dei puntatori);**
  - **l'operatore di dereferenziazione C++ * trasforma il contenuto di un puntatore, un *rvalue*, in un indirizzo, un *lvalue*, quello della variabile indirizzata.**

Parametri di un tipo puntatore (2)

Esempio:

```
extern "C" void due(int* p , int*  q)
{    *p = *p+1; // aggiunge 1 all'intero puntato da p
     q = q+1;   // aggiunge 4 al contenuto di q
}                // aritmetica degli indirizzi
extern "C" void uno()
{    int n = 5; int* pun;
     // …
     due(&n, pun);
     // …
}
```



- **Ambiente locale dell'istanza di *due()*:**
  - **parametro formale puntatore *p*, inizializzato col valore-indirizzo di *n*;**
  - **parametro formale puntatore *q*, inizializzato col valore di *pun;***
- **Azioni:**
  - **incremento della variabile indirizzata da *p*, ossia di *n*;**
  - **somma di 4 al contenuto di *q* (aritmetica degli indirizzi).**

Parametri di un tipo array (1)

- **Parametri formali di un tipo array:**
  - **corrispondono a parametri formali di un tipo puntatore (al tipo degli elementi).**
- **Formalismo utilizzato in C++ per i parametri di un tipo array:**
  - **un parametro formale array si indica con *a[]*, e corrisponde a un puntatore *a* al primo elemento (indipendentemente dal numero di elementi);**
  - **il corrispondente parametro attuale array è l'identificatore di un array *ar*, e corrisponde all'indirizzo del primo elemento *&ar[0]* (*ar* è il valore iniziale di *a)*;**
  - **nel corpo della funzione, l'indirizzo di un elemento *a[i],* corrisponde ad **(a + i)*, con utilizzo della dereferenziazione (o indirezione) e dell'aritmetica degli indirizzi; possono venir modificati sia gli elementi dell'array attuale, (aventi indirizzo **(a + i)* = *(&ar[0]+i)), che restano in vita dopo l'istanza, sia il puntatore *a* (il cui valore attuale è &ar[0]), che invece muore con la fine dell'istanza.**
- **Oltre alla precedente notazione, propria degli array, in C++ si può utilizzare esplicitamente quella con i puntatori ordinari;**
  - **per il risultato, non essendo definito un tipo come *int[],* si deve utilizzare un puntatore ordinario, come *int**.**

Parametri di un tipo array (2)

- **Esempio 1 (notazione propria degli array):**

```
extern "C" int* due(int a[], int b[], int n)  //l'array attuale ar (formale a) viene modificato
{       int i, int s1 = 0, s2 = 0;
        for(i=0; i<n; i++) s1 = s1 + a[i];
        for(i=0; i<n; i++) s2 = s2 + b[i];
        for(i=0; i<n; i++) a[i] = a[i] + b[i];
        if (s1 < s2) a = b; return a;          // il puntatore attuale ar (formale a):
                                               // viene modificato dalla funzione, ma muore.
        // risultato della funzione : puntatore a all'array in cui gli elementi hanno
        // somma  maggiore;
}
extern "C" void uno()
{       int i; int* cr;  // viene restituito in ar[] il puntatore all'array somma dei due,
                         // in cr[] il puntatore all'array con la somma degli elementi maggiore
        int ar[5] = { 2, 2, 1, 2, 1 };         // s1 = 8
        int br[5] = { 1, 9, 5, 0, 0 };         // s2 = 15
        cr = due(ar, br, 5);                   // s1 < s2
        for(i=0; i<n; i++) cout << ar[i] << ' '; cout <<endl;  // 3, 11, 6, 2, 1
        for(i=0; i<n; i++) cout << cr[i] << '  '; cout << endl;  // 1, 9, 5, 0, 0
}
```

Parametri di un tipo array (3)

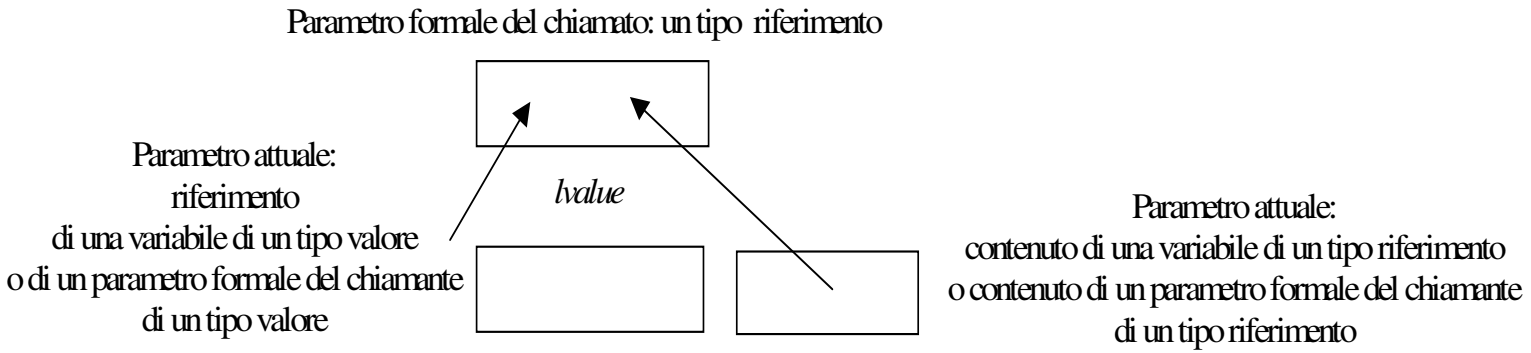
- **Esempio 2 (notazione con puntatori espliciti)**

```
extern "C" int* due(int* a, int* b, int n,)
{       int i; int s1 = 0, s2 = 0;
        for(i=0; i<n; i++) s1 =  s1 +  *(a+i);
        for(i=0; i<n; i++) s2 =  s2 + *(b+i);
        if (s1 > s2)  a = b; return a;

 extern "C" int main()
{       int i; int* cr;
        int ar [5] = { 2, 6, 7, 2, 1 };
        int br[5] = { 1, 9, 5, 0, 0 };
        cr = fun(ar, br, 5,);
        for(i=0; i<n; i++) cout << *(ar+i) << ' '; cout << endl;
        for(i=0; i<n; i++) cout << *(cr+i) << ' '; cout << endl;
}
```

Parametri di un tipo riferimento (1)

- **Parametri formali di un tipo riferimento:**
  - **alla chiamata, un parametro formale di un tipo riferimento assume un valore iniziale dato da un'espressione che produce un *lvalue* (parametro attuale);**
  - **le elaborazioni specificate nel corpo della funzione avvengono sulla entità il cui indirizzo è contenuto del parametro formale (inizializzato), ossia sulla entità specificata dal parametro attuale;**
  - **casi tipici di inizializzazione:**
    - **riferimento di una variabile di un tipo valore o di un parametro formale del chiamante di un tipo valore (certamente già inizializzato) (tipicamente, il suo identificatore C++);**
    - **contenuto di una variabile di un tipo riferimento o di un parametro formale del chiamante di un tipo riferimento (parametro certamente già inizializzato);**

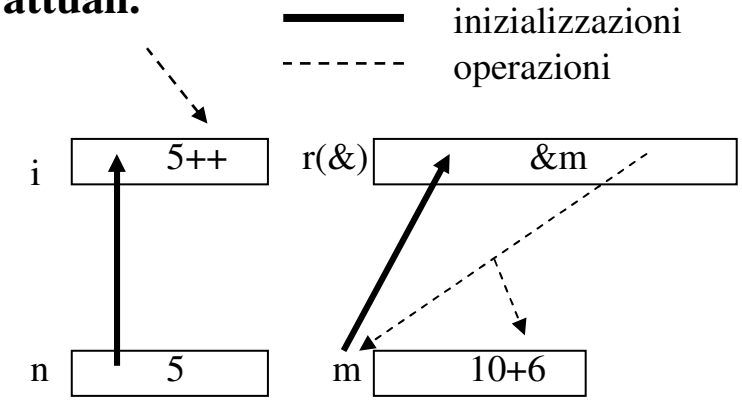


Parametri di un tipo riferimento (2)

- **Inizializzazione dei parametri riferimento:**
  - **il Compilatore, utilizzando istruzioni macchina appropriate, considera i parametri formali C++ di un tipo riferimento come parametri formali del corrispondente tipo puntatore (esempio: int& viene considerato int*), e li inizializza, con parametri attuali valori-indirizzo, realizzando l'operatore *C++* &.**
- **Elaborazione che coinvolgono parametri riferimento:**
  - **quando viene indirizzato un parametro formale, viene aggiunto l'operatore C++ *, e quindi selezionata l'entità il cui indirizzo è contenuto nel parametro formale stesso;**
  - **quindi, non si possono indirizzare i parametri formali, ma solamente le entità indirizzate dai parametri attuali.**
- **Esempio:**

```
extern "C" void due(int i, int& r)
{       i++;
        r = r + i;
}

extern "C" void uno()
{       int n = 5, m = 10;
        due(n, m);
        // …
 }
```



Parametri riferimento e parametri puntatore

- **Parametri formali di un tipo riferimento:**
  - **non è però possibile, a causa della dereferenziazione, riferire il puntatore, per effettuarne letture o modifiche.**
  - **pertanto, i parametri riferimento vengono realizzati con puntatori *nascosti* (al programmatore) e (conseguentemente) *costanti*.**
- **Potenzialità espressiva dei riferimenti:**
  - **inferiore a quella dei puntatori (non possono essere indirizzati);**
  - **tuttavia, non richiedono al programmatore C++ operatori appositi né per l'inizializzazione (&), né per la dereferenziazione (*).**
- **Realizzazione dei parametri riferimento di puntatori:**
  - **come puntatori a puntatori.**

Convenzioni utilizzate (1)

- **Tipi di dato esaminati:**
  - **quelli discreti, manipolati dalla ALU:**
  - **non vengono trattati i tipi reali, manipolati dalla FPU.**
- **Variabili globali (tempo di vita statico):**
  - **memorizzate in locazioni di memoria (zona *.data*)**
- **Entità locali (tempo di vita automatico):**
  - **variabili: memorizzate in pila**
  - **parametri formali: memorizzati provvisoriamente nei registri e veicolati in pila.**
- **Tipi dei parametri formali:**
  - **tipi *singoli* (esclusi i tipi reali): carattere, interi, naturali, booleano, enumerati, puntatori, array, riferimenti;**
    - **i tipi array sono considerati singoli, in quanto corrispondono a tipi puntatore;**
  - **tipi *composti*: strutture, unioni, classi;**
- **Regole valide per tipi singoli:**
  - **trasmissione dei parametri nei registri RDI, RSI, RDX, RCX, R8, R9, uno per registro, in numero non superiore a 6 ;**
    - **chiamante: pone i parametri attuali (valori o riferimenti), utilizzando i precedenti registri;**
    - **chiamato: lascia spazio in pila per la memorizzazione dei parametri formali e delle variabili locali (numero di byte multiplo di 8);**
    - **ricopia i parametri attuali dai registri nei parametri formali (veicolamento);**
  - **per funzioni non *void*, lascia il risultato nel registro RAX.**

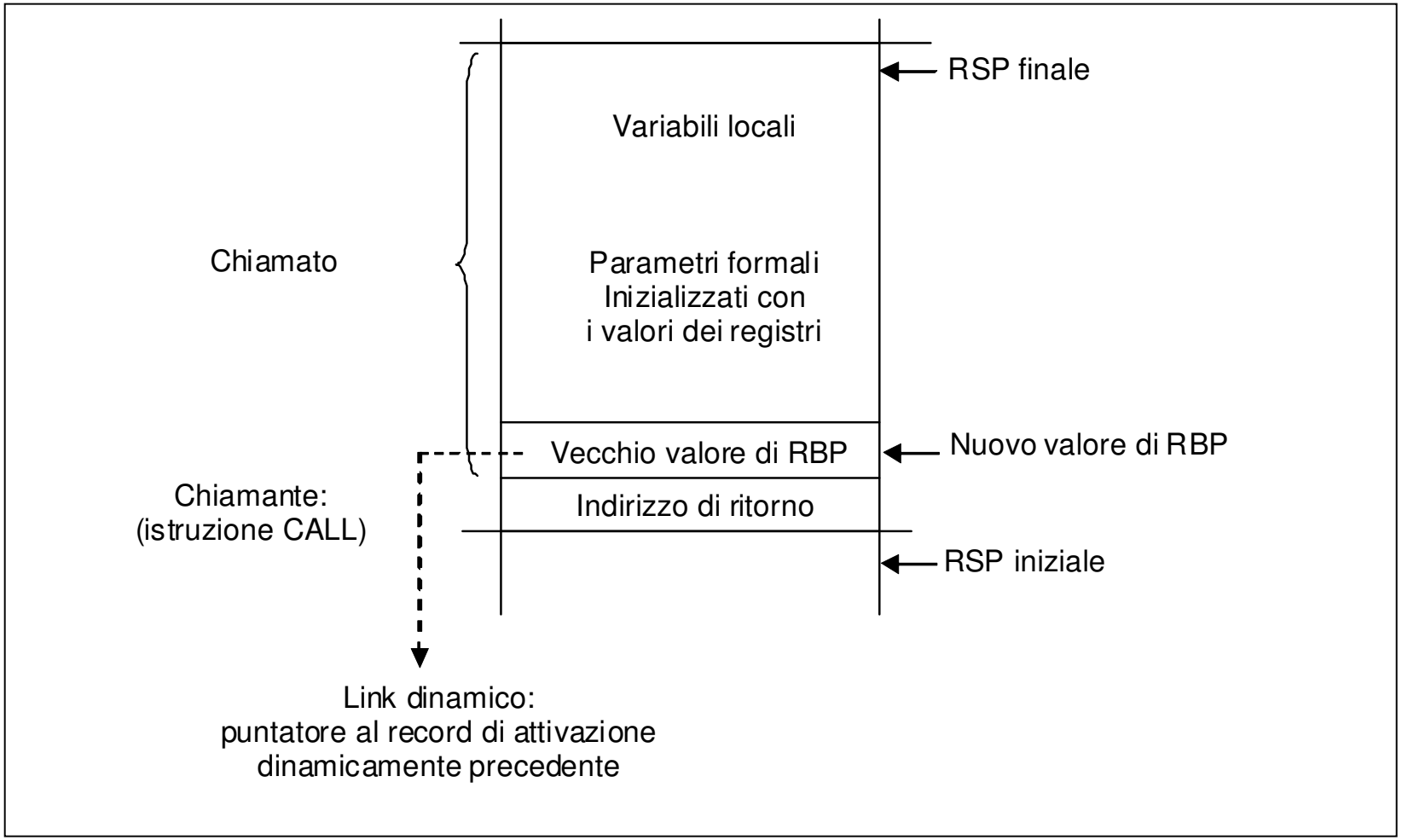
Convenzioni utilizzate (2)

- **Ricopiamento dei parametri attuali in pila:**
  - **azione che in genere è opportuno eseguire;**
  - **per esempio, nel caso in cui il chiamato effettui una nuova chiamata, e quindi specifichi nuovi parametri attuali, inserisce nuovi valori nei registri RDI, RSI, RDX, RCX, R8, R9, distruggendo i valori precedenti.**
- **Sottoprogramma (può essere in Assembler o in C++):**
  - **se in Assembler, è opportuno che si comporti come una funzione C++, che preserva i valori dei registri RSP, RBP, RBX, R12-R15.**
  - **pertanto:**
    - **quando chiama un altro sottoprogramma, può lasciare informazioni nei registri RSP, RBP, RBX, R12-R15, in quanto il contenuto di questi non viene modificato dal chiamato.**
    - **quando viene chiamato da un altro sottoprogramma:**
      - **se usa qualcuno dei registri fra RSP, RBP, RBX, R12-R15 deve all'inizio salvarne il valore e alla fine ripristinare il valore stesso;**
      - **non occorre che salvi e ripristini i valori degli altri registri, in particolare di quelli utilizzabili per trasmettere i parametri attuali o per memorizzare l'eventuale risultato (RAX, RDI, RSI, RDX, RCX, R8, R9).**

Record di attivazione (1)

- **Istanza di funzione:**
  - **costruzione sulla pila di un *record di attivazione* (o *frame*), che contiene le informazioni utili per quella istanza.**
- **Record di attivazione:**
  - **utilizza come registro base (o registro puntatore) RBP, che individua i parametri formali e le variabili locali, oltre ad altre informazioni utili per quella istanza.**
- **Costruzione di un nuovo record di attivazione:**
  - **iniziata dal chiamante;**
    - **dopo aver lasciato i parametri attuali nei registri, effettua l'istruzione di chiamata CALL;**
  - **terminata dal chiamato (le seguenti prime tre azioni costituiscono il *prologo*);**
    - **costruisce il link dinamico (salva in pila il contenuto di RBP);**
    - **stabilisce il nuovo valore del registro base RBP, trasferendovi il valore attuale di RSP;**
    - **lascia in pila spazio per i parametri formali e per le variabili locali, decrementando opportunamente RSP (di un multiplo di 8);**
    - **inizializza i parametri formali con il contenuto dei registri contenenti i parametri attuali (veicolamento dei paramertri attuali).**

Record di attivazione (2)



Record di attivazione (3)

- **Distruzione del record di attivazione:**
  - **effettuata dal chiamato (*epilogo*);**
    - **rimozione dalla pila dello spazio per i parametri formali e per le variabili locali (ricopiamento in RSP del valore di RBP):**
    - **eliminazione del link dinamico (ripristino del vecchio valore di RBP (istruzione POPQ);**
    - **estrazione dalla pila dell'indirizzo di ritorno (istruzione RET).**
- **Istruzioni del prologo e dell'epilogo di una funzione:**
  - **prologo:**

```
pushq   %rbp
movq    %rsp,%rbp
subq    $(par+var), %rsp
```

  - **epilogo:**

```
movq    %rbp, %rsp
popq    %rbp
ret

oppure:
leave
ret
```

- **Allineamento:**
  - **l'allineamento dei dati in pila viene effettuato dal chiamato con il veicolamento;**
    - **la pila è allineata a multipli di 8, e le singole entità sono allineate a multipli della loro lunghezza;**
  - **il chiamante immette semplicemente i parametri nei registri.**

Codice di una funzione

- **Dopo il prologo:**
  - **il corpo inizializza i parametri formali, ai quali è stato lasciato spazio in pila, con i corrispondenti parametri attuali, contenuti nei registri di 64 bit (o in parti di essi);**
    - **esempio (4 parametri di tipo intero, ciascuno di 4 byte):**

```
subq    $16, %rsp           # nel prologo

movl    %edi, -16(%rbp)     # veicolamento dei parametri
movl    %esi, -12(%rbp)
movl    %edx, -8(%rbp)
movl    %ecx, -4(%rbp)
```

  - **salva in pila i contenuti di eventuali registri utilizzati, fra RBX, R12-R15 (RSP ed RBP vengono gestiti (in modo appropriato) solo nel prologo e nell'epilogo);**
  - **esegue le istruzioni relative alle operazioni specificate (codice);**
  - **ripristina i valori salvati dei registri.**

Indirizzi di entità globali

- **Funzione:**
  - **ha una singola copia di codice;**
- **Il codice deve riferire:**
  - **tutte le funzioni e le variabili globali (ambiente globale);**
  - **per ogni specifica istanza, l'ambiente locale.**
- **Proprietà dell'ambiente globale:**
  - **gli indirizzi di memoria delle entità globali sono uguali per tutte le istanze.**
- **Entità dell'ambiente globale:**
  - **riferita secondo quanto specificato col modello di memoria,: uilizzando anche identificatori C++.**

Indirizzamento di entità locali

- **Entità dell'ambiente locale:**
  - **si deve indirizzare con *un'espressione canonica*, utilizzando come registro base RBP (che contiene l'indirizzo del record di attivazione), e uno spiazzamento numerico che individua l'entità locale all'interno del record di attivazione stesso (nessun uso di identificatori C++);**
  - **con questa modalità di indirizzamento, l'indirizzo viene generato a tempo di esecuzione, ed è diverso per ogni istanza, cambiando il contenuto di RBP;**

```
movl   spiazzamento_numerico(%rbp), %eax
```

- **Parametri formali valori-indirizzo:**
  - **il parametro formale può essere un puntatore, nel qual caso il programmatore *può* effettuare una successiva dereferenziazione esplicita;**

```
movq    spiazzamento_numerico(%rbp, …), %rax
movl    (%rax), %ebx          # eventuale dereferenziazione
```

Parametri formali riferimento:

- **il Compilatore C++, in traduzione, introduce automaticamente le dereferenziazione del parametro formale, per cui in Assembler il programmatore *deve* effettuare obbligatoriamente la dereferenziazione:**

```
movq    spiazzamento_numerico(%rbp, …), %rax
movl  (%rax), %ebx            # dereferenziazione necessaria.
```

Indirizzo di memoria e dereferenziazione

- **Operatore C++ & (in Assembler, l'indirizzo è comunque di 64 bit):**
  - **l'indirizzo di un'entità globale può essere memorizzato sia con un'istruzione MOVQ (MOVABSQ) con operando immediato (simbolo &, modelli utilizzati piccolo e grande), che con un'istruzione LEAQ (qualunque modello), le quali trasformano un *lvalue* in un *rvalue*:**
  - **l'indirizzo di una entità locale deve essere necessariamente memorizzato con un'istruzione LEAQ, utilizzando un'espressione canonica, in questa forma:**

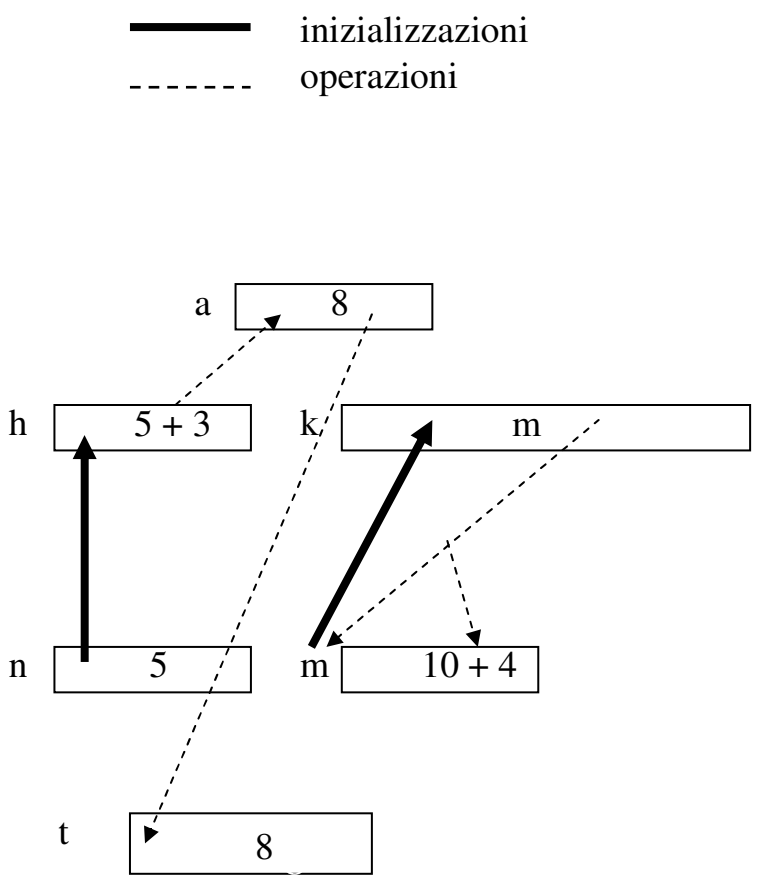
```
leaq      spiazzamento numerico(%rbp), %regq
```

  - **come detto, l'indirizzo viene calcolato a tempo di esecuzione, ed è diverso per ogni istanza, cambiando il contenuto di RBP.**
  - **Operatore C++ *:**
  - **la dereferenziazione C++ *, che, per le istruzioni operative, trasforma un *r-value* in un *l-value*, non corrisponde a un indirizzamento indiretto in Assembler, previsto solo per le istruzioni di salto incondizionato, ma a un indirizzamento con registro puntatore (indicato con una coppia di parentesi tonde), che è un caso particolare di un indirizzamento di memoria canonico (senza Displacement e senza Registro indice).**
- **Dereferenziazione:**
  - **non esiste la dereferenziazione automatica, ma va espressamente indicata.**

Caso illustrativo I (1)

```
int n = 5, m =10;
// …
extern "C" int fai(int h, int& k)
{       int a;
        // …
        h = h + 3;
        a = h;
        k = k + 4;
        // …
        return a;

}
int main()
{       int t;
        // …
        t = fai(n, m);
        // …
}
```



Caso illustrativo I (2)

```
.data
.global n, m
n:   .long   5
m:   .long   10

.text
.global fai
.set  a, -16  # intero, 4 byte
.set  h, -12  # intero, 4 byte
.set  k, -8    # riferimento (puntatore), 8 byte
fai:           # a, h e k non sono identificatori
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp

      movl   %edi, h(%rbp)
      movq  %rsi, k(%rbp)

      addl    $3, h(%rbp)       # h=h+3;

      movl   h(%rbp), %eax   # a=h;
      movl   %eax, a(%rbp)

     movq     k(%rbp), %rax      # *k=*k+4;
     movl     (%rax), %ebx
     addl     $4, %ebx
     movl     %ebx, (%rax)

     movl     a(%rbp), %eax      # return a;

     leave    # muoiono k ed a,
                              # contenuto di a
restituito
     ret

.global main
.set t, -8      # intero, 4 byte arrotondato a 8
main:           # n e m sono identificatori, t no
     pushq   %rbp                        # prologo
     movq    %rsp, %rbp
     subq     $8, %rsp

     movl     n(%rip), %edi    # fai(val. di n,
&m)
     leaq     m(%rip), %rsi
     call     fai
     movl     %eax, t(%rbp)    # risultato in t

     movl     $0, %eax

     leave                              # epilogo
     ret
```

Parametri attuali del chiamato: entità locali del chiamante

- **DISP:**
  - **numero riferito al record di attivazione del chiamante**

valore-valore

```
movl  DISP(%rbp), %edi      # entità locale del chiamante: valore di tipo int
                            # trasmesso il valore dell'entità
```

indirizzo-valore

```
movq DISP(%rbp), %rax       # entità locale del chiamante : indirizzo di tipo int*
movl  (%rax), %edi          # trasmesso il valore dell'intero indirizzato
```

valore-indirizzo

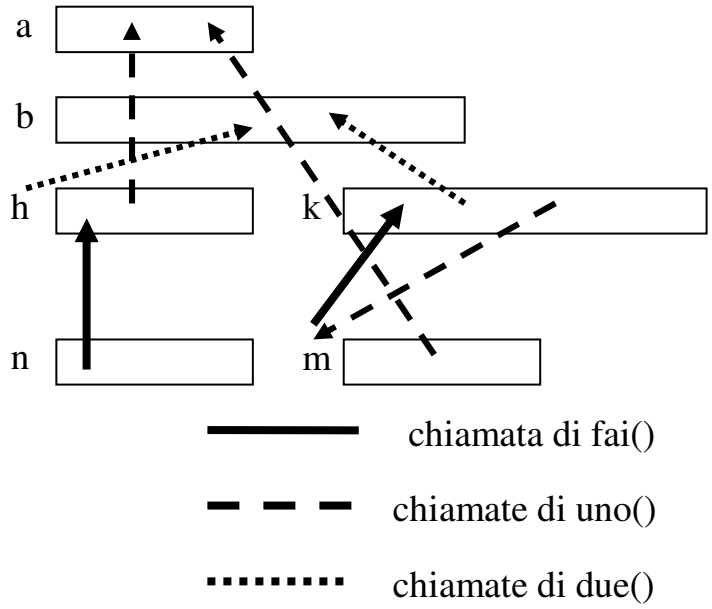
```
leaq   DISP(%rbp), %rdi     # entità locale del chiamante: valore di tipo int
                            # trasmesso l'indirizzo dell'entità
```

indirizzo-indirizzo

```
movq DISP(%rbp), %rdi       # entità locale del chiamante: indirizzo di tipo int*
                            # trasmesso l'indirizzo stesso
```

Caso illustrativo II (1)

```
int n =5, m =10;
extern "C" void uno(int a)
 {   /* …*/    }
extern "C" void due(int& b)
 {   /* …*/    }
extern "C" void fai(int h, int& k)
{      // …
       uno(h);        # realizzazione: uno(h)
       // parametro valore trasmesso per valore
       uno(k);        # realizzazione: uno(*k)
       // parametro riferimento trasmesso per valore
       due(h);        # realizzazione: due(&h)
      // parametro valore trasmesso per riferimento
       due(k);        # realizzazione: due(k)
      // parametro riferimento trasmesso per riferimento
}
int main()
{       // …
        fai(n, m);
}
```



Caso illustrativo II (2)

```
.data
.global n, m
n:   .long   5
m:   .long   10

.text
.global uno        # uno(int a)
uno: # ...
.global due        # due(int& b)
due:         # ...
.global fai        # fai(int h, int& k)
.set  h, -16 # intero, 4 byte, allineati a 8
.set  k, -8   # riferimento, 8 byte
fai:          # h e k non sono identificatori
      pushq %rbp                  # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp

      movl   %edi, h(%rbp)
      movq  %rsi, k(%rbp)
      movl    h(%rbp), %edi   # uno(h) (val-val)
      call      uno
      movq   k(%rbp), %rax   # uno(*k) (ind_val)
      movl    (%rax), %edi
      call      uno
      leaq     h(%rbp), %rdi    # due(&h) (val-ind)
      call      due
      movq   k(%rbp), %rdi    # due(k)    (ind-ind)
      call      due
      leave                                          #
epilogo
      ret

.global main
main:                    # n e m sono identificatori
      pushq  %rbp                      # prologo
      movq   %rsp, %rbp

      movl    n(%rip), %edi    # n
      leaq     m(%rip), %rsi    # & m
      call      fai

      movl    $0, %eax

      leave                          # epilogo
      ret
```

Funzioni semplicissime

- **Caratteristiche:**
  - **chiamata di un'altra funzione: assente;**
  - **variabili locali: assenti;**
  - **parametri: pochissimi;**
  - **codice: semplicissimo.**
- **Realizzazione:**
  - **prologo: assente.**
  - **epilogo: sola istruzione *ret*.**
  - **ricopiamento dei parametri attuali dai registri in pila: assente.**
- **Esempio C++:**

```
extern "C" int fun(int a)
{   a++;
    return a;
}
```

- **Realizzazione:**

```
.text
.global fun
fun:
    incl    %edi
    movl    %edi, %eax
    ret
```

Esempi riassuntivi seguenti

- **In ognuno degli esempi seguenti, sia Esempio*N* (*N* va da 1 a 14):**
  - **versione C++: programma esempio*N*, due file es*N*a.cpp ed es*N*b.cpp;**
  - **versione equivalente Assembler: due file es*N*a.s ed es*N*b.s**
- **Si può ottenere un programma eseguibile equivalente (nel file *a.out*) in uno di questi quattro modi possibili:**

  g++ esNa.cpp esNb.cpp

  g++ esNa.cpp esNb.s

  g++ esNa.s esNb.cpp

  g++ esNa.s esNb.s

Esempio 1: somma con variabili globali *alfa* e *beta* (1)

```
// programma sommmaintGlob, file es1a.cpp
#include"servi.cpp"
extern "C" int elab1(int n, int m);
int alfa, beta;
int main()
{     int ris;
      alfa = leggiint(); beta = leggiint();
      ris = elab1(alfa, beta);
      scriviint(ris); nuovalinea();
      return 0;
};

// programma sommmaintGlob, file es1b.cpp
extern "C" int elab1(int n1, int n2)
{     int i, j;
      i = n1+n2;
      j = n1-n2;
      return i*j;
};
```

Esempio 1: le variabili globali vengono indirizzate da *main()* tramite RIP (2)

```
# programma sommaintGlob, file es1a.s
# registri per i parametri: rdi, rsi
# registro per il  risultato: rax
.include "servi.s"
.data
alfa:       .long       0
beta:       .long       0

.text
# .extern elab1
.global main
.set  ris, -8
main:
      pushq %rbp                # prologo
      movq  %rsp, %rbp
      subq   $8, %rsp
      # ris: 4 byte (allineamento a 8)

      call     leggiint
      movl   %eax, alfa(%rip)  # risultato  in alfa
      call     leggiint
      movl   %eax, beta(%rip) # risultato in beta

      movl   alfa(%rip), %edi
      movl   beta(%rip), %esi
      call     elab1
      movl   %eax, ris(%rbp)  # risultato in ris

      movl   ris(%rbp), %edi   # parametro ris
      call     scriviint
      call     nuovalinea

      movl  $0, %eax

      leave                          # epilogo
      ret
```

Esempio 1: somma con variabili globali, ricopiate da *elab1()* in *n1*, *n2* e indirizzate tramite RBP (3)

```
# programma somimaintGlob, file es1b.s
# registri per i parametri: rdi, rsi
# registro per il  risultato: rax
.text
.global elab1
.set  i, -16
.set  j, -12
.set  n1, -8
.set  n2, -4
elab1:
      pushq %rbp                    # prologo
      movq  %rsp, %rbp
      subq    $16, %rsp
      # i: 4 byte, j: 4 byte, n1: 4 byte, n2: 4byte

      movl   %edi, n1(%rbp)
      movl   %esi, n2(%rbp)

      movl   n1(%rbp), %eax
      addl   n2(%rbp), %eax
      movl   %eax, i(%rbp)          # i = n1+ n2

      movl   n1(%rbp), %eax
      subl   n2(%rbp), %eax
      movl   %eax, j(%rbp)          # j = n1- n2

      movl   i(%rbp), %eax          # ritorna i * j
      imull  j(%rbp), %eax

      leave                         # epilogo
      ret
```

Esempio 2: somma con variabili locali *a*, *b* e *ris* (1)

```
// programma sommaintLoc, file es2a.cpp
#include"servi.cpp"
extern "C" int elab2(int n, int m);
int main()
{     int a, b, ris;
      a = leggiint(); b = leggiint();
      ris = elab2(a, b);
      scriviint(ris); nuovalinea();
      return 0;
};

// programma sommaintLoc, file es2b.cpp
extern "C" int elab2(int n1, int n2)
{     int i, j;
      i = n1+ n2;
      j = n1- n2;
      return i*j;
};
```

Esempio 2: somma con variabili locali ricopiate da *main()* nei registri, utilizzando RBP (2)

```
# programma sommaintLoc, file es2a.s
# registri per i parametri: rdi, rsi
# registro per il  risultato: rax
.include "servi.s"
.text
# .extern elab2
.global main
.set  a, -12
.set  b, -8
.set  ris, -4
main:
      pushq %rbp                  # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp
      # a: 4 byte, b: 4 byte,ris: 4 byte,
      # allineamento

      call     leggiint
      movl   %eax, a(%rbp)   # risultato in a
      call     leggiint
      movl   %eax, b(%rbp)   # risultato in b

      movl    a(%rbp), %edi
      movl    b(%rbp), %esi
      call     elab2
      movl    %eax, ris(%rbp) # risultato  in ris

      movl    ris(%rbp), %edi  # parametro ris
      call      scriviint
      call     nuovalinea

      movl   $0, %eax

      leave                       # epilogo
      ret
```

Esempio 2: somma con variabili locali

```
# programma sommaintLoc, file es2b.s
# registri per i parametri: rdi, rsi
# registri per il  risultato: rax
.text
.global elab2
.set  i, -16
.set  j, -12
.set  n1, -8
.set  n2, -4
elab2:
      …
      # come il file  elab1.s
      …
     leave
     ret
```

Esempio 3: somma con parametro riferimento (1)

```
// programma sommaintRif, file es3a.cpp
#include"servi.cpp"
extern "C" void elab3(int& tot, int n1, int n2);
int main()
{     int a, b; int& ris;
      a = leggiint(); b = leggiint();
      elab3(ris, a, b);
      scriviint(ris); nuovalinea();
};

// programma sommaintRif, file es3b.cpp
extern "C" void elab3(int& tot, int n1, int n2)
{     int i, j;
      i = n1+n2;
      j = n1-n2;
      tot = i*j;
};
```

Esempio 3: somma con parametro formale riferimento *tot* e parametro attuale riferimento *ris* (locale a *main()*) (2)

```
# programma sommaintRif, file es3a.s
# registri per i parametri: rdi, rsi, rdx
# registro per il  risultato: rax
.include  "servi.s"
.text
# .extern   elab3
.global   main
.set  a, -16
.set  b, -12
.set  ris, -4     # è un int
                  # trasmesso a elab3 con LEA
main:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp
      # a: 4 byte, b: 4 byte, ris:  8 byte

      call     leggiint
      movl   %eax, a(%rbp)
      call     leggiint
      movl   %eax, b(%rbp)
```

```
      leaq    ris(%rbp), %rdi  # parametro &ris
      movl   a(%rbp), %esi    # parametro a
      movl   b(%rbp), %edx   # parametro b
      call     elab3
      # ris: intero per il risultato
      # riferimento attuale: indirizzo di ris
      movl    ris(%rbp), %edi
      call      scriviint
      call      nuovalinea

      movl    $0, %eax

      leave                          # epilogo
      # muore ris,
      ret
```

Esempio 3: somma con parametro riferimento (3)

```
# programma sommaintRif, file es3b.s
# registri per i parametri: rdi, rsi, rdx
# registro per il  risultato: rax
.text
.global elab3
.set  i, -24
.set  j, -20
.set  tot, -16                # int&
.set  n1, -8
.set  n2, -4
elab3:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $24, %rsp
      # i: 4 byte, j: 4 byte
      # tot: 8 byte; i, j, n1, n2 : 4 byte
      # in %rdi  c'è  l'indirizzo di ris,
      # che va in tot
      movq  %rdi, tot(%rbp)
      movl   %esi,  n1(%rbp)
      movl   %edx, n2(%rbp)
      movl    n1(%rbp), %eax
      addl    n2(%rbp), %eax
      movl     %eax, i(%rbp)      # i = n1+n2
      movl     n1(%rbp), %eax
      subl      n2(%rbp), %eax
      movl      %eax, j(%rbp)     # j = n1-n2

      movq    tot(%rbp), %rdx   # tot: contiene ris&
      movl     i(%rbp), %eax
      imull    j(%rbp), %eax
      movl     %eax, (%rdx)
      # risultato i*j va nella variabile riferita da tot
      # (tot) = i*j

      leave                           # epilogo
     # muoiono n1, n2, i, j, tot, sopravvive (tot)
      ret
```

Esempio 4 : parametro puntatore (1)

```
// programma puntatore, fle es4a.cpp
#include "servi.cpp"
extern "C" void add(int* p, int i);
int main()
{     int a, b;
      a = leggiint(); b = leggiint();
      add(&a, b);
      scriviint(a); nuovalinea();
      return 0;
};

// programma puntatore, file es4b.cpp
extern "C" void add (int* p, int i)
{     *p = *p + i;          // attenzione: non p = p+i, ma *p = *p + i
                            // aritmetica degli interi e non aritmetica degli indirizzi
};
```

Esempio 4 : parametro puntatore (2)

```
# programma puntatore, file es4a.s
# registri per i parametri: rdi, rsi
.include "servi.s"
.text
# .extern add
.global main
.set  a, -8
.set  b, -4
main:
      pushq %rbp            # prologo
      movq  %rsp, %rbp
      subq   $8, %rsp
      # a: 4 byte, b: 4 byte

      call     leggiint
      movl   %eax, a(%rbp)
      call     leggiint
      movl   %eax, b(%rbp)
```

```
      leaq     a(%rbp), %rdi   # &a
      movl    b(%rbp), %esi
      call      add
      movl     a(%rbp), %edi
      call      scriviint
      call      nuovalinea

      movl     $0, %eax

      leave                    # epilogo
      ret
```

Esempio 4 : parametro puntatore (3)

```
# programma puntatore, file es4b.s
# registri per i parametri: rdi, rsi
.text
.global add
.set  p, -16
.set  i, -8
add:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp
      # p: 8 byte, i: 4 byte

      movq  %rdi, p(%rbp)
      movl   %esi, i(%rbp)

      movq  p(%rbp), %rax
      movl   (%rax), %ebx
      addl    i(%rbp), %ebx
      movl   %ebx, (%rax)

      leave                   # epilogo
      ret
```

```
# programma puntatore, file es6b.s
# versione semplificata
# registri per i parametri: rdi, rsi
.text
.global add
add:
            movq   %rdi, %rax
            movl          (%rax), %ebx
            addl          %esi, %ebx
            movl          %ebx, (%rax)

            ret
```

Esempio 5: parametro riferimento di puntatore (1)

```
// programma puntatoreRif, file es5a.cpp
#include "servi.cpp"
extern "C" void trovamin(int*& p, int* pa, int* pb);
// restituisce in p uno fra pa e pb, a seconda del minimo intero puntato
int main()
{     int n, m; int* pun;
      n = leggiint();
      m = leggiint();
      trovamin(pun, &n, &m);
      scriviint(*pun); nuovalinea();
      return 0;
};

// programma puntatoreRif, file es5b.cpp
extern "C" void trovamin(int*& p, int* pa, int* pb)
{ if (*pa <= *pb) p = pa; else p = pb;
}
```

Esempio 5: parametro riferimento di puntatore (2)

```
// programma puntatoreRif, file es5a.s
# registri per i parametri: rdi, rsi, rdx
# registri per il  risultato: rax
.include "servi.s"
.text
# .extern trovamin
.global main
.set  n, -16
.set  m, -12
.set  pun, -8
main:
      pushq %rbp                  # prologo
      movq  %rsp,  %rbp
      subq   $16, %rsp
      # n: 4 byte, m: 4 byte, pun: 8 byte
      call     leggiint
      movl   %eax, n(%rbp)
      call     leggiint
      movl   %eax, m(%rbp)
```

```
      leaq   pun(%rbp), %rdi  # &pun
      leaq   n(%rbp), %rsi      # &n
      leaq   m(%rbp), %rdx    # &m
      call     trovamin
      # pun (*p) contiene & n o &m
      movq  pun(%rbp), %rax
      movl   (%rax), %edi       # pun vale &m o &n
      call     scriviint            # *pun vale m o n
      call     nuovalinea

      movl   $0, %eax

      leave                          # epilogo
      ret
```

Esempio 5: parametro riferimento di puntatore (3)

```
# programma puntRif, file es5b.s
# registri per i parametri: rdi, rsi, rdx
.text
.global trovamin
.set  p, -24     # p: riferimento di puntatore
.set  pa, -16
.set  pb, -8
trovamin:
      pushq %rbp            # prologo
      movq  %rsp, %rbp
      subq   $24, %rsp
      # p: 8 bye, pa: 8 byte, pb: 8 byte

      movq  %rdi, p(%rbp)
      movq  %rsi, pa(%rbp)
      movq  %rdx, pb(%rbp)

      movq  pa(%rbp), %rdi
      movl   (%rdi), %esi
      movq  pb(%rbp), %rdi
      movl   (%rdi), %edx
      cmpl   %esi, %edx
      jp       oltre          # edx (*pb) > esi (*pa)
```

```
            movq   p(%rbp), %rdi
            movq  pa(%rbp), %rsi
            movq   %rsi, (%rdi)     # pa va in *p
            jmp    avanti
oltre:      movq  p(%rbp), %rdi
            movq  pb(%rbp), %rsi
            movq  %rsi, (%rdi)      # pb va in *p

avanti:  leave                       # epilogo
            ret

          # *p (ossia pun) contiene pa o pb
```

Esempio 6: parametro array (1)

```
// programma array, file es6a.cpp
#include"servi.cpp"
extern "C" void raddoppia(int a[], int n);
int main()
{     int ar[5]; int i;
      for(i=0; i<5; i++) ar[i] = leggiint();
      raddoppia(ar, 5);
      for(i=0; i<5; i++) scriviint(ar[i]);
      nuovalinea();
};

// programma array, file es6b.cpp
extern "C" void raddoppia(int a[], int n)
{     int i;
      for(i=0; i<n; i++) a[i] = 2*a[i];
};
```

Esempio 6: parametro array (2)

```
# programma array, file es6a.s
# registri per i parametri: rdi, rsi
.include "servi.s"
.text
# .extern raddoppia
.global main
.set  ar, -24
.set  i, -4
main:
        pushq       %rbp                    # prologo
        movq        %rsp, %rbp
        subq $24, %rsp
        # a[5]:  5 int = 20 byte, i: 4 byte

        movl $0,  i(%rbp)                   # i = 0
rip1: cmpl $5, i(%rbp)
        jge    avan1                        # salta se i>=5

        call   leggiint
        movslq i(%rbp), %rdi
        movl %eax, ar(%rbp,%rdi,4) # lettura di a[i]
        incl   i(%rbp)                      # i++
        jmp    rip1
```

```
avan1:  leaq    ar(%rbp), %rdi  # &ar[0]
        movl    $5, %esi
        call    raddoppia

        movl    $0, i(%rbp)         # i = 0
rip2:   cmpl    $5, i(%rbp)         # scritture
        jge     avan2
        movslq  i(%rbp), %rsi
        movl    ar(%rbp, %rsi,4), %edi
        call    scriviint
        incl    i(%rbp)             # i++
        jmp     rip2
avan2:  call    nuovalinea

        movl    $0, %eax

        leave                       # epilogo
        ret
```

Esempio 6: parametro array (3)

```
# programma array, file es6b.s
# registri per i parametri: rdi, rsi
.text
.set  i, -24
.set  a, -16      # puntatore al  primo elemento
.set  n, -8
.global raddoppia
raddoppia:
      pushq %rbp                # prologo
      movq  %rsp, %rbp
      subq   $24, %rsp
      # i: 4 byte, a: 8 byte (allineato), n: 4 byte

      movq  %rdi, a(%rbp)     # a (puntatore)
      movl   %esi, n(%rbp)

      movl   $0, i(%rbp)        # i = 0
rip: movl   n(%rbp), %edi
      cmpl    %edi, i(%rbp)
      jge      fine               # salta se i >= n
```

```
            movq      a(%rbp), %rdi
            movslq    i(%rbp), %rsi
            movl      (%rdi, %rsi, 4), %edx
            addl      %edx, %edx
            movl      %edx, (%rdi, %rsi, 4)
            incl      i(%rbp)
            jmp       rip

fine:       leave     # epilogo
            ret
```

Esempio 7: array locale e globale (1)

```
// programma arrayLocGlob, file es7a.cpp
#include"servi.cpp"
int alfa[5];
extern "C" void addar(int a[], int b[], int n);
int main()
{     int beta[5]; int i;
      for (i=0; i<5; i++) alfa[i]= leggiint();
      for (i=0; i<5; i++) beta[i]= leggiint();
      addar(alfa, beta, 5);
      for (i=0; i<5; i++) scriviint(alfa[i]);
      nuovalinea();
      return 0;
};

// programma arrayLocGlob, file es7b.cpp
extern "C" void addar(int a[], int b[], int n)
{     int i;
      for (i=0; i<n; i++) a[i]= a[i]+b[i];
};
```

Esempio 7: array locale e globale (2)

```
# programma arrayLocGlob, file es9a.s
# registri per i parametri: rdi, rsi, rdx
.include "servi.s"

.data
.global alfa
alfa:  .fill    5, 4

.text
#.extern addar
.global main
.set  beta, -24
.set  i,  -4
main:
      pushq %rbp            # prologo
      movq  %rsp, %rbp
      subq   $24, %rsp
```

```
          movl  $0, i(%rbp)      # i = 0
rip1:     cmpl  $5, i(%rbp)
          jge    avan1           # salta se i >= 5
          call   leggiint
          leaq  alfa(%rip), %rsi
          # array globale, indirizzo relativo a rip
          movslq   i(%rbp), %rdi
          movl %eax, (%rsi,%rdi, 4)
          # intero letto in alfa[i]
          incl   i(%rbp)         # i++
          jmp  rip1
avan1: movl  $0, i(%rbp)
rip2:     cmpl  $5, i(%rbp)
          jge    avan2           # salta se i >= 5
          call   leggiint
          movslq   i(%rbp), %rdi
          movl  %eax, beta(%rbp,%rdi,4)
          # array locale, beta non e' un indirizzo,
          # ma un displacement numerico (.set)
          incl    i(%rbp)        # i++
          jmp   rip2
```

Esempio 7: array locale e globale (3)

```
# programma arrayLocGlob, file es9a.s, continua …
avan2:   leaq   alfa(%rip), %rdi      # array globale, indirizzo relativo a rip
         leaq   beta(%rbp), %rsi      # array locale: numero (.set) relativo a rbp
         movl   $5, %edx
         call   addar

         movl  $0, i(%rbp)
 rip3:   cmpl  $5, i(%rbp)
         jge    avan3                 # salta se i >= 5
         leaq   alfa(%rip), %rax      # arry globale
         movslq  i(%rbp), %rsi
         movl   (%rax,%rsi, 4), %edi
         call   scriviint
         incl   i(%rbp)               # i++
         jmp    rip3
avan3:   call   nuovalinea

         movl   $0, %eax

         leave                        # epilogo
         ret
```

Esempio 7: array locale e globale (4)

```
# programma arrayLocGlob, file es7b.s
# registri per i parametri: rdi, rsi, rdx
.text
.set  i, -32
.set  a, -24
.set  b, -16
.set  n, -8
.global addar
addar:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq  $32, %rsp         # allineamento

      movq  %rdi, a(%rbp)
      movq  %rsi, b(%rbp)
      movl  %edx, n(%rbp)

      movl  $0, i(%rbp)       # i = 0
rip:  movl  n(%rbp), %edx
      cmpl  %edx, i(%rbp)
      jge   fine              # salta se i >= n
      movslq  i(%rbp),%rcx
      movq    a(%rbp), %rdi
      movq    b(%rbp), %rsi
      movl    (%rdi, %rcx, 4), %eax
      addl    (%rsi, %rcx, 4), %eax
      movl    %eax, (%rdi, %rcx, 4)
      incl    i(%rbp)
      jmp     rip

fine: leave           # epilogo
      ret
```

Esempio 8: scambio (1)

```
// programma scambio, file es8a.cpp
#include"servi.cpp"
extern "C" void sca(int& r1, int& r2);
int main()
{     int a, b;
      a = leggiint(); b = leggiint();
      sca(a, b);
      scriviint(a); scriviint(b); nuovalinea();
      return 0;
};

// programma scambio, file es8b.cpp
extern "C" void sca(int& r1, int& r2)
{     int lav;
      lav = r1; r1 = r2; r2 = lav;  // non scambia r1 con r2, ma gli interi riferiti da r1 ed r2
};
```

Esempio 8: scambio (2)

```
# programma scambio, file es8a.s
# registri per i parametri: rdi, rsi
.include  "servi.s"
.text
# .extern sca
.global main
.set  a, -8
.set  b, -4
main:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $8, %rsp
      # a: 4 byte, b: 4 byte

      call     leggiint
      movl   %eax, a(%rbp)
      call     leggiint
      movl   %eax, b(%rbp)
```

```
      leaq    a(%rbp), %rdi       # parametro &a
      leaq    b(%rbp), %rsi       # parametro &b
      call     sca

      movl    a(%rbp), %edi
      call     scriviint
      movl     b(%rbp), %edi
      call     scriviint
      call     nuovalinea

      movq    $0, %rax
      leave                        # epilogo
      ret
```

Esempio 8: scambio (3)

```
# programma scambio, file es8b.s
# registri per i parametri: rdi, rsi
.text
.global sca
.set  lav, -24
.set  r1, -16
.set  r2, -8
sca:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $24, %rsp
      # lav: 4 byte, r1: 8 byte, r2: 8 byte

      movq  %rdi, r1(%rbp)
      movq  %rsi, r2(%rbp)
      movq  r1(%rbp), %rax    # lav = *r1
      movl   (%rax), %eax
      movl   %eax, lav(%rbp)
```

```
      movq   r1(%rbp), %rax   # *r1 = *r2
      movq   r2(%rbp), %rdx
      movl    (%rdx), %edx
      movl     %edx, (%rax)

      movq    r2(%rbp), %rax  # *r2 = lav
      movl    lav(%rbp), %edx
      movl    %edx, (%rax)

      leave                           # epilogo
      ret
```

Esempio 9 : massimo riferito (1)

```
// programma massimo, file es9a.cpp
#include"servi.cpp"
extern "C" int& massi(int& ra, int& rb);
int main()
{     int a, b;
      a = leggiint(); b = leggiint();
      massi(a, b) = 0;
      scriviint(a); scriviint(b); nuovalinea();
};

// programma massimo, file es9b.cpp
extern "C" int& massi(int& ra, int& rb)
{     if (ra >= rb) return ra; return rb;
      // i riferimenti vengono automaticamente dereferenziati
      // la funzione restituisce il riferimento alla variabile riferita di valore massimo
};
```

Esempio 9 : massimo riferito (2)

```
# programma massimo, file es9a.s
# registri per i parametri: rdi, rsi,
# registri per il  risultato: rax
.include "servi.s"
.text
# .extern massi
.global main
.set  a, -8
.set  b, -4
main:
      pushq %rbp            # prologo
      movq  %rsp, %rbp
      subq   $8, %rsp
      # a: 4 byte, b: 4 byte

      call     leggiint
      movl   %eax, a(%rbp)
      call     leggiint
      movl   %eax, b(%rbp)
```

```
      leaq   a(%rbp), %rdi    # a &
      leaq   b(%rbp), %rsi    # b &
      call    massi
      movl  $0, (%rax)        # massi() = 0;

      movl   a(%rbp), %edi
      call    scriviint
      movl   b(%rbp), %edi
      call   scriviint
      call    nuovalinea

      movl   $0, %eax

      leave                   # epilogo
      ret
```

Esempio 9 : massimo riferito (3)

```
# programma massimo, file es9b.s
# registri per i parametri: rdi, rsi
# registri per il  risultato: rax,
.text
.global massi
.set  ra, -16
.set  rb, -8
massi:
      pushq %rbp                  # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp
      # ra: 8 byte, rb: 8 byte

      movq  %rdi, ra(%rbp)
      movq  %rsi, rb(%rbp)

      movq  ra(%rbp), %rax   # ra
      movl   (%rax), %edi      # (ra)
      movq  rb(%rbp), %rax   # rb
      movl   (%rax), %esi      # (rb)
      cmpl   %edi, %esi
      jl        oltre               # salta se (rb)<(ra)
```

```
            movq   rb(%rbp),%rax
            jmp      avanti
oltre:      movq   ra(%rbp), %rax
            # rax: puntatore alla variabile
            # di valore maggiore

avanti:     leave        # epilogo
            ret

# risultato in rax: ra oppure rb, essendo la
# funzione di tipo riferimento int &
# se fosse stata di tipo int avremmo avuto:
# la successiva istruzione
#  movl      (%rax), %rax
```

Parametri e risultato di un tipo struttura (o unione)

- **Caso semplice (preso in esame nell'esempio successivo):**
  - **parametri e risultato valore di un tipo struttura, con numero di byte del tipo non superiore a 16:**
  - **per ogni parametro possono essere utilizzati fino a 2 registri consecutivi liberi, fra RDI, RSI, RDX, RCX, R8, R9;**
    - **il numero totale dei registri utilizzati per i parametri non deve comunque superare 6.**
  - **per il risultato può essere utilizzata, in tutto o in parte, la coppia di registri RAX-RDX.**
- **Parametri e risultato di un tipo riferimento di struttura:**
  - **il numero di byte richiesto dal tipo riferito può essere qualsivoglia;**
  - **un suo riferimento occupa sempre un solo registro.**
- **Allineamenti di strutture nel record di attivazione:**
  - **l'ambiente locale occupa un numero di byte multiplo di 8 byte;**
  - **le strutture, per semplicità, si allineano di solito a multipli di 8 byte, anche se l'allineamento minimo da rispettare è determinato dal campo avente vincoli maggiori.**

Esempio 10: parametro struttura (1)

```
// programma struttura, file es10a.cpp
#include"servi.cpp"
struct s { int n1; char c; int n2; };
extern "C" s leggis()
{     s ss;
      ss.n1 = leggiint(); ss.c = leggichar();
                ss.n2 = leggiint();
      return ss;            // in rax, edx
};
extern "C" void scrivis(s ss)
{     scriviint(ss.n1); scrivichar(ss.c);
                scriviint(ss.n2); nuovalinea();
};
extern "C" s fai(s st);
int main()
{     s st1, st2;
      st1 = leggis();
      st2 = fai(st1);
      scrivis(st2);
      return 0;
};
```

```
// programma struttura, file es10b.cpp
struct s { int n1; char c; int n2; };
extern "C" s fai(s st)
{     s ss;
      ss.n1 = st.n1+ 5; ss.c = st.c + 1;
      ss.n2 = st.n2 + 10;
      return ss;
}
```

Esempio 10: parametro struttura (2)

```
# programma struttura, file es10a.s
# registri per il  risultato di legis: rax, rdx
.include "servi.s"
.text
# ss.n1: ss(%rbp), ss.c: ss+4(%rbp)
# ss.n2: ss+8(%rbp)

.set  ss, -16
leggis:
      pushq %rbp          # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp

      call     leggiint
      movl   %eax, ss(%rbp)
      call     leggichar
      movb  %al, ss+4(%rbp)
      call     leggiint
      movl   %eax, ss+8(%rbp)

      movq  ss(%rbp), %rax   # return ss
      movl   ss+8(%rbp), %edx
      leave
      ret
```

```
# registri per il parametro di scrivis: rdi, rsi
.set   ss, -16
 scrivis:
      pushq  %rbp
      movq    %rsp, %rbp
      subq    $16, %rsp

      movq    %rdi, ss(%rbp)       # ss.n1 e ss.c
      movl     %esi, ss+8(%rbp)   # ss.n2

      movl      ss(%rbp), %edi
      call       scriviint
       movb     ss+4(%rbp), %dil
       call       scrivichar
       movl      ss+8(%rbp), %edi
       call       scriviint
       call       nuovalinea

       leave                  # epilogo
       ret
```

Esempio 10: parametro struttura (3)

```
# programma struttura, file es10a.s, continua
      …
# registri per il parametro di fai: rdi, rsi
# registri per il  risultato di fai: rax, rdx
#.extern fai
.global main
.set st1, -32
.set st2, -16
main:
      pushq %rbp                          # prologo
      movq  %rsp, %rbp
      subq   $32, %rsp
      # st1, st2: 16 byte ciascuna

      call     leggis
      movq  %rax, st1(%rbp)
      movl   %edx, st1+8(%rbp)
```

```
movq    st1(%rbp), %rdi
movl    st1+8(%rbp), %esi
call     fai
movq    %rax, st2(%rbp)   # risultato in  st2
movl    %edx, st2+8(%rbp)

movq     st2(%rbp), %rdi
movl     st2+8(%rbp), %esi
call     scrivis

movl     $0, %eax

leave                              # epilogo
ret
```

Esempio 10: parametro struttura (4)

```
# programma struttura, file es10b.s
# registri per il parametro: rdi, rsi
# registri per il  risultato: rax, rdx
.text
.global fai
.set ss, -32
.set st, -16
fai:
      pushq %rbp                    # prologo
      movq  %rsp, %rbp
      subq   $32, %rsp
      # var. ss: 16 byte, par. st: 16 byte,

      movq  %rdi, st(%rbp)
      movl   %esi, st+8(%rbp)

      movl   st(%rbp), %eax
      addl    $5, %eax
      movl   %eax, ss(%rbp)
      movb  st+4(%rbp), %al
      addb   $1, %al
      movb  %al, ss+4(%rbp)
```

```
      movl    st+8(%rbp), %eax
      addl    $10, %eax
      movl    %eax, ss+8(%rbp)

      movq   ss(%rbp), %rax
      movl    ss+8(%rbp), %edx

      leave                         # epilogo
      ret
```

Esempio 11: parametro riferimento di struttura (1)

```
// programma strutturaRif, file es11a.cpp
#include"servi.cpp"
struct s { int n1; char c; int n2; };
extern "C" s leggis()
{     s ss;
      ss.n1 = leggiint(); ss.c = leggichar();
            ss.n2 = leggiint();
      return ss;
};
extern "C" void scrivis(s ss)
{     scriviint(ss.n1); scrivichar(ss.c);
         scriviint(ss.n2); nuovalinea();
};
extern "C" void fair(s& ss);
int main()
{     s st;
      st = leggis();
      fair(st);
      scrivis(st);
      return 0;
};
```

```
// programma strutturaRif, file es11b.cpp
struct s { int n1; char c; int n2; };
extern "C" void fair(s& ss)
{               ss.n1 = ss.n1+5; ss.c = ss.c+1;
                ss.n2 = ss.n2 + 10;
}
```

Esempio 11: parametro riferimento di struttura (2)

```
# programma strutturaRif, file es11a.s
.include "servi.s"
.text
# .extern fair
leggis: …
scrivis: …

# registro per il parametro: rdi
.global main
.set st, -12
main:
      pushq %rbp              # prologo
      movq  %rsp, %rbp
      subq   $16, %rsp       # allineamento

      call     leggis
      movq  %rax, st(%rbp)
      movl   %edx, st+8(%rbp)

      leaq    st(%rbp), %rdi   # st&
      call     fair
```

```
      movq      st(%rbp), %rdi
      movl      st+8(%rbp), %esi
      call      scrivis

      movl      $0, %eax

      leave                  # epilogo
      ret
```

Esempio 11: parametro riferimento di struttura (3)

```
# programma strutturaRif, file es11b.s
# registro per il parametro: rdi
.text
.global fair
.set ss, -8
fair:
      pushq %rbp                # prologo
      movq  %rsp, %rbp
      subq   $8, %rsp
      # parametro ss (riferimento): 8 byte

      movq  %rdi, ss(%rbp)    # rdi: riferimento st&
      movq  ss(%rbp), %rdi
      addl    $5, (%rdi)
      incb    4(%rdi)
      addl    $10, 8(%rdi)

      leave                     # epilogo
      ret
```

```
# programma strutturaRif, file es11b.s
# versione semplificata
# rdi: registro per il parametro
# rdi: contiene il riferimento ss
.text
.global fair
fair:
            addl        $5, (%rdi)
            incb        4(%rdi)
            addl        $10, 8(%rdi)

            ret
```

Risultato: struttura con più di 16 byte

- **Caso di un risultato di un tipo struttura che occupa più di 16 byte (i parametri valore, se di un tipo struttura, occupano invece un numero di byte inferiore a 16).**
- **Azioni del chiamante:**
  - **risultato (struttura che occupa più di 16 byte):**
    - **utilizza per la struttura risultato una variabile di lavoro (fra le variabili locali);**
    - **tramite il registro RDI, trasmette al chiamato un primo parametro aggiuntivo costituito dall'indirizzo della variabile di lavoro.**
- **Azioni del chiamato:**
  - **effettua le dovute elaborazioni, tenendo conto che i registri per i parametri cominciano da RSI;**
  - **lascia il risultato all'indirizzo specificato dal chiamante col primo parametro aggiuntivo (contenuto in RDI);**
  - **ricopia tale indirizzo in RAX (ai fini di una verifica da parte del chiamante).**

Esempio 12: risultato struttura lunga (1)

```
// programma strutturaLunga, file
      es12a.cpp
#include"servi.cpp"
struct s { int n1; int n2; char a[10]; };
extern "C" s fstruct(int a, char c);
extern "C" void scriviris(s& ss)
{     int i;
      scriviint(ss.n1); scriviint(ss.n2);
      for (i=0; i<10; i++) scrivichar(ss.a[i]);
      nuovalinea();
}
int main()
{     s sa;
      sa = fstruct(5, 'a');
      scriviris(sa);
      return 0;
      # la struttura non viene letta,
      # ma restituita da fstruct()
};
```

```
// programma strutturaLunga, file es12b.cpp
struct s { int n1; int n2; char a[10]; };
extern "C" s fstruct(int a, char c)
{       int i; s st;
        st.n1 = a; st.n2  = 2*a;
        for (i=0; i<10; i++) st.a[i] = c+i;
        return st;
}
```

Esempio 12: risultato struttura lunga (2)

```
# programma strutturaLunga, file es12a.s
.include "servi.s"
.text
# registro per il  parametro: rdi
.set  i, -16
.set  ss, -8     # parametro riferimento
scriviris:  pushq %rbp     # prologo
        movq   %rsp, %rbp
        subq    $16, %rsp

        movq   %rdi, ss(%rbp)
        movq    ss(%rbp), %rax
        movl    (%rax), %edi            # ss.n1
        call      scriviint
        movq   ss(%rbp), %rax
        movl    4(%rax), %edi           # ss.n2
        call      scriviint
        movl    $0, i(%rbp)             # i
ciclo: cmpl   $10, i(%rbp)
        jge       finec
        movq    ss(%rbp), %rax
        movslq i(%rbp), %rcx
        movb    8(%rax,%rcx), %dil  # ss.a[i]
      call     scrivichar  #dil parte di rdi
          incl  i(%rbp)
          jmp  ciclo
finec:  call   nuovalinea
          leave                         # epilogo
          ret

 .global main
.set  sa, -48
.set  lav, -24
main:
          pushq   %rbp              # prologo
          movq    %rsp, %rbp
          subq    $48, %rsp
          leaq    lav(%rbp), %rdi # par.aggiuntivo
          movl    $5, %esi          # parametro 5 II reg
          movb    $'a', %dl         # parametro 'a' III reg
          call      fstruct
          movq    lav(%rbp), %rax # lav in sa, 18 byte
          movq    %rax, sa(%rbp)
          movq    lav+8(%rbp), %rax
          movq    %rax, sa+8(%rbp)
          movw    lav+16(%rbp), %ax
          movw    %ax, sa+16(%rbp)
          leaq    sa(%rbp), %rdi
          call      scriviris

          movl    $0, %eax
          leave                             # epilogo
          ret
```

Esempio 12: risultato struttura lunga (3)

```
# programma strutturaLunga, file es12b.s
# registri per i parametri: rdi, rsi, rdx
# di e dil appartengono  a rdi; dx e dl a rdx
.text
.global fstruct
.set  i, -40
.set  st, -32          # 18 byte, allineato a 24
.set  a, -8
.set  c, -4
fstruct:
      pushq %rbp                  # prologo
      movq  %rsp, %rbp
      subq   $40, %rsp

      movl   %esi, a(%rbp)       # II registro
      movb  %dl, c(%rbp)         # III registro

      movl   a(%rbp), %eax       # st.n1 = a
      movl   %eax, st(%rbp)

      movl    a(%rbp), %eax      # st.n2 = 2*a
      addl    a(%rbp), %eax
      movl    %eax, st+4(%rbp)
```

```
          movl     $0, i(%rbp)       # st.a[i]=c+i
ciclo:    cmpl     $10, i(%rbp)
          jge      finec
          movzbl   c(%rbp), %eax
          addl     i(%rbp), %eax
          movslq   i(%rbp), %rcx
          movb     %al, st+8(%rbp,%rcx)
          incl     i(%rbp)
          jmp      ciclo

finec:    movq     %rdi, %rax    # rdi: ind. risultato
          movq     st(%rbp), %rdi    # ritorna st
          movq     %rdi, (%rax)
          movq     st+8(%rbp), %rdi
          movq     %rdi, 8(%rax)
          movw     st+16(%rbp), %di
          movw     %di, 16(%rax)

          leave  # epilogo (ind. risultato già in rax)
          ret
```

Esempio 12: risultato struttura lunga (4)

- **Regola generale:**
  - **non è detto che il risultato di una funzione debba essere assegnato a una variabile;**
  - **esempi:**

```
scriviris(fstruct(5,'a');
sa = fstruct(5,'a') + fstruct(10,'m'); // operatore + per le strutture: ridefinito
```

- **Ottimizzazione possibile in caso di assegnamento:**
  - **trasmettere come indirizzo del risultato, invece che quello di una variabile ausiliaria *lav*, quello della variabile *sa* a cui viene assegnato il risultato.**

Esempio 13: parametro riferimento di struttura lunga (1)

```
// programma strutturaLungaRif, file
      es13a.cpp
#include"servi.cpp"
struct s { int n1; int n2; char a[10]; };
extern "C" void fstructr(s& st, int a, char c);
extern "C" void scriviris(s& ss)
{     int i;
      scriviint(ss.n1); scriviint(ss.n2);
      for (i=0; i<10; i++) scrivichar(ss.a[i]);
      nuovalinea();
}
int main()
{     s sa;
      fstructr(sa, 5, 'a');
      scriviris(sa);
      return 0;
};
```

```
// programma strutturaLungaRif, file es13b.cpp
struct s { int n1; int n2; char a[10]; };
extern "C" void fstructr(s& st, int a, char c)
{     int i;
      st.n1 = a; st.n2  = 2*a;
      for (i=0; i<10; i++) st.a[i] = c+i;
}
```

- **Con questa soluzione:**
  - **non esistono parametri aggiuntivi;**
  - **il primo parametro effettivo (riferimento) è visibile anche in C++;**
  - **in Assembler, si deve semplicemente tradurre quanto scritto in C++.**

Esempio 13: parametro riferimento di struttura lunga (2)

```
# programma strutturaLungaRif, file
      es13a.s
.include "servi.s"
.text
# registro per il  parametro: rdi
.set   i, -16
.set   ss, -8        # parametro riferimento
      (puntatore)
scriviris: pushq %rbp              # prologo
        movq       %rsp, %rbp
        subq       $16, %rsp
        movq       %rdi, ss(%rbp)
        movq       ss(%rbp), %rax
        movl       (%rax), %edi         # (*ss).n1
        call       scriviint
        movq       ss(%rbp), %rax
        movl       4(%rax), %edi        # (*ss).n2
        call       scriviint
        movl       $0, i(%rbp)                # i
ciclo: cmpl        $10, i(%rbp)
        jge        finec
                   movq       ss(%rbp), %rax
                   movslq     i(%rbp), %rcx
                   movb       8(%rax,%rcx), %dil #
(*ss).a[i]
                   call                    scrivichar
                   incl       i(%rbp)
                   jmp        ciclo
finec:  call                  nuovalinea

                   leave
                # epilogo
                   ret
```

Esempio 13: parametro riferimento di struttura lunga (3)

```
# programma strutturaLungaRif, file es13a.s, continua
.global main
.set   sa, -24
main:
        pushq       %rbp        # prologo
        movq        %rsp, %rbp
        subq        $24, %rsp          # sa: 18 byte allineati

        leaq        sa(%rbp), %rdi  # I par &lav:
        movl        $5, %esi           # II parametro 5
        movb        $'a', %dl          # III parametro 'a'
        call        fstructr
        leaq        sa(%rbp), %rdi
        call        scriviris
        movl        $0, %eax

        leave                   # epilogo
        ret
```

Esempio 13: parametro riferimento di struttura lunga (4)

```
# programma strutturaLungaRif, file es13b.s
# registri per i parametri: rdi, rsi, rdx
.text
.global fstructr
.set   i, -40
.set   st, -32
.set   a, -8
.set   c, -4
fstructr:
        pushq     %rbp              # prologo
        movq      %rsp, %rbp
        subq      $40, %rsp         # i, st, a, c

        movq      %rdi, st(%rbp)    # I par. (s*)
        movl      %esi, a(%rbp)     # II par. (int)
        movb      %dl, c(%rbp)      # III par. (char)

        movl      a(%rbp), %eax     # (*st).n1 = a
        movq      st(%rbp), %rbx
        movl      %eax, (%rbx)

                  movl      a(%rbp), %eax  #
(*st).n2 = 2*a
                  addl      a(%rbp), %eax
                  movq  st(%rbp), %rbx
                  movl      %eax, 4(%rbx)

                  movl      $0, i(%rbp)
# (*st).a[i]=c+i
ciclo: cmpl $10, i(%rbp)
                  jge       finec
                  movzbl c(%rbp), %eax
                  addl      i(%rbp), %eax
                  movslq i(%rbp), %rcx
                  movq  st(%rbp), %rbx
                  movb  %al, 8(%rbx,%rcx)
                  incl      i(%rbp)
                  jmp       ciclo

finec: leave
# epilogo
                   ret
```