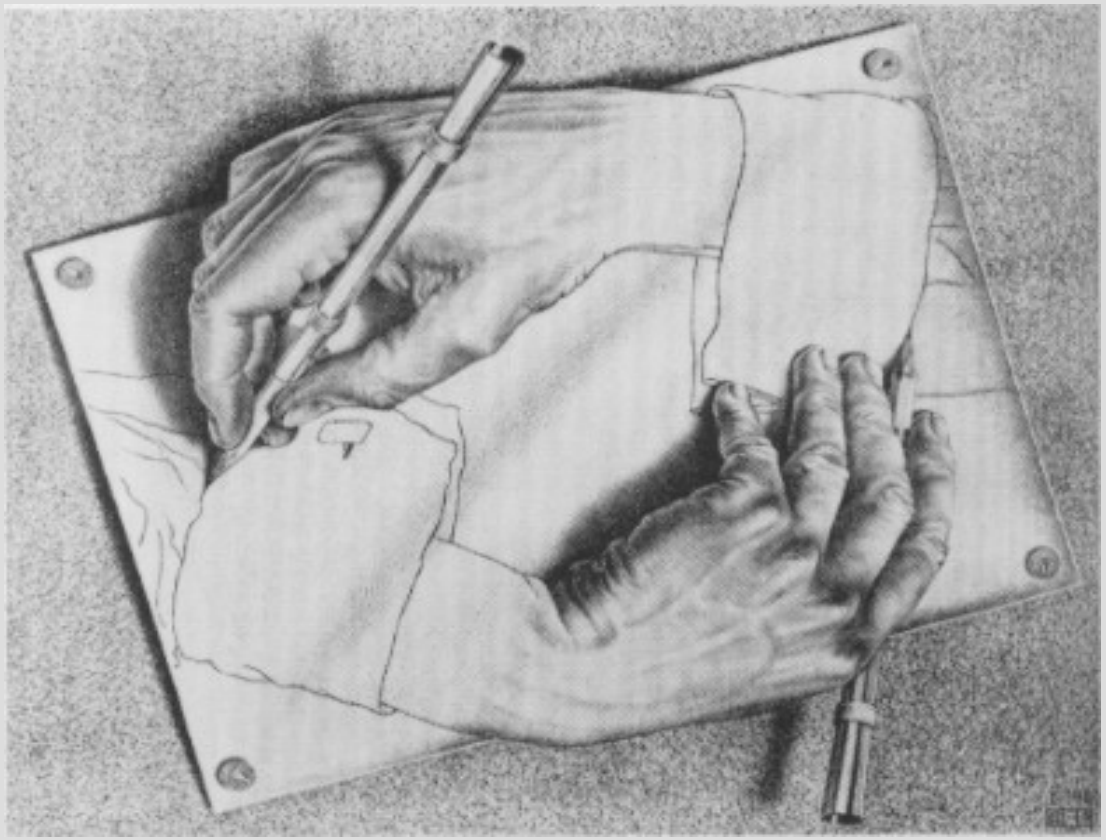
Parte 2

Ricorsione



[M.C.Escher – Drawing hands, 1948]

Funzioni ricorsive

- Una funzione si dice ***ricorsiva*** se richiama se stessa, direttamente o indirettamente
- La ricorsione si dice ***diretta*** se la funzione invoca se stessa senza intermediari
- La ricorsione si dice ***indiretta*** se coinvolge più di una funzione
  - Es: funzione uno()che invoca una funzione due() che a sua volta richiama uno()
    → uno() è stata chiamata due volte senza mai tornare al primo chiamante main()

Esempio

```
void fun(int n)
{  cout<<n<<endl ;
    if (n>0)
    fun(n-1) ;
}

int main ()
{   fun(2) ;
    return 0 ;
}
```

Funzione ricorsiva (diretta)

Che cosa fa il programma?

Programma *ric1.cc*

Soluzioni ricorsive

- La ricorsione è una ***soluzione concisa ed elegante*** per alcune tipologie di problemi (***non adatta a tutte!***)
  - Adatta a problemi complessi scomponibili in sottoproblemi più semplici
  - Utile per svolgere compiti ripetitivi su di un set di input variabili
- Alcuni problemi hanno una ***natura intrinsecamente ricorsiva***
  - Es: calcolo del fattoriale di un numero intero

Calcolo del fattoriale

$n! = n^* (n-1) * (n-2) * \ldots * 1$

$5! = 5 * 4 * 3 * 2 * 1 = 5 * 4! = 120$

$(5 - 1)!$

$n! = n * (n-1)!$

Intrinseca natura ricorsiva: si calcola il fattoriale di n calcolando il fattoriale di n-1

Pensate a come scrivere la funzione...

Funzione ricorsiva fattoriale

```
/* Legge in ingresso un numero intero n non
negativo, e ritorna il valore n!  */
int fattoriale (int n)
{
    if (n == 0)
        return 1;
    return n * fattoriale(n - 1);
}
```

Cosa succederebbe senza questa istruzione?

- Programma *fatt_ricorsivo.cc*

Condizione di terminazione

Cosa succede se scriviamo:

```
void fun(int n)
{    cout<<n<<endl;

     // if (n>0)
      fun(n-1);
}

main () {
     fun(2);
}
```

?

Catena infinita di chiamate ricorsive!
Manca la ***condizione di terminazione*** *if (n > 0)*

Condizione di terminazione

- ***Elemento necessario*** in una funzione ricorsiva
- Determina la fine della catena delle chiamate ricorsive
- Presenza di un ***caso base*** che viene eseguito quando diventa vera la condizione di terminazione
  - Può identificare semplicemente la fine della catena ricorsiva
  - Può eseguire qualche specifica operazione

Argomento di controllo

- La ricorsione è controllata, attraverso la condizione di terminazione, da un argomento della funzione (o eventualmente più di uno), detto ***argomento di controllo***
- In funzione del valore dell'argomento di controllo si sceglie se eseguire il caso base o la chiamata ricorsiva
  - Nel caso della chiamata ricorsiva, la funzione viene invocata passandole un ***nuovo valore dell'argomento di controllo***
- Funzione simile alla variabile di controllo usata per la terminazione di un ciclo iterativo

Errori frequenti

Uno degli ***errori più frequenti*** nel caso di funzioni ricorsive è quello di ***dimenticare o sbagliare la funzione di terminazione***

- La catena delle chiamate ricorsive non termina più
- Tipicamente il programma fallisce perchè i record di attivazione esauriscono lo ***stack***
  - *Prova: eseguire ric1.cc senza condizione di terminazione*

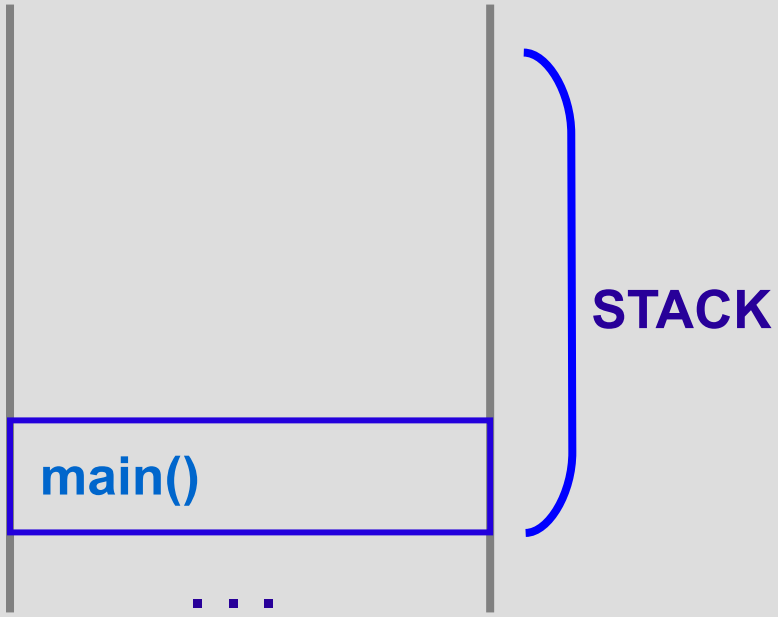
Record di attivazione

- Quando una funzione viene invocata, alcune informazioni vengono salvate sullo stack:
  - il punto del codice in cui è stata invocata (indirizzo di ritorno)
  - i parametri e le variabili locali
- L'insieme di questi dati sullo stack è detto ***record di attivazione***
- Cosa succede nel caso della ricorsione?

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



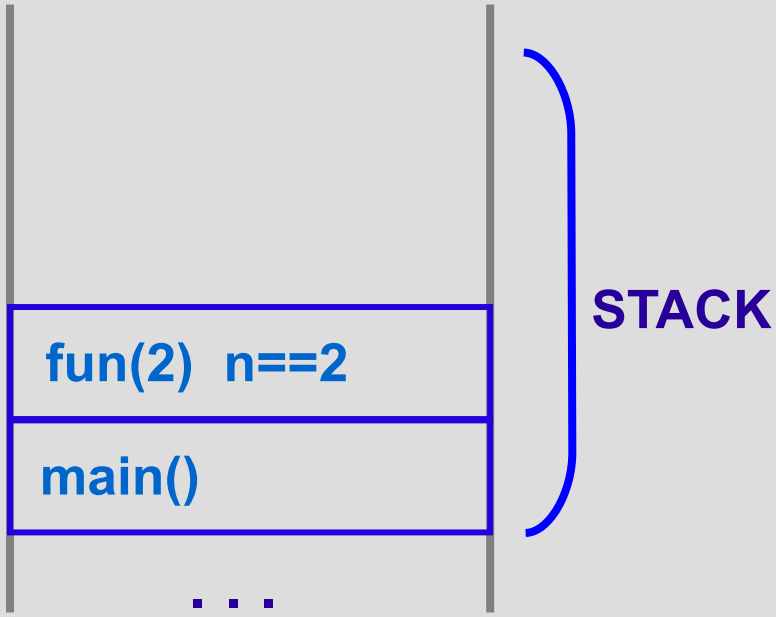
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



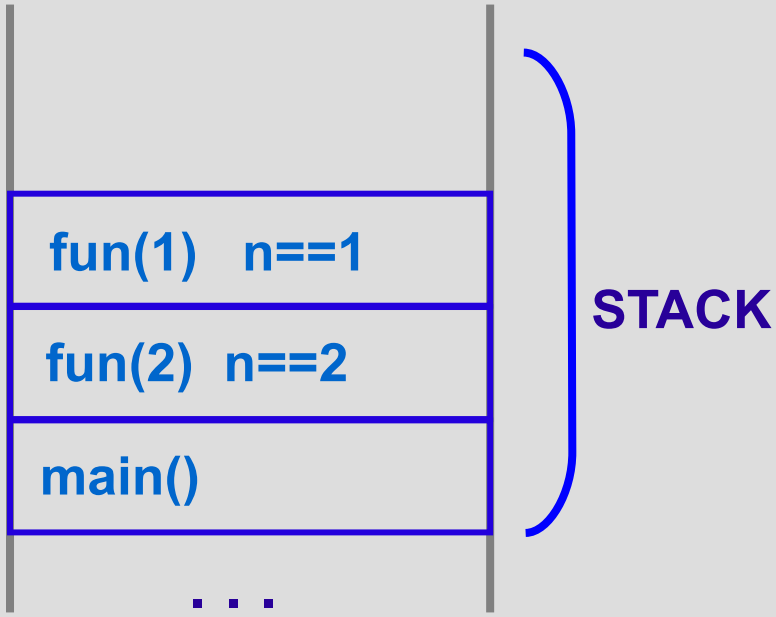
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



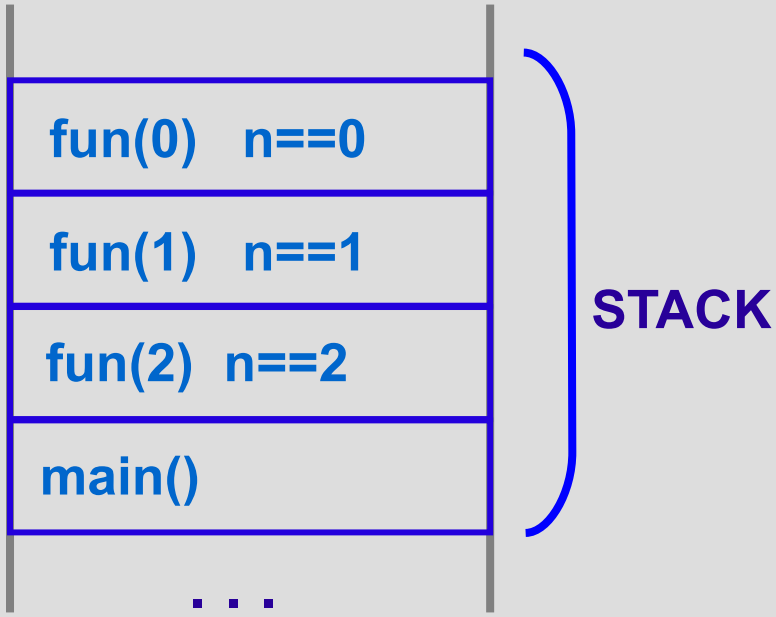
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



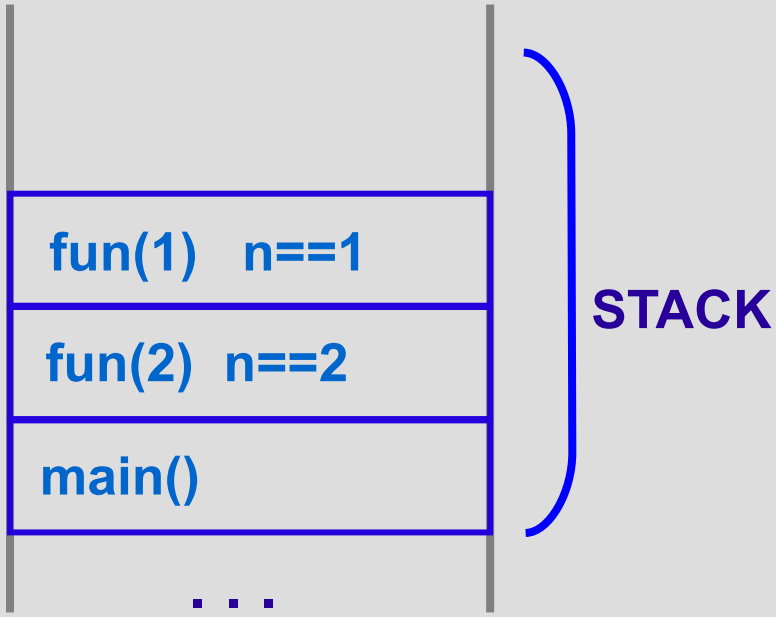
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



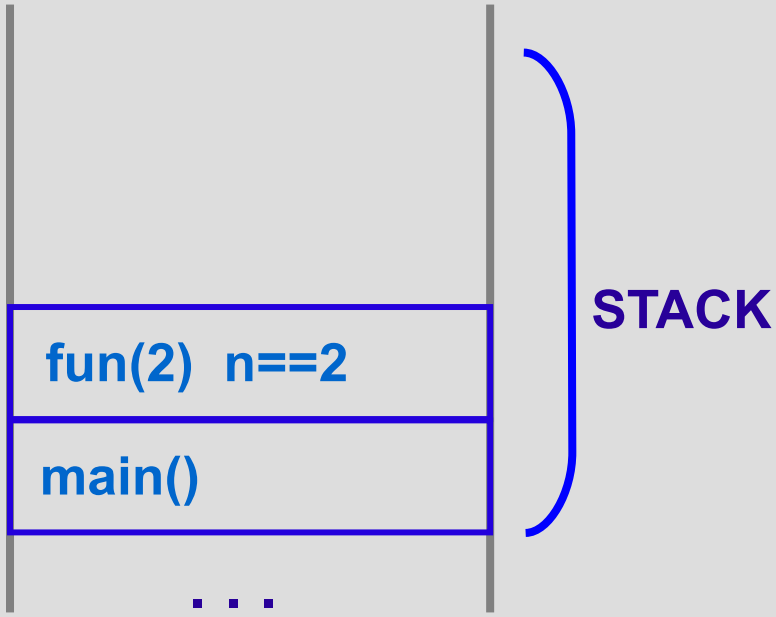
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



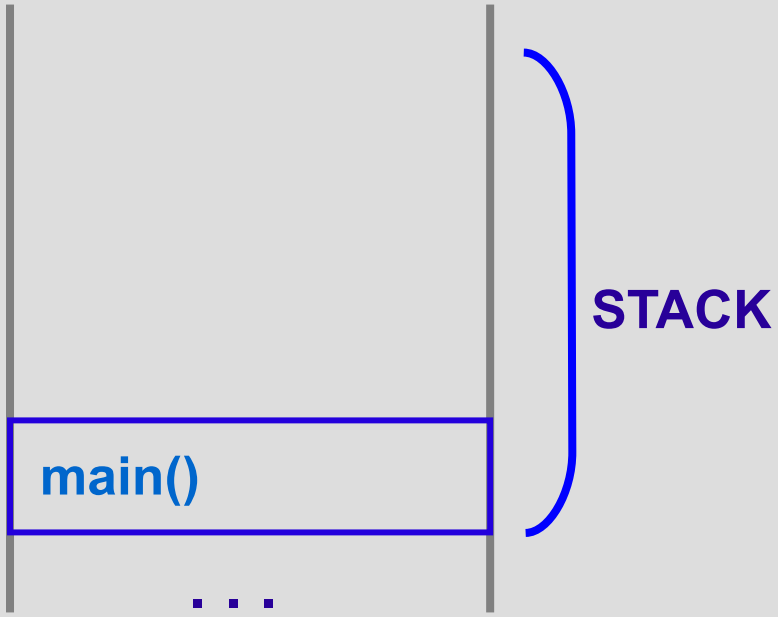
Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Esempio: fun()

```
void fun(int n)
{ cout<<n<<endl ;
  if (n>0)
     fun(n-1) ;
}

main () {
     fun(2) ;
}
```



Sequenza di attivazioni:

(S.O.)->main()->fun(2)->fun(1)->fun(0)

Istanze e variabili locali

- Quando una funzione viene invocata, viene creata una ***istanza*** della funzione
- Ogni istanza di una funzione ha una propria copia delle variabili locali e dei parametri formali (nel ***record di attivazione***)
- Nel caso delle funzioni ricorsive possono esistere più istanze della stessa funzione nello stesso istante (***istanze multiple***)
  - Esempio precedente: esiste una copia con ***diverso valore di n in ogni istanza di fun()***

Possibile forma della funzione

```
forma fun_ricorsiva (...)
{
blocco_istruzioni1;
... = fun_ricorsiva(...);  //se non
                           //caso base
blocco_istruzioni2;
}
```

Programmi

- Stampa tutti i numeri da n a 1, con n inserito da utente
  - *Stampa_ric_decresc.cc*
- Stampa tutti i numeri da 1 ad n, con n inserito da utente
  - *Stampa_ric_cresc.cc*

Ricorsione e scomposizione

algoritmo_ric(...) {
operazioni da eseguire prima ;
// eseguite man mano che la catena di
// invocazioni ricorsive innestate viene costruita

(a meno che non si sia nel caso base) esecuzione ricorsiva algoritmo_ric su sotto-problema ;

operazioni da eseguire dopo ;
// eseguite, in ordine inverso, man mano che la
// catena di invocazioni ricorsive innestate
// viene distrutta
}

Potenza ricorsiva

- Programma *potenza_ric.cc*
- Calcolo ricorsivo della potenza di un numero reale elevato ad esponente intero, positivo o nullo

Potenza ricorsiva: suggerimenti

- Si può scrivere a^3 in funzione di a^2?

  a^3=a*a^2

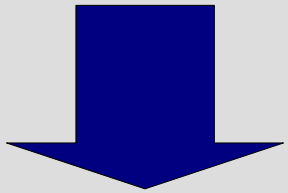
- In generale:

  a^n=a*a^(n-1)

- C'è qualcosa da fare prima della chiamata ricorsiva?
- Qual'è la condizione di terminazione corretta?

Iterazione vs. ricorsione

- Entrambe implicano la ripetizione e si basano su un test di controllo
- ***Ricorsione*** → concisa ed elegante per problemi scomponibili
- ***Iterazione*** → meno concisa per lo stesso tipo di problemi, ma più efficiente nello spazio e nel tempo... *Perchè?*



Non è necessario allocare spazio in memoria per un nuovo record di attivazione ad ogni chiamata ricorsiva

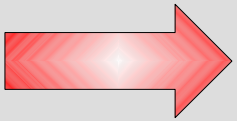
Domanda

- Alla luce della considerazioni su iterazione e ricorsione, pensate che la soluzione ricorsiva sia la migliore per risolvere i problemi di calcolo di fattoriale e potenza di un numero n?

NO!

Iterazione vs. ricorsione

- Alcuni problemi hanno un'implementazione ricorsiva naturale e intuitiva



semplificano il lavoro del programmatore sia nella scrittura che nella manutenzione

- Esempi:
  - Ricerca binaria ricorsiva in un vettore ordinato
  - Algoritmi di ordinamento
  - Visita di grafi ad albero

Non limitati ad un numero conosciuto di iterazioni!

Scomposizione nello spazio

- La ricorsione è un efficace schema di risoluzione per problemi scomponibili
- Ad esempio, un algoritmo che lavora su un array potrebbe ottenere il risultato finale desiderato lavorando su dei sotto-array

- Programma *trova_max_ricorsivo.cc*

Mutua ricorsione

- Due funzioni si dicono ***mutuamente ricorsive*** se una invoca l'altra e viceversa
- Esempio:

```
int is_even(unsigned int n) {
if (n==0) return 1;
else return(is_odd(n-1));
}
int is_odd(unsigned int n) {
return (!is_even(n));
}
```

NOTA: una prima funzione ne richiama una seconda che è definita in termini della prima

Ricorsione: considerazioni

Buone norme di programmazione:

- Verificare la correttezza della condizione di terminazione per evitare ricorsioni infinite
- Tenere sotto controllo lo stack
  - Eventuale uso di contatori di sicurezza
- Usare la ricorsione solo per problemi adatti
- Limitare la ricorsione ad una funzione
  - Ricorsioni mutue sono pericolose in quanto difficili da gestire e/o rilevare