IEIM 2019-2020

Esercitazione VIII

"Ricorsione"

Alessandro A. Nacci
alessandro.nacci@polimi.it - www.alessandronacci.it

RICORSIONE

Divide et impera

- Metodo di approccio ai problemi che consiste nel dividere il problema dato in problemi più semplici
- I risultati ottenuti risolvendo i problemi più semplici vengono combinati insieme per costituire la soluzione del problema originale
- Generalmente, quando la semplificazione del problema consiste essenzialmente nella semplificazione dei DATI da elaborare (ad es. la riduzione della dimensione del vettore

La ricorsione (I)

- Una funzione è detta **ricorsiva** se chiama se stessa
- Se due funzioni si chiamano l'un l'altra, sono dette **mutuamente ricorsive**
- La funzione ricorsiva sa risolvere direttamente solo casi particolari di un problema detti **casi di base**: se viene invocata passandole dei dati che costituiscono uno dei casi di base, allora restituisce un risultato
- Se invece viene chiamata passandole dei dati che NON costituiscono uno dei casi di base, allora chiama se stessa (passo ricorsivo) passando dei DATI semplificati/ridotti

La ricorsione (II)

- Ad ogni chiamata si semplificano/riducono i dati, così ad un certo punto si arriva ad uno dei casi di base
- Quando la funzione chiama se stessa, sospende la sua esecuzione per eseguire la nuova chiamata
- L'esecuzione riprende quando la chiamata interna a se stessa termina
- La sequenza di chiamate ricorsive termina quando quella più interna (annidata) incontra uno dei casi di base
- Ogni chiamata alloca sullo stack (in stack frame diversi) nuove istanze dei parametri e delle variabili locali (non static)

Esempio: il fattoriale

Funzione ricorsiva che calcola il fattoriale di un numero n

Premessa (definizione ricorsiva):

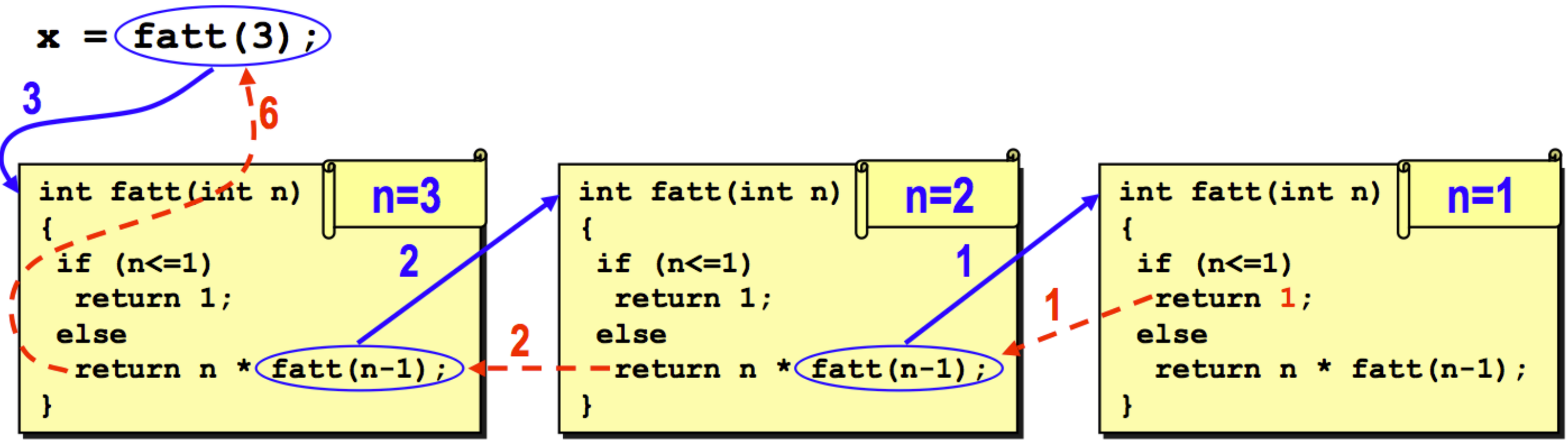
se $n \leq 1$ → $n! = 1$

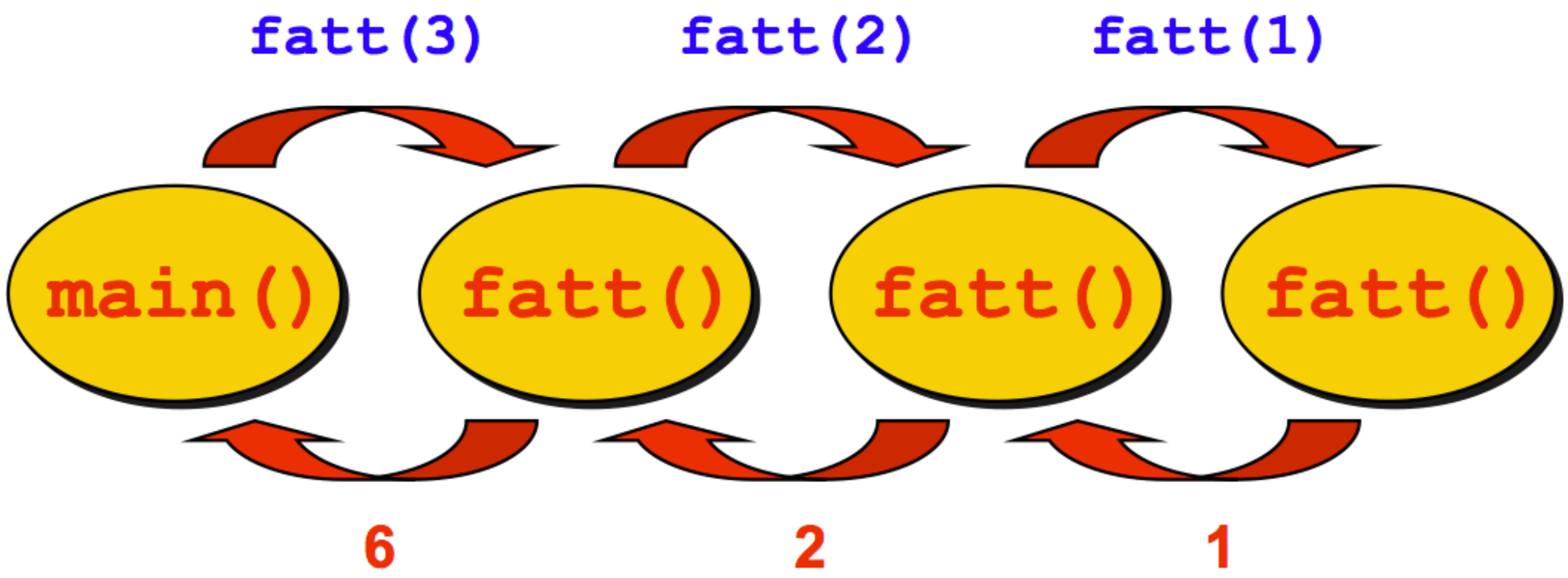
se $n > 1$ → $n! = n * (n-1)!$

```
int fatt(int n)
{
   if (n<=1)
      return 1;   → Caso di base
   else
      return n * fatt(n-1);
}
```

Semplificazione dei dati del problema

Esempio: il fattoriale - passo per passo





Ricorsione - Esercizio 1

- Scrivere una funziona ricorsiva che calcoli ricorsivamente la somma di tutti i numeri compresi tra 0 ed x
- Il prototipo della funzione è: int ric(int x)

```
int ric(int x) {



}
```

Ricorsione - Esercizio 1

- Scrivere una funziona ricorsiva che calcoli ricorsivamente la somma di tutti i numeri compresi tra 0 ed x
- Il prototipo della funzione è: int ric(int x)

```
int ric(int x) {
  if (x == 0)
      return 0;
  else
      return x + ric(x-1);
}
```

Ricorsione - Esercizio 11

- Scrivere le versioni ricorsiva ed iterativa di una funziona che calcoli il seguente valore

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

```
double f(double a, int n)
{

}
```

Ricorsione - Esercizio 11

- Scrivere le versioni ricorsiva ed iterativa di una funziona che calcoli il seguente valore

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

```
double f(double a, int n)
{

}
```

Ricorsione - Esercizio 11

- Scrivere le versioni ricorsiva ed iterativa di una funziona che calcoli il seguente valore

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

```
double f(double a, int n)
{ if (n==1) return a - 1/a;
  else return a - n/a + f(a, n-1);
}
```

Ricorsione - Esercizio 11

- Scrivere le versioni ricorsiva ed iterativa di una funziona che calcoli il seguente valore

$$\sum_{i=1}^{n}\left(a-\frac{i}{a}\right)$$

```
double f(double a, int n)
{ if (n==1) return a - 1/a;
  else return a - n/a + f(a, n-1);
}
```

```
double f(double a, int n)
{ int i=1;
  double sum=0;
  while(i<=n)
       {sum = sum + a - i/a;
        i++;}
  return  sum;
}
```

Qualche considerazione

- L'apertura delle chiamate ricorsive semplifica il problema, ma non calcola ancora nulla
- Il valore restituito dalle funzioni viene utilizzato per calcolare il valore finale man mano che si chiudono le chiamate ricorsive: ogni chiamata genera valori intermedi a partire dalla fine
- Nella ricorsione vera e propria non c'è un mero passaggio di un risultato calcolato nella chiamata più interna a quelle più esterne, ossia le return non si limitano a passare indietro invariato un valore, ma c'è un'elaborazione intermedia

Quando utilizzare la ricorsione

- PRO
  Spesso la ricorsione permette di risolvere un problema anche molto complesso con poche linee di codice
- CONTRO
  La *ricorsione è poco efficiente* perché richiama molte volte una funzione e questo:
  - richiede tempo per la gestione dello stack (allocare e passare i parametri, salvare l'indirizzo di ritorno, e i valori di alcuni registri della CPU)
  - consuma molta memoria (alloca un nuovo stack frame ad ogni chiamata, definendo una nuova ulteriore istanza delle variabili locali non `static` e dei parametri ogni volta)

Quando utilizzare la ricorsione

- CONSIDERAZIONE
  Qualsiasi problema ricorsivo può essere risolto in modo non ricorsivo (ossia iterativo), ma la soluzione iterativa potrebbe non essere facile da individuare oppure essere molto più complessa

- CONCLUSIONE
  Quando non ci sono particolari problemi di efficienza e/o memoria, l'approccio ricorsivo è in genere da preferire se:
  - è più intuitivo di quello iterativo
  - la soluzione iterativa non è evidente o agevole

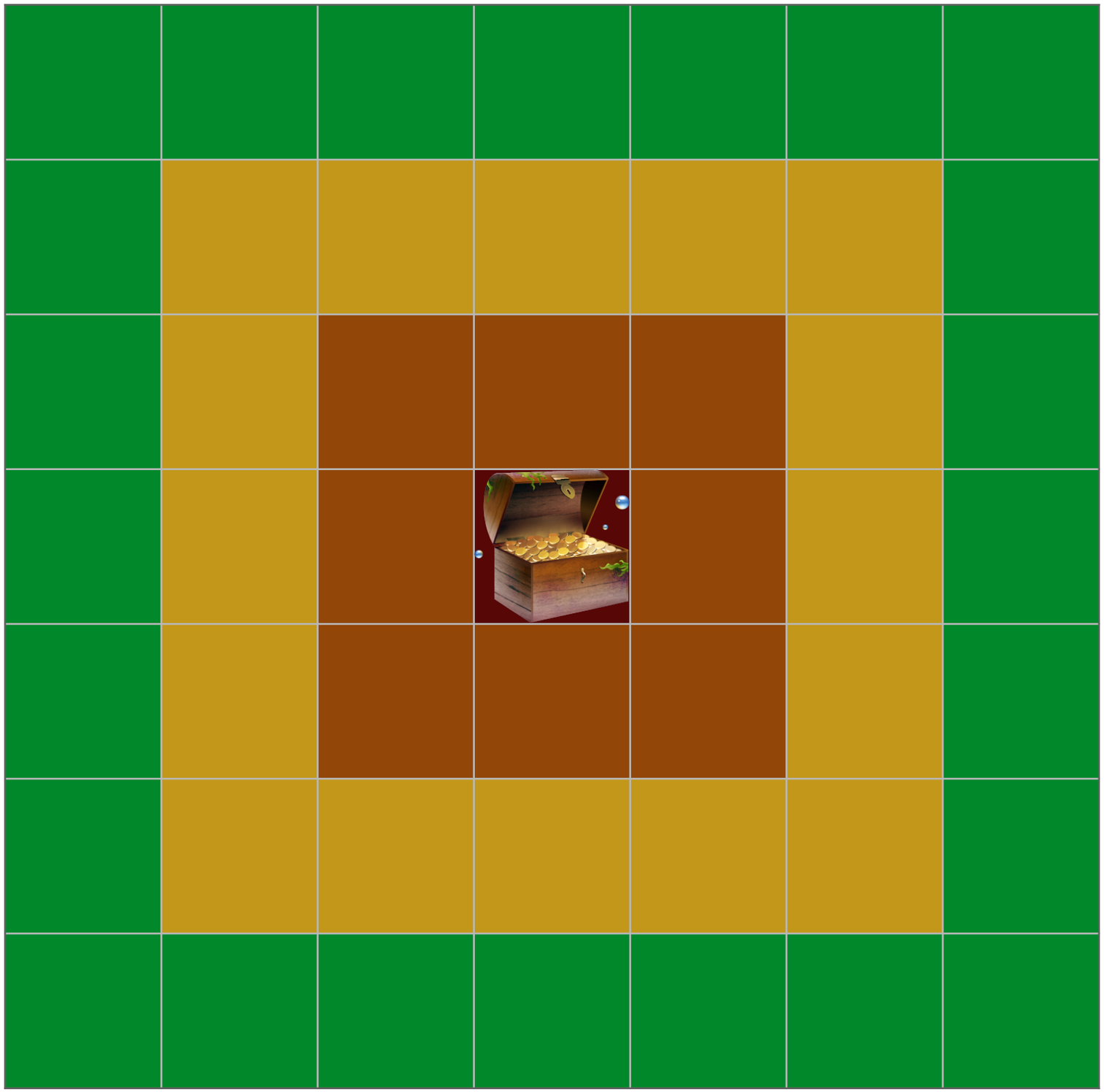
Mappa del Tesoro

Matrici e Ricorsione

LA MAPPA DEL TESORO

Una mappa particolare!

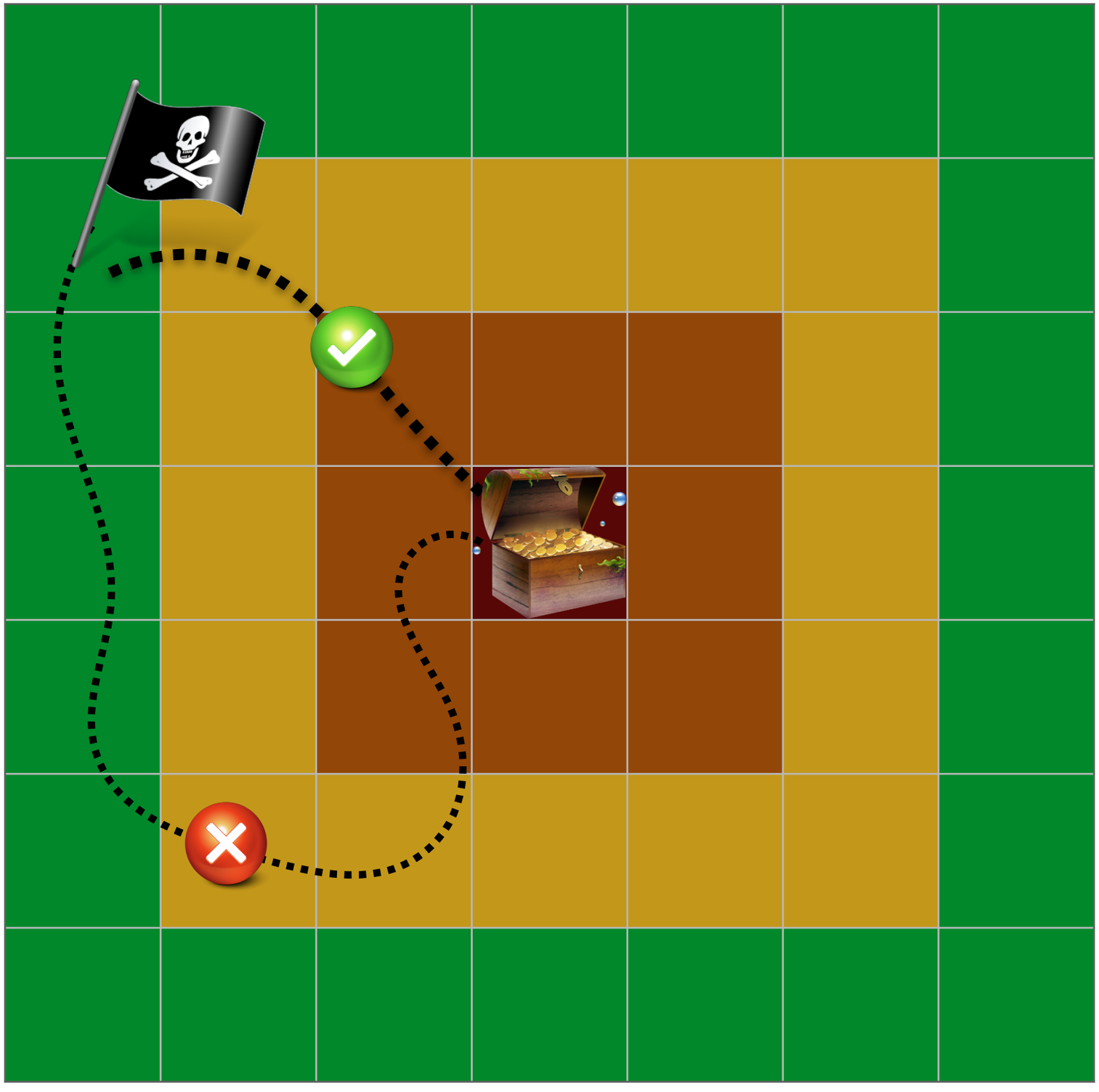
- Immaginiamo di avere una mappa del tesoro… un po particolare!



- E’ una sorta di mappa termica **radiale** della “vicinanza” al tesoro

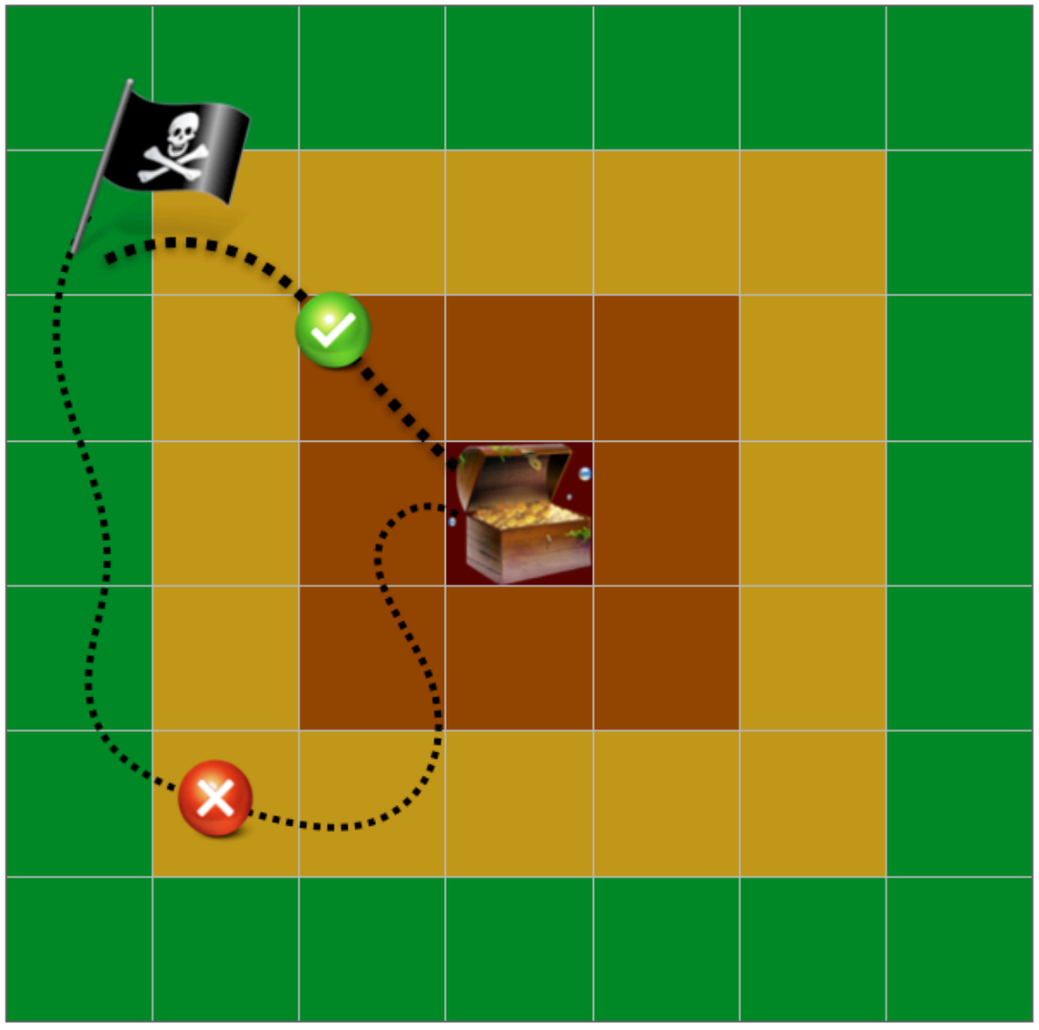
La strategia del pirata

- Un pirata che vuole muoversi su questa mappa per cercare il tesoro, si sposterà **sempre** dalle celle con colori più freddi verso celle con colori più caldi



Traccia dell'esercizio

- Scrivere un programma C che sia in grado di
  - gestire la mappa appena introdotta
  - date le coordinate del tesoro, creare la *mappa termica di vicinanza*
  - date delle coordinate di partenza, tracciare un percorso valido per arrivare al tesoro



Gestione della mappa termica

- Come rappresentiamo la mappa termica?

Gestione della mappa termica

- Come rappresentiamo la mappa termica?
  - Usiamo una matrice
  - I colori diventano numeri

Gestione della mappa termica

- Come rappresentiamo la mappa termica?
  - Usiamo una matrice
  - I colori diventano numeri

11	11	11	11	11	11	11	11	11
11	12	12	12	12	12	12	12	11
11	12	13	13	13	13	13	12	11
11	12	13	14	14	14	13	12	11
11	12	13	14	15	14	13	12	11
11	12	13	14	14	14	13	12	11
11	12	13	13	13	13	13	12	11
11	12	12	12	12	12	12	12	11
11	11	11	11	11	11	11	11	11

Include, define e dichiarazione variabili

```c
#include <stdio.h>
#include <math.h>

#define W 10
#define H 10

#define VALORE_TESORO 15
```

VALORE_TESORO è legato alla dimensione della mappa

```c
int main()
{
    int mappa[W][H];
    int x,y;
    int trovato = 0;
```

Inizializziamo la mappa

```
void init_mappa(int mappa[W][H])
{
```

Scrivere una funziona che riempia di '0' una matrice 'mappa' in ingresso.

```
}
```

Inizializziamo la mappa

```
void init_mappa(int mappa[W][H])
{

      int x,y;

    for (x = 0; x < W; x++)
        {
            for (y = 0; y < H; y++)
            {
                mappa[x][y] = 0;
            }
        }

}
```

Stampiamo la mappa

```
void stampa_mappa(int mappa[W][H])
{

}
```

Scrivere una funziona che visualizzi a schermo una matrice 'mappa' in ingresso

Stampiamo la mappa

```
void stampa_mappa(int mappa[W][H])
{
     int x,y;

     for (x = 0; x < W; x++)
     {
          for (y = 0; y < H; y++)
          {
               printf("%d\t", mappa[x][y]);
          }
          printf("\n");
     }

     printf("\n\n\n");

}
```

Metti tesoro - corretto

```
void metti_tesoro(int mappa[W][H], int tes_x, int tes_y)
{

}
```

Scrivere una funziona che, date due coordinate tes_x e tes_y - che sono le coordinate del tesoro - crei la mappa termica radiale presentata in precedenza all'interno di una matrice 'mappa' passata in ingresso.

Metti tesoro - corretto

```
void metti_tesoro(int mappa[W][H], int tes_x, int tes_y)
{

    int radius;
    int x,y;
    int start_x, start_y;
    int end_x, end_y;

    int val_tesoro = VALORE_TESORO;

    mappa[tes_x][tes_y] = val_tesoro;

    // attenzione alla dimensione massima del raggio!
    for (radius = 1; radius < W; radius++)
    {
        start_x = tes_x - radius;
        start_y = tes_y - radius;

        end_x = tes_x + radius;
        end_y = tes_y + radius;

        for (x = start_x; x <= end_x; x++)
            for (y = start_y; y <= end_y; y++)
                if ( ((x == start_x) || (x == end_x)) || ((y == start_y) || (y == end_y)))
                    if (x >= 0 && y >= 0 && x < W && y < H )
                        mappa[x][y] = val_tesoro - radius;
    }
}
```

Cerchiamo il tesoro e tracciamo il percorso…

```
int cerca_tesoro(int mappa[W][H], int start_x, int start_y)
{
```

Scrivere una funzione C che date due coordinate da cui il pirata parte ('start_x' e 'start_y'), cerchi un percorso ***corretto*** *per arrivare al tesoro.*

Per creare un percorso corretto, dato un punto generico in cui il pirata si trova, il pirata può spostarsi solo in una cella adiacente che abbia un valore superiore a quello della cella corrente.

Una volta che una cella viene visitata, è necessario 'marcarla' per indicare che quella cella è parte del percorso scelto.

```
}
```

Cerchiamo il tesoro e tracciamo il percorso...

```
int cerca_tesoro(int mappa[W][H], int start_x, int start_y)
{
    if (mappa[start_x][start_y] == VALORE_TESORO) return 1;

    int x,y;

    for (x = start_x - 1; x <= start_x + 1; x++)
        for (y = start_y - 1; y <= start_y + 1; y++)
            if (x>0 && y >= 0 && x<W && y<H)
                if (mappa[x][y] > mappa[start_x][start_y])
                {
                    mappa[start_x][start_y] = -1;
                    return cerca_tesoro(mappa, x, y);
                }

    return 0;
}
```

Mostriamo a schermo il percorso...

```
void stampa_percorso(int mappa[W][H])
{
```

Scrivere una cella che visualizzi in modo chiaro quale è il percorso scelto dal pirata per raggiungere il tesoro.

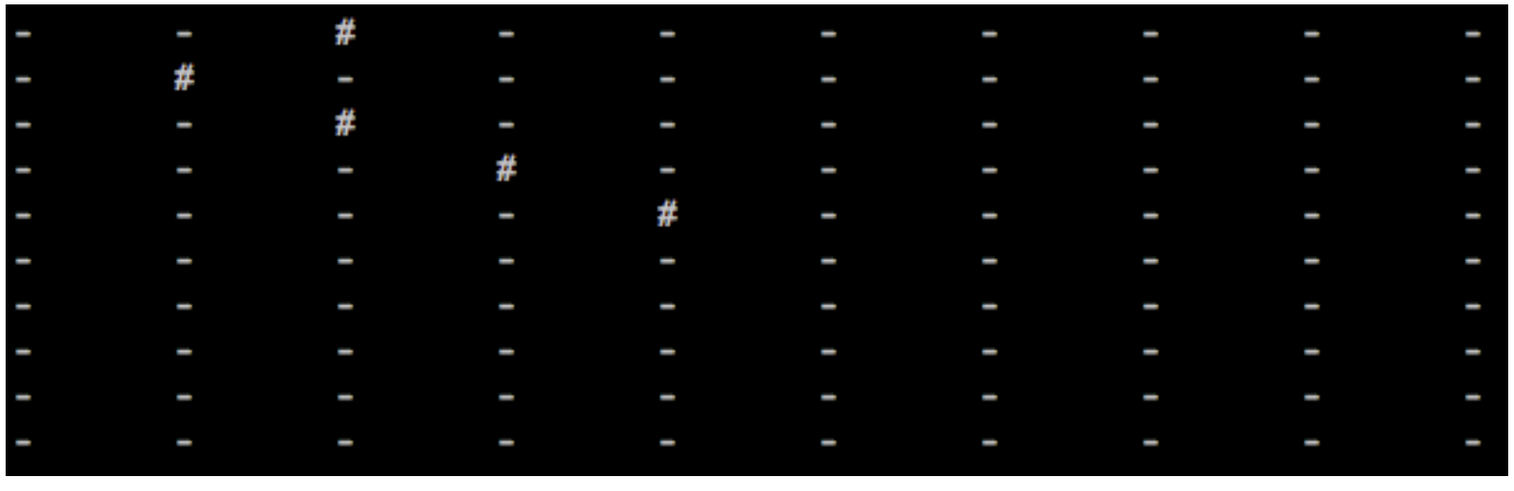
```
}
```

Mostriamo a schermo il percorso...

```
void stampa_percorso(int mappa[W][H])
{
     int x,y;

     for (x = 0; x < W; x++)
     {
          for (y = 0; y < H; y++)
          {
               if (mappa[x][y] == -1) printf("#\t");
               else printf("-\t");
          }
          printf("\n");
     }

     printf("\n\n\n");

}
```

E facciamo il main :)

```
int main()
{
    int mappa[W][H];
    int x,y;
    int trovato = 0;

    init_mappa(mappa);
    stampa_mappa(mappa);

    metti_tesoro(mappa, 5,5);
    stampa_mappa(mappa);

    trovato = cerca_tesoro(mappa, 0,2);

    if (trovato) printf("Ho trovato il tesoro!\n");

    stampa_mappa(mappa);
    stampa_percorso(mappa);

}
```



Tutte il materiale sarà disponibile sul mio sito internet!

alessandronacci.it