



IEIM 2017-2018

Esercitazione IX ***“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):

$$\begin{cases} \text{se } n \leq 1 \rightarrow n! = 1 \\ \text{se } n > 1 \rightarrow n! = n * (n-1)! \end{cases}$$

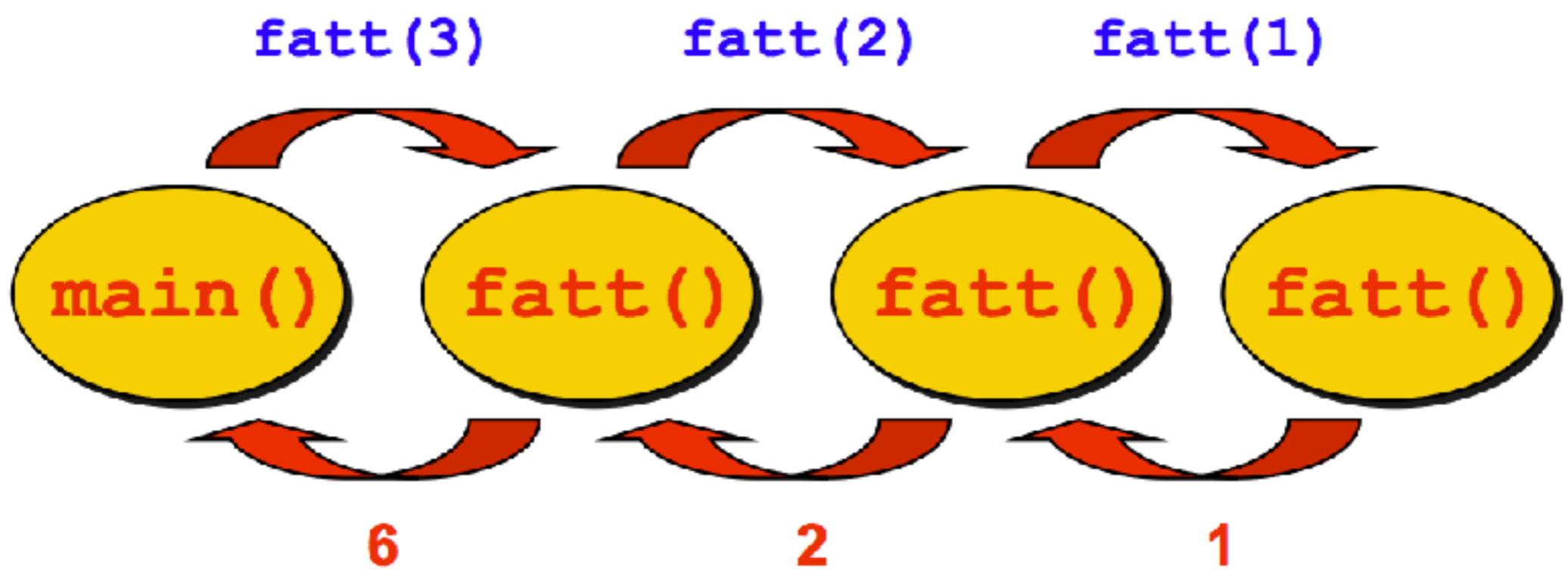
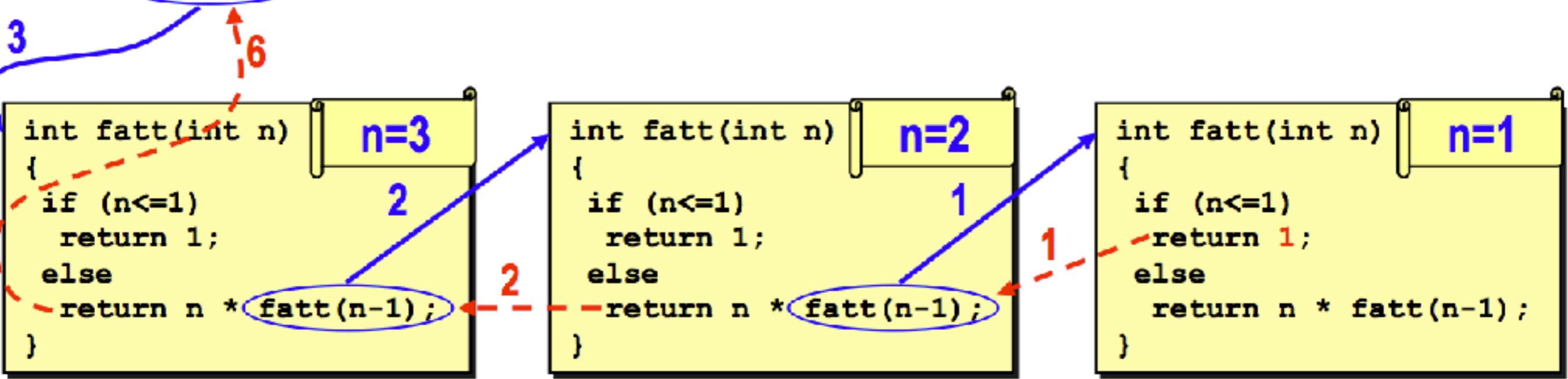
```
int fatt(int n)
{
    if (n<=1)
        return 1; → Caso di base
    else
        return n * fatt(n-1);
}
```

Semplificazione
dei dati del
problema



Esempio: il fattoriale - passo per passo

x = fatt(3);





Ricorsione - Esercizio I

- Scrivere una funzione 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 II

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

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

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



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



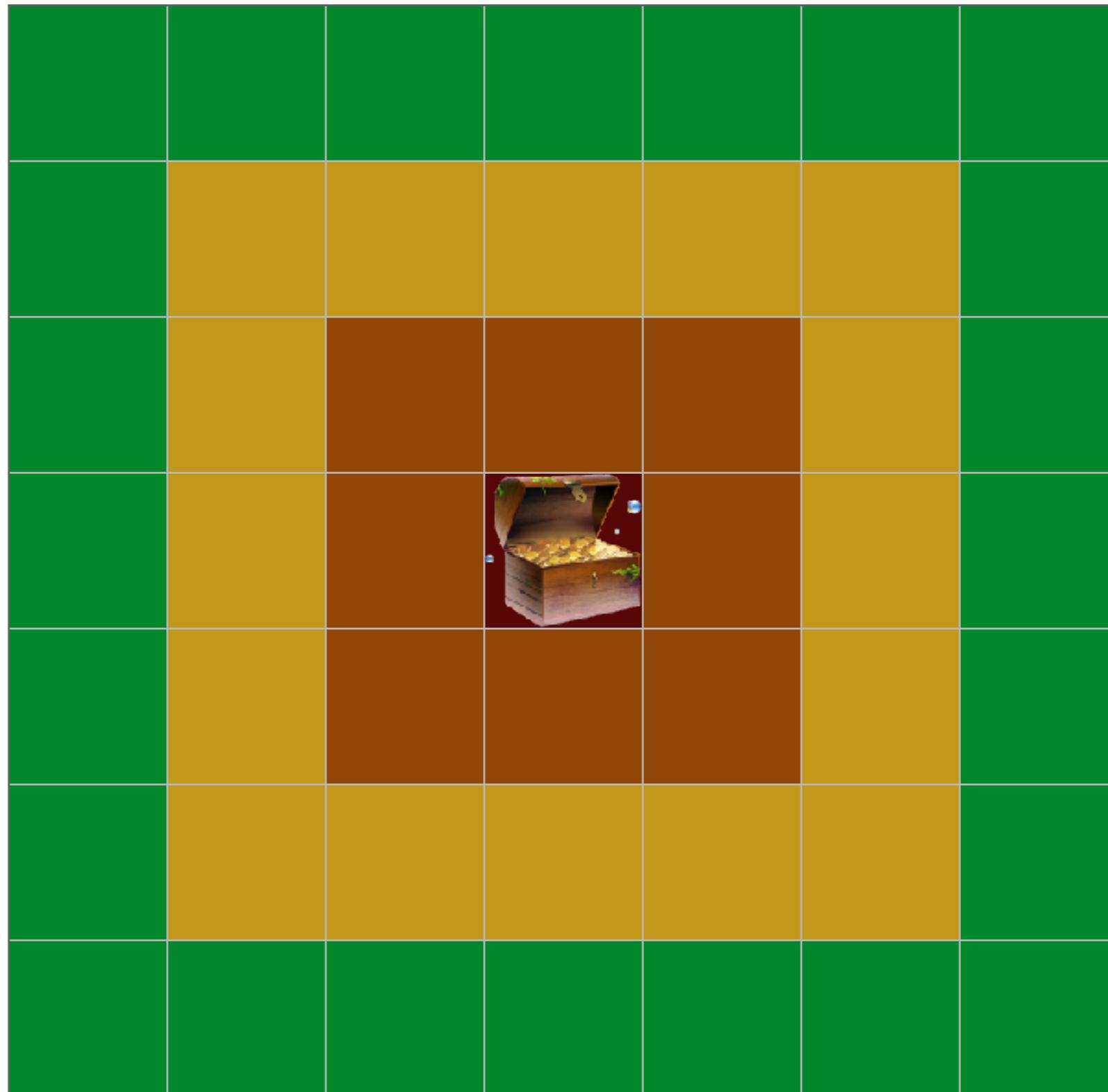
Mappa del Tesoro

Matrici e Ricorsione

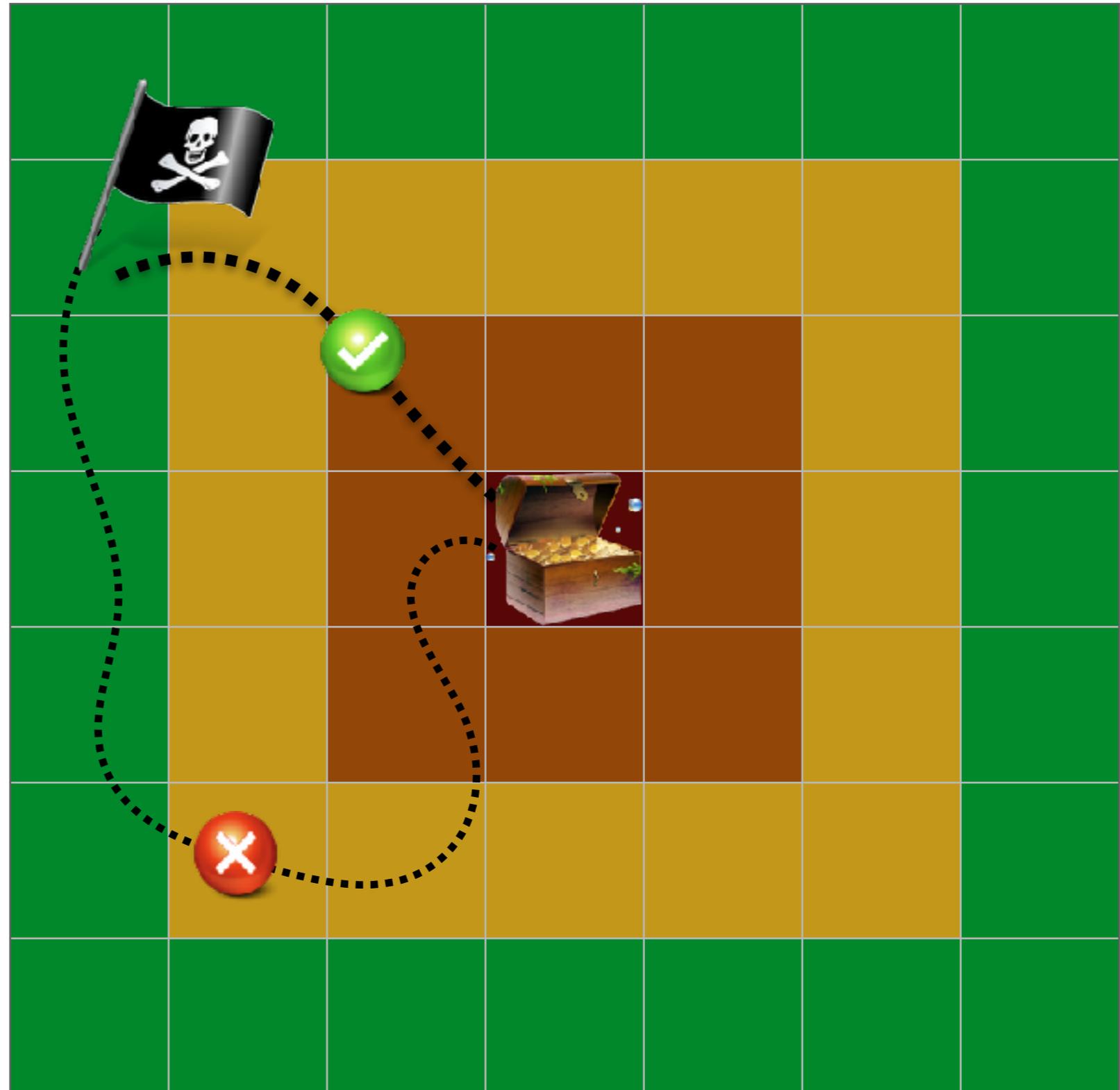
LA MAPPA DEL TESORO



- Immaginiamo di avere una mappa del tesoro... un po particolare!
- E' una sorta di mappa termica **radiale** della “vicinanza” al tesoro



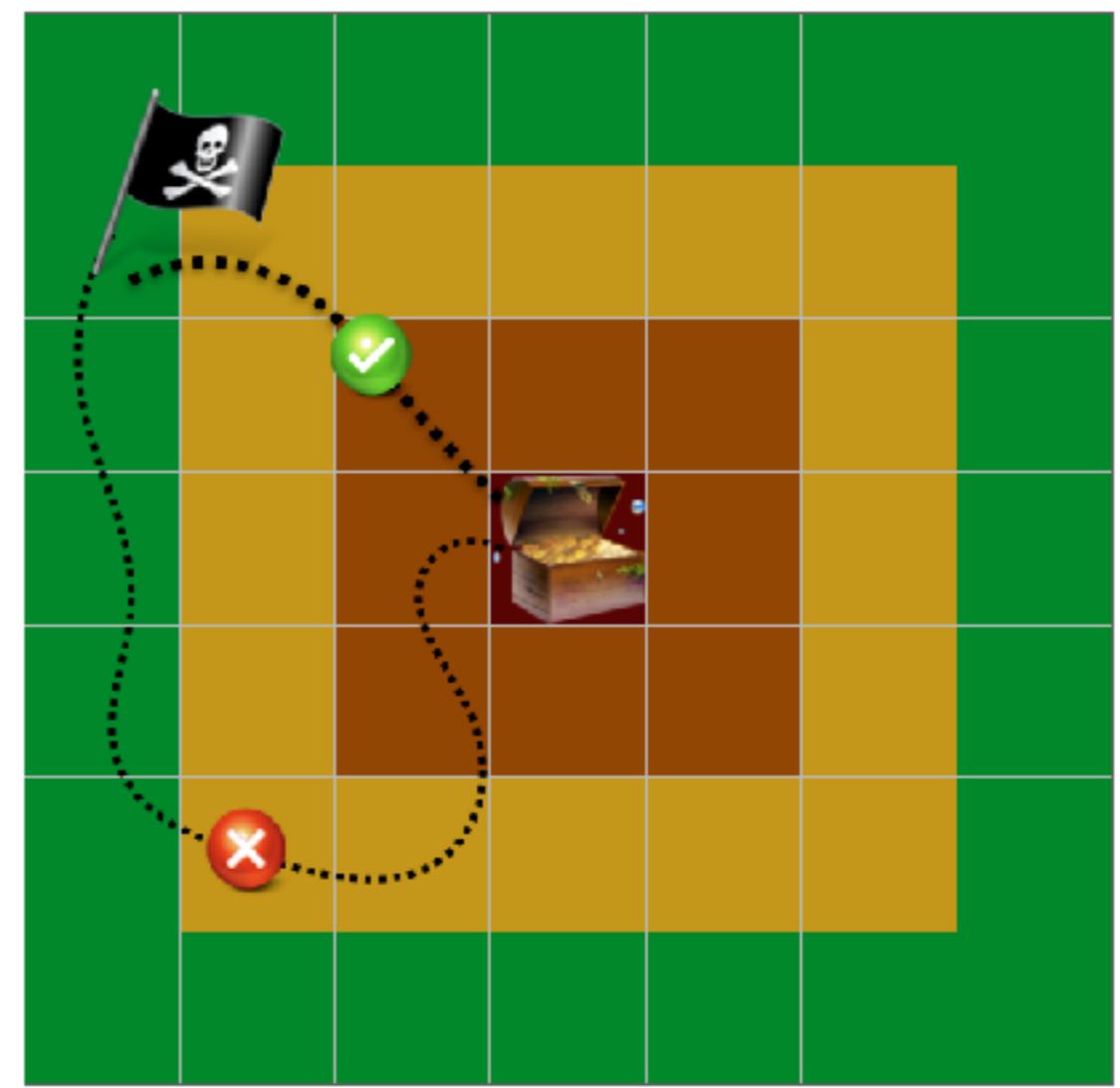
- Un pirata che vuole muoversi su questa mappa per cercare il tesoro, si sposterà **sempre** dalle celle con colori più freddi verso celle con colori più caldi





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?
 - Usiamo una matrice
 - I colori diventano numeri

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



Include, define e dichiarazione variabili

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

```
#define W 10
#define H 10
```

```
#define VALORE_TESORO 15
```

**VALORE_TESORO è legato alla
dimensione della mappa**



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



Inizializziamo la mappa

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

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

```
}
```



Stampiamo la mappa

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

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

```
}
```



Metti tesoro - corretto

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

Scrivere una funzione 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.

```
}
```



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.

```
}
```



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.

```
}
```



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);
}
```

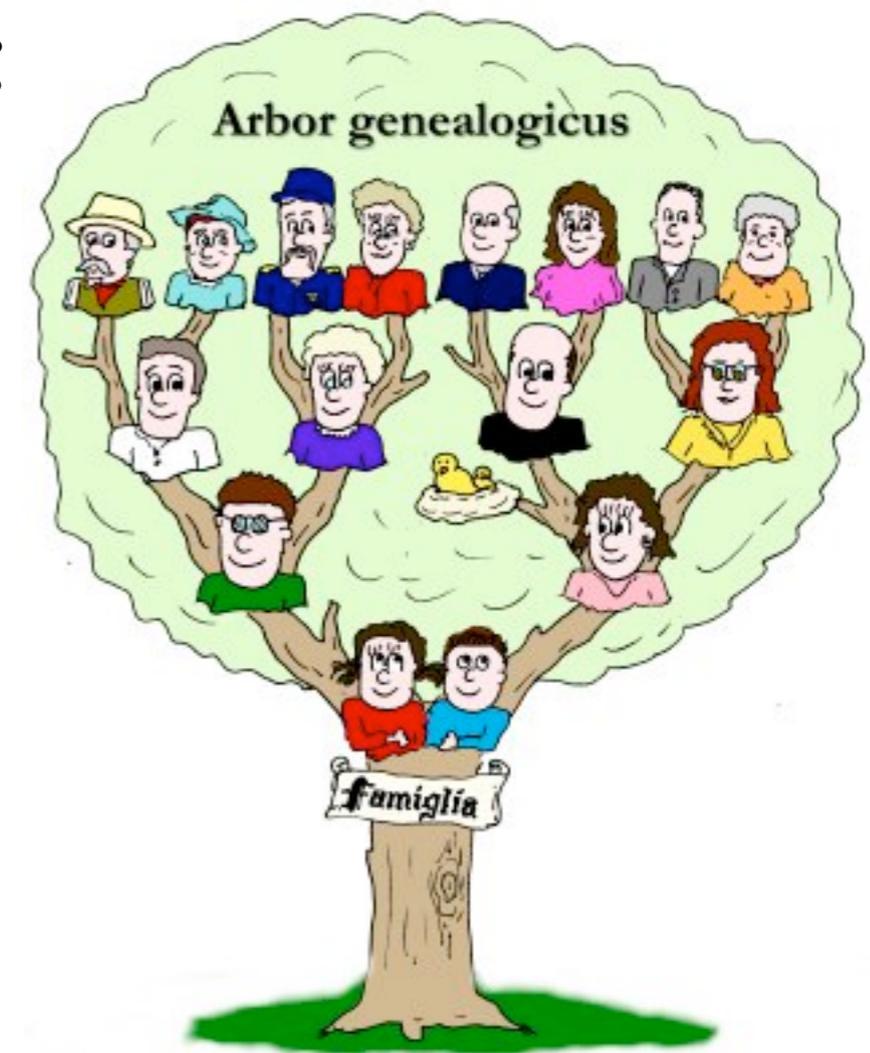
```
- # - - - - -
- # - - - - -
- # - - - - -
- - # - - - - -
- - - # - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
- - - - - - - - -
```



L'albero genealogico

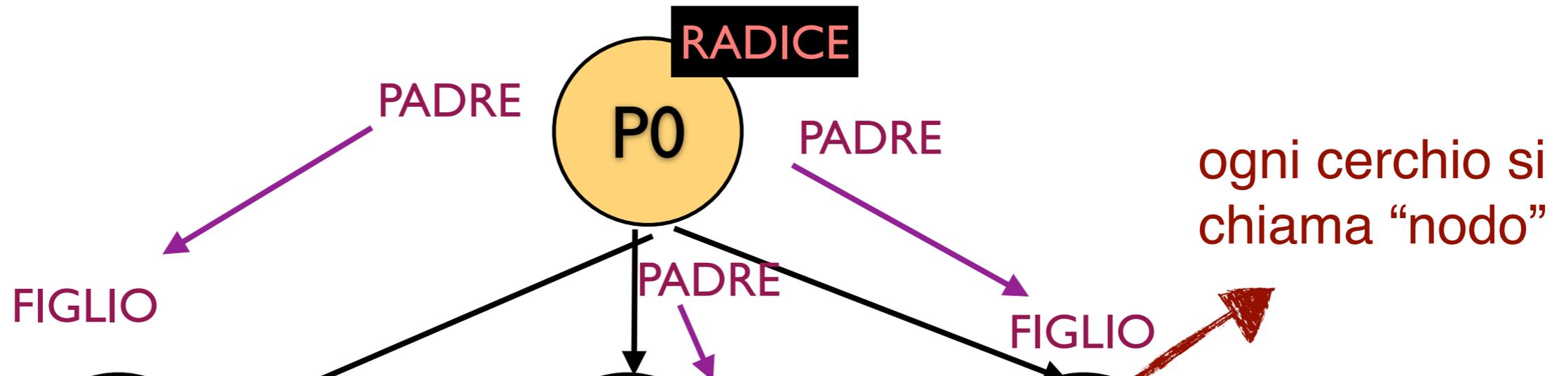
Esercizio 2

- Scrivere un programma C che sia in grado di rappresentare e gestire un albero genealogico
- In particolare, vogliamo poter fare:
 - Creare una persona
 - Rappresentare di una popolazione
 - Aggiungere figli ad una persona
 - Elencare i figli e i nipoti dato un antenato

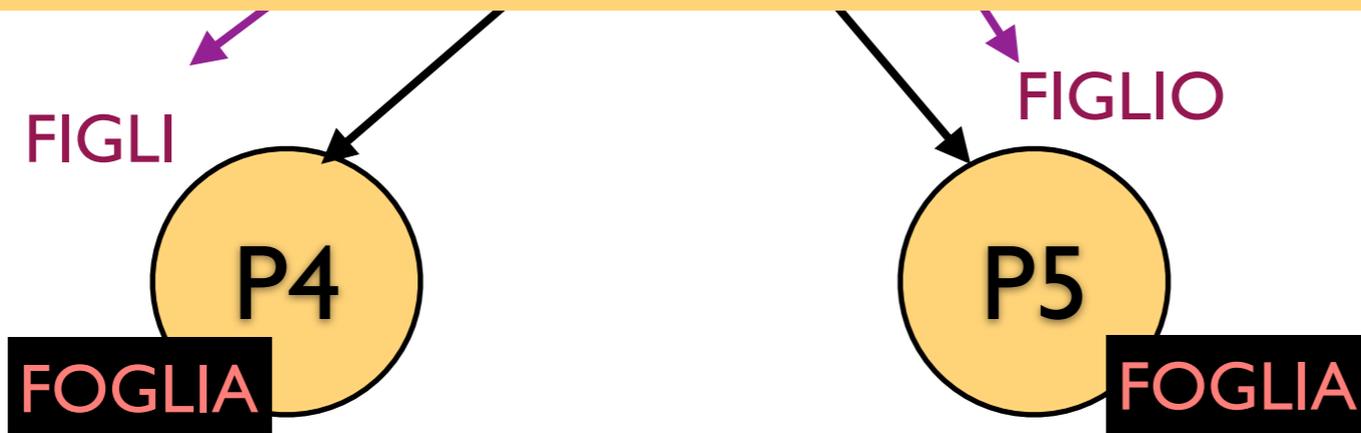




Una famosa struttura dati: l'albero

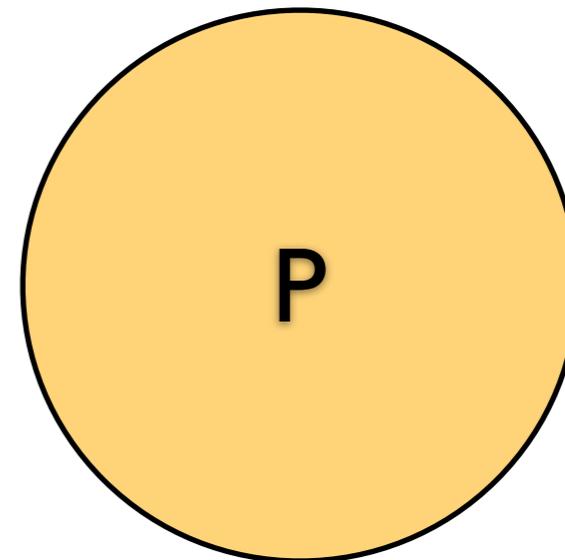
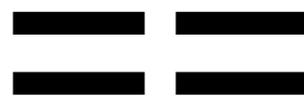


OVVIAMENTE SI



Può essere utile per rappresentare un albero genealogico?

- **OGNI NODO DELL'ALBERO SARA' PER NOI UNA PERSONA**





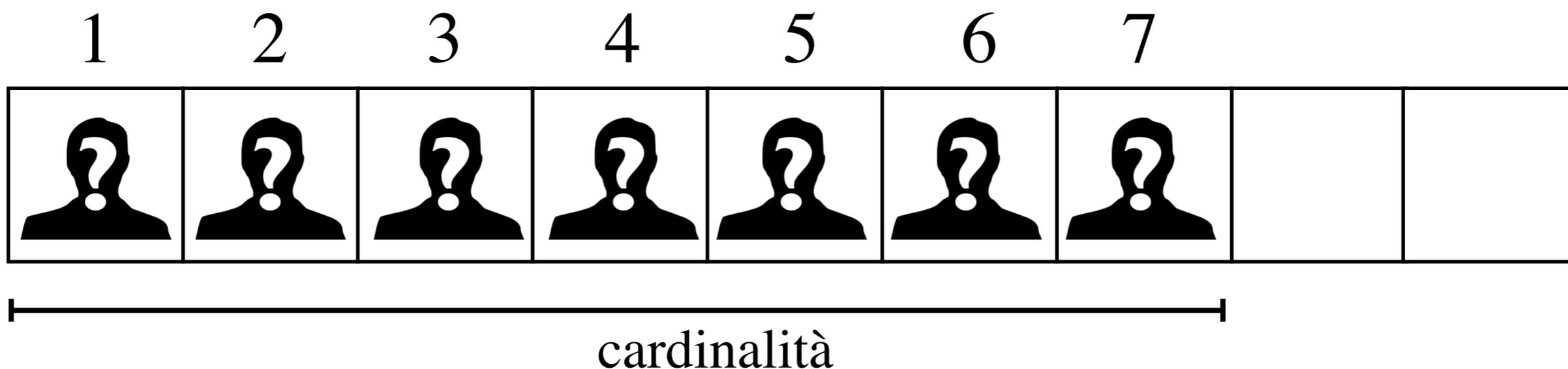
Una Persona

- SESSO
- NOME
- ETA?
- CHI SONO I GENITORI?
- CHI SONO I FIGLI?
- QUANTI FIGLI?



Una Popolazione

- Una popolazione è rappresentata da un insieme di persone
- Ogni persona ha un suo indice (numero univoco di identificazione)
- Esiste un numero di persone della

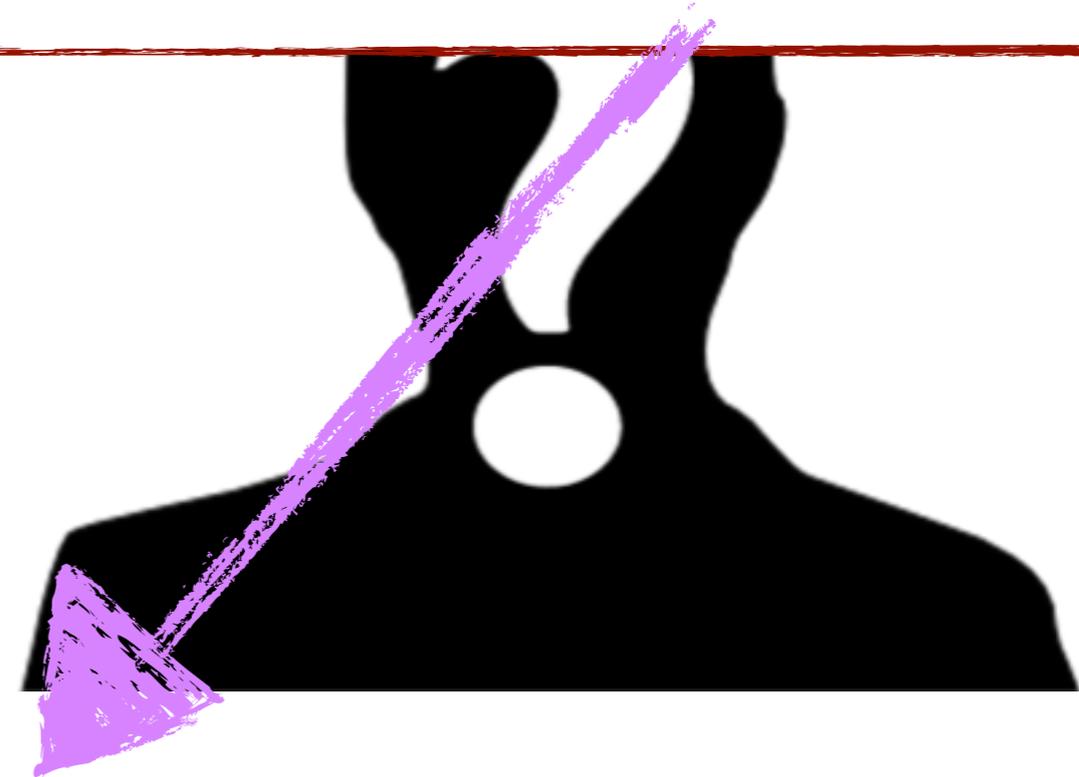




Una Persona nella popolazione

- SESSO
- NOME
- ETA?
- CHI SONO I GENITORI?
- CHI SONO I FIGLI?
- QUANTI FIGLI?

Li rappresentiamo con l'indice della persona nella popolazione



1 2 3 4 5 6 7



cardinalità



```
typedef struct {
```

```
} persona;
```

```
int main () |  
{
```

```
    persona* popolazione[MAX_PERSONE];  
    int cardinalita_popolazione = -1;
```



Creazione di una persona

```
persona crea_persona(genere sesso, char nome[STR_LEN], int eta)  
{
```

```
}
```



Aggiunta persona alla popolazione

```
int aggiungi_a_popolazione(persona* popolazione[MAX_PERSONE],
    persona* p, int* cardinalita_popolazione)
{
    .
    .
    .
}
}
```



Aggiunta di un figlio

```
void aggiungi_figlio(persona* genitore, persona* figlio)
{

}
}
```



Funzioni di stampa a schermo

```
char* genere_to_str(genere sesso)
{

}

void stampa_persona(persona* p)
{

}
```



Elenco dei figli e dei nipoti

```
void elenca_figli_nipoti(persona* popolazione[MAX_PERSONE],
    persona* p, genere sesso, int eta_minima)
{

}
}
```



La nostra popolazione



MARCO

P0



STEFANIA

P1



LUCA

P2



PIPPO

P3



LUCIA

P4



ARIANNA

P5



RINALDO

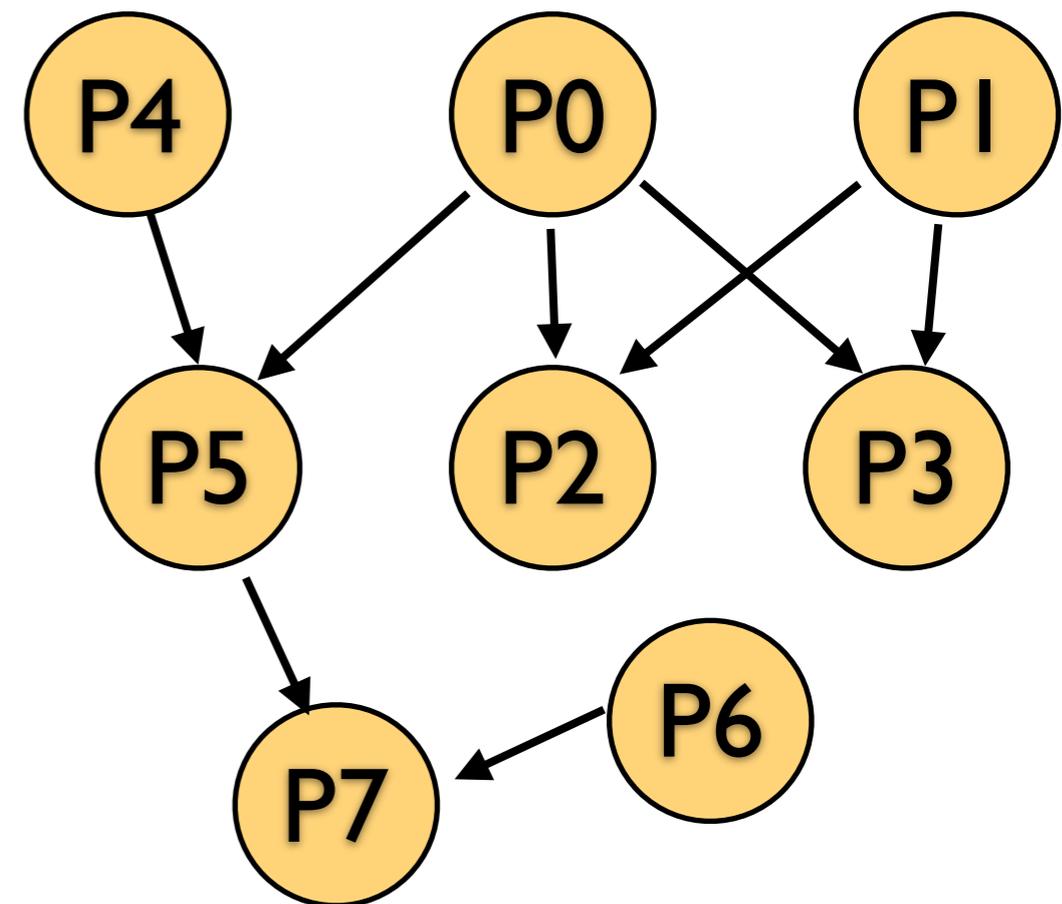
P6



STEFANO

P7

Marco e' padre di LUCA e di PIPPO
Stefania e' madre di LUCA e di PIPPO
Arianna e' figlia di Marco e Lucia
Stefano e' figlio di Arianna e Rinaldo





La nostra popolazione (codice C)

```
int main ()
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;

    persona p0 = crea_persona(MASCHIO, "MARCO", 50);
    persona p1 = crea_persona(FEMMINA, "STEFANIA", 49);
    persona p2 = crea_persona(MASCHIO, "LUCA", 30);
    persona p3 = crea_persona(MASCHIO, "PIPP0", 26);
    persona p4 = crea_persona(FEMMINA, "LUCIA", 53);
    persona p5 = crea_persona(FEMMINA, "ARIANNA", 30);
    persona p6 = crea_persona(MASCHIO, "RINALDO", 32);
    persona p7 = crea_persona(MASCHIO, "STEFANO", 10);

    aggiungi_a_popolazione(popolazione, &p0, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p1, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p2, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p3, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p4, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p5, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p6, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p7, &cardinalita_popolazione);
}
```



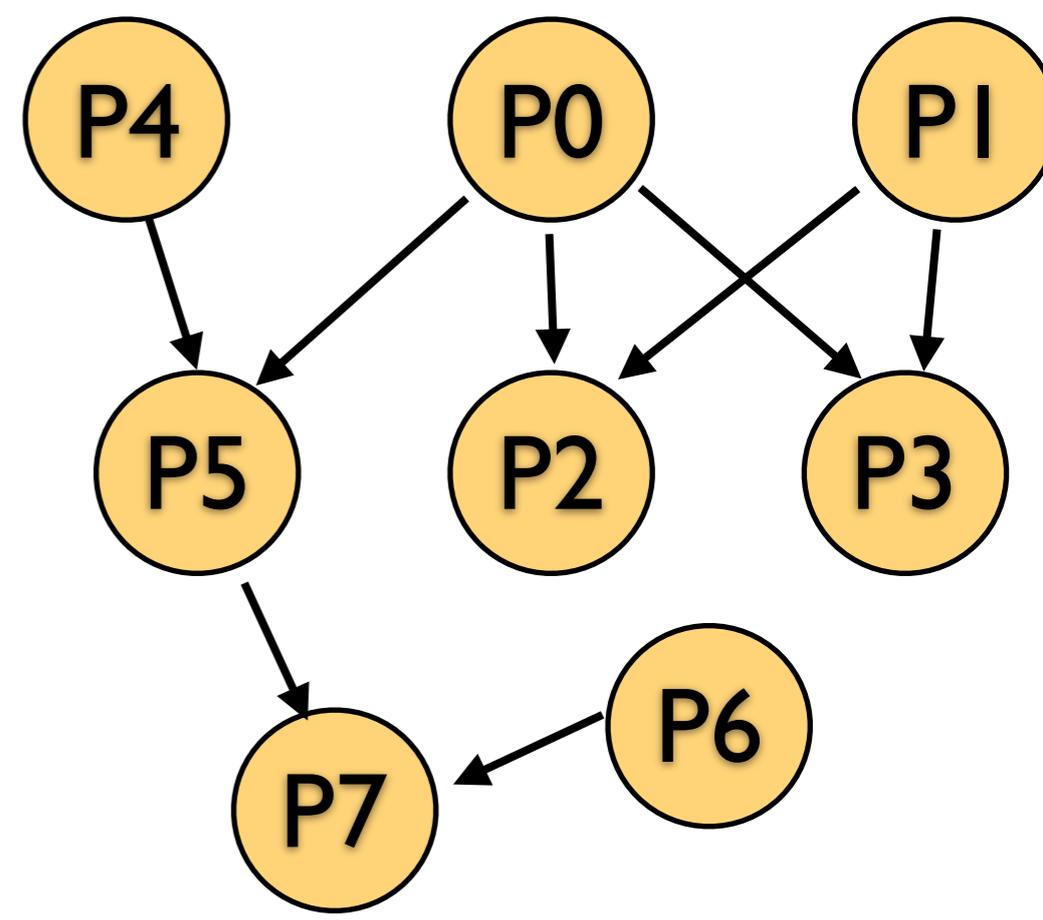
Aggiungiamo le parentele

```
// Marco e' padre di LUCA e di PIPPO
aggiungi_figlio(popolazione[0], popolazione[2]);
aggiungi_figlio(popolazione[0], popolazione[3]);

// Stefania e' madre di LUCA e di PIPPO
aggiungi_figlio(popolazione[1], popolazione[2]);
aggiungi_figlio(popolazione[1], popolazione[3]);

// Arianna e' figlia di Marco e Lucia
aggiungi_figlio(popolazione[0], popolazione[5]);
aggiungi_figlio(popolazione[4], popolazione[5]);

// Stefano e' figlio di Arianna e Rinaldo
aggiungi_figlio(popolazione[5], popolazione[7]);
aggiungi_figlio(popolazione[6], popolazione[7]);
```





Il main()

```
int main ()
{

    persona* popolazione[MAX_PERSONE];
    int cardinalita_popolazione = -1;

    persona p0 = crea_persona(MASCHIO, "MARCO", 50);
    persona p1 = crea_persona(FEMMINA, "STEFANIA", 49);
    persona p2 = crea_persona(MASCHIO, "LUCA", 30);
    persona p3 = crea_persona(MASCHIO, "PIPP0", 26);
    persona p4 = crea_persona(FEMMINA, "LUCIA", 53);
    persona p5 = crea_persona(FEMMINA, "ARIANNA", 30);
    persona p6 = crea_persona(MASCHIO, "RINALDO", 32);
    persona p7 = crea_persona(MASCHIO, "STEFANO", 10);

    aggiungi_a_popolazione(popolazione, &p0, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p1, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p2, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p3, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p4, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p5, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p6, &cardinalita_popolazione);
    aggiungi_a_popolazione(popolazione, &p7, &cardinalita_popolazione);

    // Marco e' padre di LUCA e di PIPPO
    aggiungi_figlio(popolazione[0], popolazione[2]);
    aggiungi_figlio(popolazione[0], popolazione[3]);

    // Stefania e' madre di LUCA e di PIPPO
    aggiungi_figlio(popolazione[1], popolazione[2]);
    aggiungi_figlio(popolazione[1], popolazione[3]);

    // Arianna e' figlia di Marco e Lucia
    aggiungi_figlio(popolazione[0], popolazione[5]);
    aggiungi_figlio(popolazione[4], popolazione[5]);

    // Stefano e' figlio di Arianna e Rinaldo
    aggiungi_figlio(popolazione[5], popolazione[7]);
    aggiungi_figlio(popolazione[6], popolazione[7]);

    elenca_figli_nipoti(popolazione, popolazione[0], MASCHIO, 3);

    return 0;
}
```



- Mercato
- Tris 1
- Tris 2
- Ricorsione



Es I - Mercato

- Si scriva un programma per la gestione del carrello di un supermercato.
- Si definisca una struttura dati per rappresentare un prodotto acquistabile; tale struttura deve contenere:
 - Codice prodotto
 - Nome
 - Prezzo
- Si definisca una struttura dati per rappresentare il carrello:
 - Lista di prodotti
 - Numero di prodotti acquistati
 - Totale da pagare
- Viene chiesto all'utente di inserire i prodotti che devono essere acquistati. Tali prodotti vengono inseriti nel carrello e le informazioni del carrello vengono aggiornate. Alla fine si procede alla stampa dello scontrino e del totale da pagare.
- Vincolo:
 - Si strutturi il programma dividendolo in sottofunzioni ove possibile. In particolare si creino apposite funzioni per:
 - Leggere un prodotto acquistato
 - Inserire il prodotto nel carrello
 - Calcolare il totale da pagare
 - Stampare lo scontrino



Es I - Mercato - Strutture Dati

```
#include <stdio.h>

#define MAX_PRODOTTI 10
#define MAX_NOME 50

typedef struct{
    int codice;
    char nome[MAX_NOME];
    float prezzo;
} prodotto;

typedef struct{
    prodotto prodotti[MAX_PRODOTTI];
    int numeroProdotti;
    float totale;
} carrello;
```



Es I - Mercato - Prototipi Funzioni

```
prodotto leggiProdotto();  
void inserisciProdotto(carrello *c, prodotto p);  
void calcolaTotale(carrello *c);  
void stampaScontrino(carrello *c);
```

```
#include <stdio.h>  
  
#define MAX_PRODOTTI 10  
#define MAX_NOME 50  
  
typedef struct {  
    int codice;  
    char nome[MAX_NOME];  
    float prezzo;  
} prodotto;  
  
typedef struct {  
    prodotto prodotti[MAX_PRODOTTI];  
    int numeroProdotti;  
    float totale;  
} carrello;
```



Es I - Mercato - Funzioni

```
prodotto leggiProdotto() {
    prodotto p;

    printf( "Codice: " );
    scanf( "%d", &p.codice );
    printf( "Nome: " );
    scanf( "%s", p.nome );
    printf( "Prezzo: " );
    scanf( "%f", &p.prezzo );

    return p;
}
```



Es I - Mercato - Funzioni

```
void inserisciProdotto(carrello *c, prodotto p){  
    c->prodotti[c->numeroProdotti]=p;  
    c->numeroProdotti+=1;  
}
```



Es I - Mercato - Funzioni

```
void calcolaTotale(carrello *c){  
    int i;  
  
    c->totale=0;  
  
    for(i=0;i<c->numeroProdotti;i++)  
        c->totale+=c->prodotti[i].prezzo;  
}
```



Es I - Mercato - Funzioni

```
void stampaScontrino(carrello *c){
    int i;

    for(i=0;i<c->numeroProdotti;i++)
        printf("%d\t%s\t%f\n",c->prodotti[i].codice, c->prodotti[i].nome, c->prodotti[i].prezzo);

    printf("\n\tTOTALE:\t%f\n",c->totale);
}
```



Es I - Mercato - Main

```
int main(){
    carrello c;
    prodotto p;

    c.numeroProdotti=0;

    int i;
    char s;

    do{
        printf("Vuoi comprare un prodotto (s/n)? ");
        scanf(" %c",&s);

        if(s=='s' || s=='S'){
            p=leggiProdotto();
            inserisciProdotto(&c,p);
        }
    }while((s=='s' || s=='S') && c.numeroProdotti<MAX_PRODOTTI);

    calcolaTotale(&c);
    stampaScontrino(&c);

    return 0;
}
```

**Tutte il materiale sar 
disponibile sul mio sito
internet!**

alessandronacci.it

See You Next Time!

