

TERMINALE

SPOSTA LA CARTELLA SUL DESKTOP

cd DESKTOP/

cd e ls FINO ALLA CARTELLA CON L'es.c DA FAR E
APRI IL FILE es.c DALLA CARTELLA E SALVIALO
SU TERMINALE MAKE

ls PER VEDERLE QUALE ESEGUIRE

. / test - es.c

SE DA' IL 100% RINOMINA

MV es.c 1834563.c

ls PER CONTROLLARLE

INVIA TRAMITE FORM GOOGLE (DESKTOP → CARTELLA → 1834563.c → CARICA)

```
#include <stdio.h>/<stdlib.h>/<string.h>/<math.h>
#include "file.h"
```

%c	char → 1 byte	{	%hd	short int → 2 byte
%d	int → 4 byte		%ld	long int → 8 byte
%f	float → 4 byte		%lf	double → 8 byte
%X	int esca		%p	pointer
%u	UNS. INT → 4 byte		%s	stringa

IF if(condiz){ ...; } else{ ...; }

WHILE while (condiz) { ...; }

DO do { ...; } while (condiz)

FOR for (inizial ; condiz ; incremento){ ...; }

SWITCH

```
SWITCH(espresso){  
caso 1 : ISTRUZ; break;  
...  
caso n : ISTRUZ; break;  
default : ISTRUZ;  
}
```

equivalente a: inizializzatore;

```
while (condiz) {  
ISTRUZ;  
incremento; }
```

FUNZIONI tipo riservato nomefunz (param) {
ISTRUZ;
}

CAST void * p;
int * pi = p; (AUTOMATICO) / int * pi = (int *) p; (ESPL.)

ARRAY tipo a[n] = {n0, n1, ..., nn};
accesso a[i] con posizioni da 0 a n-1.

STRINGHE char s[n];
s[ultimo] = s[n-1] = '\0';

```
{ printf ("%s\n", s);  
{ puts(s); con char * s = "stringa\0";  
{ scanf ("%s", s); } con char s[256]; inizializzata  
{ gets(s);  
getchar() / putchar()
```

ALLOC alloca n byte contigui restituendo void *

es. int * a = (int *) malloc(10 * sizeof(int));

FREE free(a); con a puntatore

CALLOC char * res = (char *) calloc((len+1), sizeof(char))

FILE

operazioni tramite puntatore a record FILE

fopen : FILE * filep = fopen ("path/ nome.txt", "w+");

modifica : r lettura
w scrittura (sovrascrive o crea)
a accodamento (accoda o crea)
r+ lettura e scrittura
w+ " " (sovrascrive o crea)
a+ " " (accoda o crea)

in caso di insuccesso : NULL

fclose : fclose (filep);

in caso di successo : 0

in caso di insuccesso : EOF

output : fprintf (filep, "...");
fputs (" ", filep);

lettura : fscanf (filep, "%s", s);
alla fine del file restituisce EOF
fgets (char *line, int n, FILE *file);
cioè fgets (line, sizeof(line), filep);
con line: char line[250];
e legge fino al carattere di fine linea
fgetc (filep); legge char per char

SCHEMA DI CYCLE DI LETTURA DA FILE

```
FILE * file = fopen (...);
```

```
while (!<fine-file>) {
```

// leggi prox blocco (char, parola o riga)

```
}
```

```
fclose (file);
```

```
...
```

fgetc fscanf fgets

MATRICI

- collezione di elementi omogenei su array BIDIMENSIONALE
- 2 indici di accesso
- DICHIARAZIONE + INIZIAZIONE.
const int r = 3;
const int c = 3;
int m[r][c] = {{1,2,3}, {4,5,6}, {7,8,9}};
- # BLOCHE TOT size of (m);
ELEM DI M sizeof(m) / sizeof (tipo di dato);
COLONNE sizeof (m[0]) / sizeof (tipo di dato);
RIGHE #elem / # colonne;
- NB se passi m tramite puntatore non puoi fare sizeof()
- CICLO X RIGHE:

```
for (int i=0; i < n-righe; i++) {
    for (int j=0; j < n-colonne; j++)
        istruzioni;
}
```

int m[r][c];
m[0][0] = 3;
m[0][1] = 5;
m[1][0] = 3;

ACCO X COLONNE:

```
for (int j=0; j < n-colonne; j++) {
    for (int i=0; i < n-righe; i++)
        istruzioni;
}
```

ALLOCAZIONE DINAMICA DI MATRICE COME ARRAY DI ARRAY

```
int righe = 10, colonne = 5;
int **m = (int **) malloc (righe, sizeof (int *));
for (int i=0; i < righe; i++) {
    m[i] = (int *) calloc (colonne, sizeof (int));
}
```

E ACCESSO con m[i][j] (poi che equivale a *(*(m+i)+j))

⊗ x definire lo SEMANTICA di m[i][j] e' necessario sapere come
e' definita m. se e' un array di array, la semantica e'
quando possi una mat come parametro dell'i passare
anche le dimensioni

ENUMERATI

```
tupedef enum {
    AUTO,
    BASSO,
    DX,
    SX,
} Direzione;
Direzione = d;
d = BASSO;
if (d == AUTO) { ... };
if (d == BASSO) { };
```

NB
MAIUSCOLO:

(c >='A' && c <='Z')

minuscolo:

(c >='a' && c <='z')

MAIUSC → minusc

c - 'A' + 'a'

c = c + 32

minusc → MAIUSC

c - 'a' + 'A'

c = c - 32

RECORD (cap 7)

DEFINIZIONE DI RECORD CON STRUCT

```
struct {  
    char * nome;  
    char * cognome;  
    short int anno;  
} persona1, persona2;
```

NB ogni struct è a sé
quindi non crea un nuovo tipo
da usare x più inizializzate.

DEFINIZIONE DI TAG DI STRUTTURA

```
struct persona {  
    char * nome *cognome;  
    short int giorno, mese, anno;  
};
```

```
struct persona persona1, persona2;  
struct persona persona3;
```

DEFINIZIONE DI RECORD CON TYPE DEF

```
typedef struct {  
    char * nome;  
    char * cognome;  
    short int anno;  
} persona;
```

DEFINIZIONE RECORD CON TYPEDEF + TAG

```
struct persona {  
    char * nome, *cognome;  
    short int giorno, mese, anno;  
};
```

```
typedef struct persona persona;  
(definisce persona tramite persona)
```

ACCESSO E INIZIALIZZAZIONE

① persona p1;
p1.nome = "Mario";
p1.cognome = "Rossi";
p1.anno = 1964;

② persona p1 = {"Mario", "Rossi", 1964};

③ persona p1 = {
 .cognome = "Rossi";
 .anno = 1964;
 .nome = "Mario";
};

ALLOCAZIONE DINAMICA

```
persona * p1 = (persona*) malloc  
(sizeof (persona));
```

```
free (p1);
```

ASSEGNAZIONE DI RECORD

persona famiglia[3];
famiglia[0].nome = "Mario";
famiglia[1].nome = "Giovanni";
famiglia[2].nome = "Giulia";

ACCESO TRAMITE PUNTATORE

```
persona * p = {"Mario", "Rossi", 1964};  
p1.punt = &p  
p1.punt = "%s\n"; (*punt).nome);  
p1.punt = "%s\n"; punt -> nome);
```

FUNZIONI SU RECORD

x non si deve trasmettere sia struttura
sue voci che il record fatto con punzatore

SCL - STRUTTURE CONNESETE LINEARI

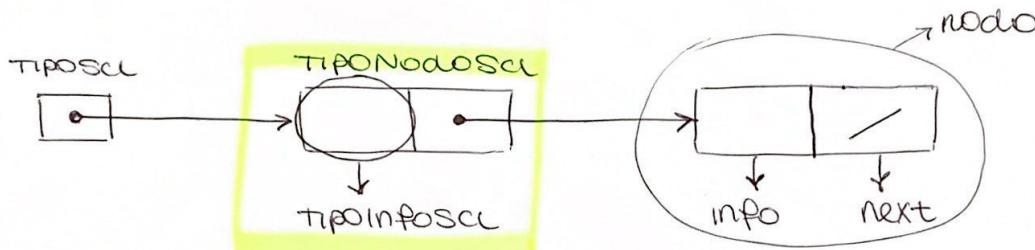
```
typedef ... TipoInfoScl;
struct ELEM_SCL {
    TipoInfoScl info;
    struct ELEM_SCL * next;           → CHIAMATA RICORSIVA
};
```

```
typedef struct ELEM_SCL TIPO_NODO_SCL;
typedef TIPO_NODO_SCL * TIPO_SCL;
```

CON: **TipoInfoScl**: tipo dati contenuti in info

TIPO_NODO_SCL: nome del tipo del nodo (record info + next)
(usato x non usare tap di struttura di ELEM_SCL)

TIPO_SCL: puntatore alla struttura del nodo TIPO_NODO_SCL



ultimo nodo \rightarrow next = NULL; (-)

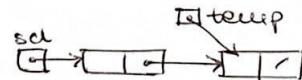
[SCL VUOTA] TIPOSCL scl = NULL;

[CREAZIONE 1 nodo]

```
TIPOSCL = (TIPO_NODO_SCL *) malloc (sizeof (TIPO_NODO_SCL));
scl -> info = e1; scl -> next = NULL;
```

[CREAZIONE collegamento nodo]

```
TIPOSCL temp = (TIPO_NODO_SCL *) malloc (sizeof (TIPO_NODO_SCL));
temp -> info = e2
temp -> next = NULL;
scl -> next = temp;
```



il collegamento è sempre del tipo $p \rightarrow next = q$

[INserimento nodo in 1a posizione]

```
void addSCL (TIPOSCL * scl, TIPOINFO_SCL e) {
    TIPOSCL temp = * scl;
    * scl = (TIPO_NODO_SCL *) malloc (sizeof (TIPO_NODO_SCL));
    (* scl) -> info = e;
    (* scl) -> next = temp;
}
```

[ELIMINAZIONE nodo in 1a posizione]

```
void delSCL (TIPOSCL * scl) {
    TIPOSCL temp = * scl;
    * scl = (* scl) -> next;
    free (temp);
}
```

Funzioni che modificano → STRUTTURA SCL
 ↳ contenuto SCL

contenuto ha uno spazio di tipo **TipoSCL**

if (!emptySCL(scl)) {
 scl → next
 → se non è vuota (cioè scl != NULL)
 → prossimo nodo}

struttura ha uno spazio di tipo **TipoSCL ***



Serve l'indirizzo

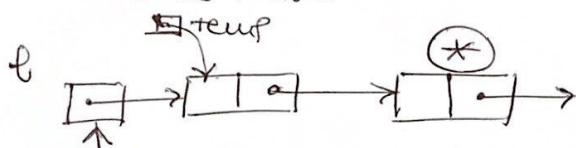
NON è contenuto

il contenuto è il nodo

⇒ serve puntatore

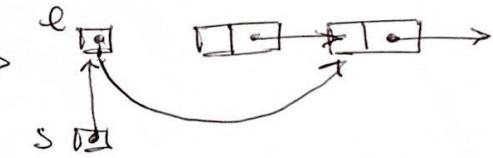
TipoSCL * s

(puntatore a tipo scl)



TipoSCL temp = *s;

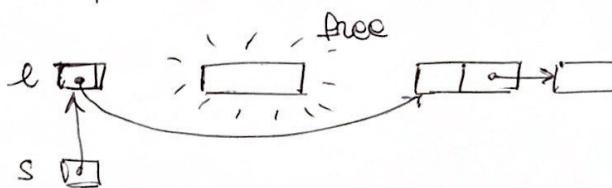
***s = (*s) → next**



è qui (*)

free(temp);

cioè alla fine



TIPO ASTRATTO

liste - code - pile

- le scu servono x implementare questi tipi di dato astratto

↓

costituito da:

- ① DOMINIO DI INTERESSE (uso dei domini del tipo) contiene gli elementi che è al tipo
 - ② COSTANTI valori del dominio utili x la rappresentazione degli elementi
 - ③ ~~XXXXXX~~ FUNZIONI che operano sul dominio e producono risultati sugli elementi
- con i tipi astratti passo definire tutti i tipi di dato.
 - Tipi astratti lineari: liste, pile, code
non lineari: alberi binari, alberi n-ari, graph
 - definizione secoli formelle

TipoAstratto T

Domini: D₁: Descriz.

...
D_m: Descriz.

Costanti: C₁: Descriz.

...
C_k: Descriz.

Funzioni: F₁: Descriz. con pre e post condizioni

F_u: Descriz. con pre e post condizioni

Fine TipoAstratto

nella pratica

implementazione:

- ① DOMINI → usando tipi concreti del linguaggio¹
- ② COSTANTI → usando costanti del linguaggio²
- ③ OPERAZIONI → usando funzioni del lin.

Dai decidere:

- FUNZIONI fanno side-effect o no?
- DATI ≠ condividono memoria o no?

Schemi realizzativi:

- CON SIDE-EFFECT se modificano i dati che vengono passati come argomenti della funzione
- IN MODELLO A' FUNZIONALE se si crea un nuovo dato × RESTITUIRE risultato senza modificare l'INPUT
- SENZA CONDIVISIONE DI MEMORIA se le funzioni assicurano che non ci siano interruzioni
- CON CONDIVISIONE DI MEMORIA

SCHEMI + INTERESSANTI:

- CON SIDE-EFFECT / SENZA CONDIVISIONE × realizzare dati mutabili
- SCHEMA FUNZIONALE / CON CONDIVISIONE × realizzare oggetti IMMUTABILI

TIPO INSIEME

usiamo i 2 schemi SIDE-EFF / NO CONDIV. (MUTABILI)

FUNZ / CON CONDIV. (IMMUTAB.)

↑
genera +
nuovo insieme

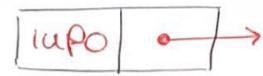
vedi funzioni di implementazione sui file
RICONDA callo c inizializza a zero!

esercitazione TDP 2018-2019 (TIPI DI DATO ASTR. - INSIEMI)

`ttypedef int T;`

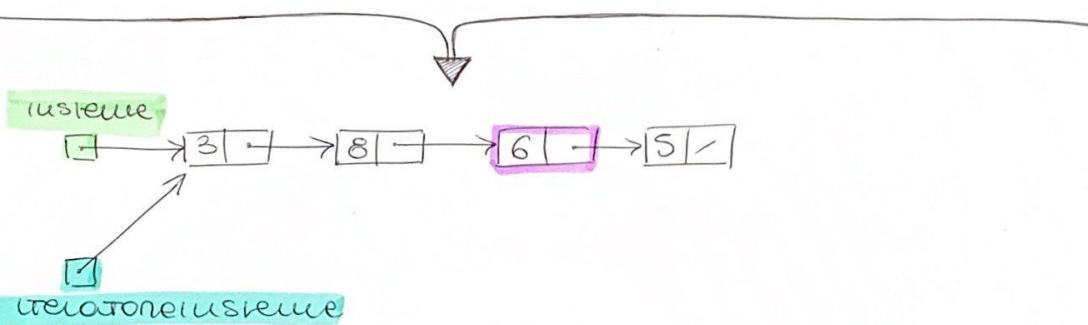
`#define TERRORVALUE -999999; → Valore errore di tipo T`
`STRUCT NodoSCL {`

`T info;`
`STRUCT NodoSCL * next; };`



`ttypedef NodoSCL * Insieme; → l'insieme è un puntatore a un Nodo SCL`

`ttypedef STRUCT {`
`NodoSCL * ptr;`
`? Iteratore Insieme; }`



Dalla versione 5/5/2020

`ttypedef int T;`

`STRUCT NodoSCL {`

`T info;`

`STRUCT NodoSCL * next;`

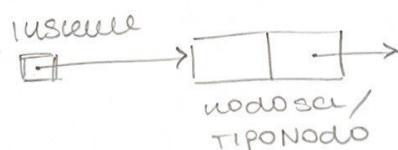
`?; }`

`ttypedef STRUCT NodoSCL TIPONODO;`

`ttypedef TIPONODO * Insieme;`

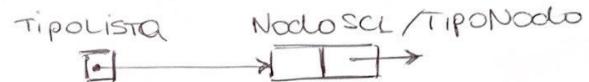
→ cioè TIPONODO è utilizzato
per dichiarare la STRUCT
NodoSCL

→ cioè Insieme è un puntatore
al Nodo SCL



LISTE: collezioni ordinate di dati omogenei ()
le operazioni di base riguardano solo il 1° elemento

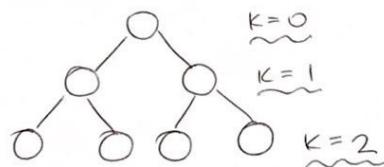
```
ttypedef int T;  
struct NodoSCL {  
    T info;  
    struct NodoSCL * next;  
};  
ttypedef struct NodoSCL TIPONodo;  
ttypedef TIPONodo * TipoListau;
```



AUBEN

albero binario: ① l'albero vuoto è un albero binario
② se B_d e B_s sono alberi binari, allora
l'albero che ha un nodo radice
e B_d e B_s come sottoalberi è binario
③ nient'altro è un albero binario

albero binario completo: se ogni nodo che non sia
una foglia ha 2 figli e se
le foglie sono tutte alla stessa
livello



nodi: un albero binario completo non vuoto
di profondità K ha $2^{K+1} - 1$ nodi

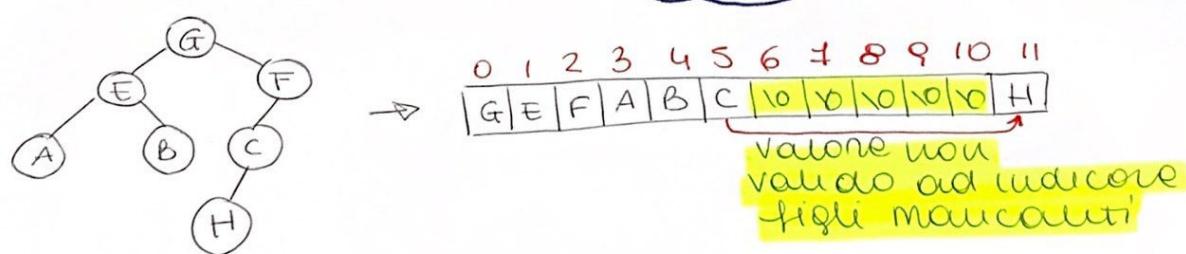
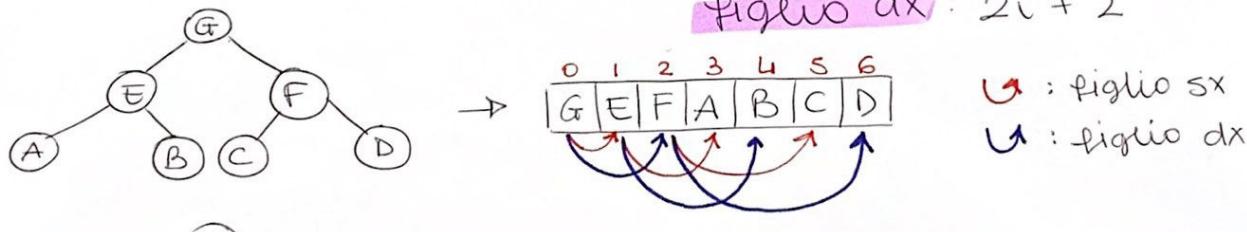
Rappresentazione con: array, scnl

RAPPRESENTAZIONE CON ARRAY INDICETATO

radice: posizione 0

Dato un nodo in posizione i : figlio sx: $2i + 1$

figlio dx: $2i + 2$



questa rappresentazione ha una dimensione
esponenziale $O(2^n)$ rispetto agli elementi dell'
albero (n) .

RAPPRESENTAZIONE CON SCNL

PRO: modifiche locali
allocazioni proporzionali all'effettivo # di nodi

```
struct StructAlbero {
```

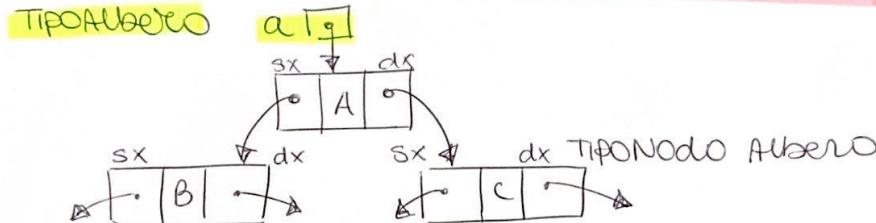
```
    TipoInfoAlbero info;
```

```
    struct StructAlbero * destro, * sinistro;
```

```
};
```

```
typedef struct StructAlbero TipoNodoAlbero;
```

```
typedef TipoNodoAlbero * TIPOAlbero;
```



TIPOAlbero a \square Albero vuoto

RAPPRESENTAZIONE PARENTETICA

albero binario T \rightarrow stringa f(T);

albero vuoto \rightarrow f(albero vuoto) = "();

albero binario con radice x e sottoalberi sx e dx

T₁ e T₂ \rightarrow f(T) = "("+ x + f(T₁) + f(T₂) + ")";



cioè $($ radice $(f(T_1))$ $(f(T_2))$ $)$

QUESTA è UN'ALTRA DEFINIZIONE INDUTTIVA!

RICORSIONE + FACILE DI ITERAZIONE X ALBERI !!!

se vuoto \rightarrow passo base

altrimenti (A T₁ T₂) \rightarrow operazione su A

f(T₁) (riconosciuta)

f(T₂) (riconosciuta)

RICORSIONE NON LINEARE !

VISITE NEGLI ALBERI

IN PROFONDITA': e' ordine di visita dipende dai collegamenti padre - figlio (cioe' 2 nodi visitati consecutivamente hanno questa relazione)

E' impossibile implementare questa visita con ciclo while di costo $O(n^2)$ (lineare)

E' necessaria una struttura dati aggiuntiva: la PILA (o STACK) (LIFO).

IN AMPIEZZA: e' ordine di visita dipende dai livelli (cioe' visita tutti i nodi dello stesso livello, poi passa al livello successivo).

E' necessaria una struttura dati aggiuntiva: la CODA (FIFO).

PROFONDITA' \rightarrow PILA \rightarrow RICORSIONE (usando pila RDA)

\rightarrow ITERAZIONE \rightarrow NECESSARIA IMPLEMENTAZIONE DI UNA PILA AGGIUNTIVA

AMPIEZZA \rightarrow CODA \rightarrow ITERAZIONE

\rightarrow RICORSIONE \rightarrow NECESSARIA IMPLEMENTAZIONE RICORSIVA DELLE CODE USANDO LE PILE (inefficiente)

TIPOLOGIE DI VISITE IN PROFONDITA'

IN PREORDINE: radice, sottoalb. sx, sottoalb. dx

SIMMETRICA: sottoalb. sx, radice, sottoalb. dx

IN POSTORDINE: sottoalb. sx, sottoalb. dx, radice

(A Ts Td) \rightarrow Analisi A (radice)

rac(Ts)

rac(Td)

PRE	SIM	POST
1	2	3
2	1	-1
3	3	2

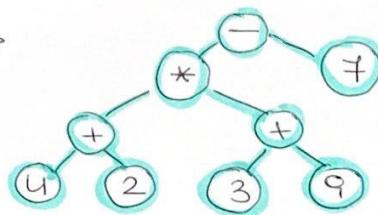


ESPRESSIONI ARITMETICHE CON ALBERI

- nodi intermedi: operatori
- foglie: valori
- sono + in profondità i nodi di operatori con priorità + alta

es

$$(4+2) * (3+9) - 7 \Rightarrow$$



valuto radice \rightarrow valuto sottobalb. sx (ricorsivamente) \rightarrow
 \rightarrow valuto sottobalb. dx (ricorsivamente) \rightarrow risultato
 \Rightarrow SI ESEGUE UNA VISITA IN PROF. IN POSTORDINE

int ValutaEspressione(TipoAlbero a) {

 if (estvuoto(a)) { errore; }

 if (estvuoto(a \rightarrow sx) || estvuoto(a \rightarrow dx)) { cioè se c'è un solo nodo
 return a \rightarrow info; }

// altrimenti se sono presenti i sottobalberi

 int s = ValutaEspressione(a \rightarrow sx);

 int d = ValutaEspressione(a \rightarrow dx);

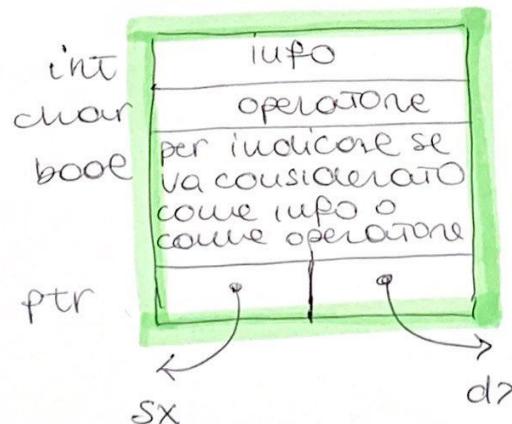
 if (operandoRadice == -) { return s - d; }

 if (operandoRadice == +) { return s + d; }

 if (operandoRadice == *) { return s * d; }

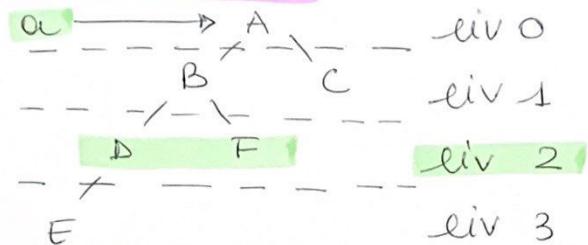
 if (operandoRadice == /) { return s / d; }

(NB) se NodoAlbero può contenere sia valori che operatori
 \Rightarrow non basta un solo campo 'info'!



\rightarrow non sempre necessaria!
 (per esempio se si usa il criterio:
 ha figli \Rightarrow operazione
 non ha figli \Rightarrow int)

STAMPA DEI NODI DI UN ALBERO IN BASE AL LIVELLO CON VISITA IN PROFONDITÀ



Dato il ptr ad albero a e scelto il livello $l=2$
risolviamo ricorsivamente

Void stampa.livello(TipoAlbero a, int l) {

if vuoto { return; }

if $l=0$ { print radice; }

if $l>0$ {

stampa.livello (sottoalb sx , $\frac{l-1}{l}$);

stampa.livello (sottoalb dx , $\frac{l-1}{l}$);

}

} → passi base

→ chiamata ricorsiva

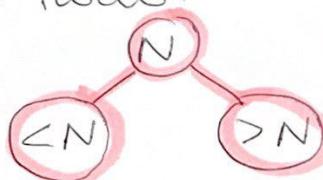
poiché il livello l
di un albero corrisponde
al livello $l-1$ del
suo sottoalbero!

ALBERI BINARI DI RICERCA

- COSTO di ricerca di un valore in un albero binario di n elementi : $\Theta(n)$

BINARY SEARCH TREE (BST)

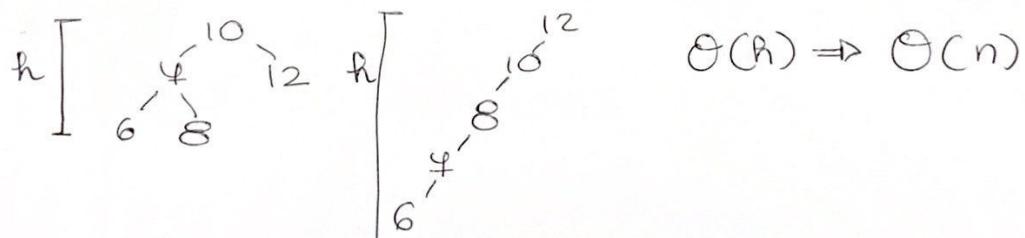
è un albero binario tale che, per OGNI nodo, tutti gli elementi del sottoalbero SX sono < del nodo, e tutti gli elementi del sottoalbero DX sono > del nodo.



essendo questa definizione valida per OGNI nodo (ricorsiva), dato un nodo N di un BST i suoi sottoalberi SX e DX sono a loro volta BST.

QUANTO COSTA LA RICERCA SU UN BST?

ad ogni passo scendiamo di un livello, e non vediamo mai più nodi sullo stesso livello.
quindi il costo è proporzionale ad h (altezza)
nel caso peggiore però $h = n !!!$



SI PUÒ RIDURRE QUESTO COSTO?

FATTORE DI BILANCIAMENTO (dato un nodo):

differenza tra l'altezza del sottoalbero SX e l'altezza del sottoalbero DX presa in modulo

Albero Bilanciato se ha fattore di bilanciamen~~to~~
to ≤ 1 per tutti i suoi nodi

Se un BST è BILANCIA~~TO~~to allora la sua h sarà
al più pari a $\log n$.

$$h \leq \log_2 n + 1$$

quindi il costo di ricerca sarà, per un BST BILANCIA~~TO~~TO, $\Theta(h) = \Theta(\log n)$