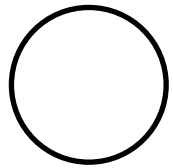
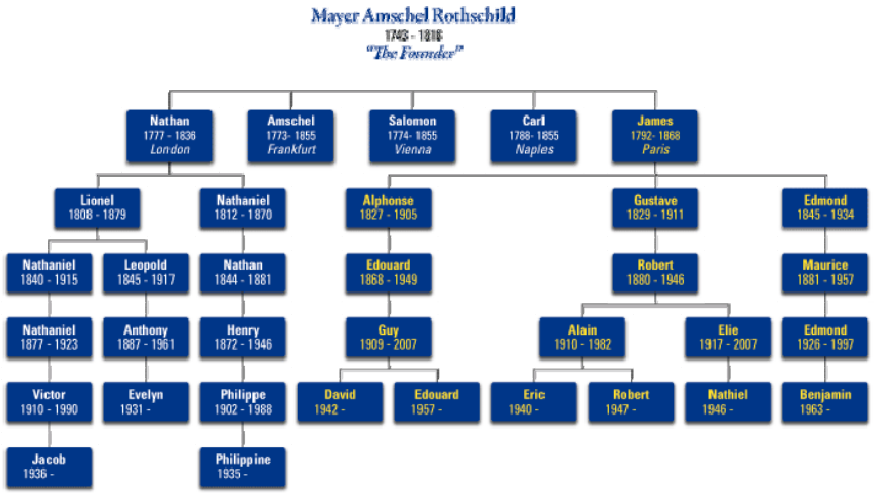
ALBERI N-ARY

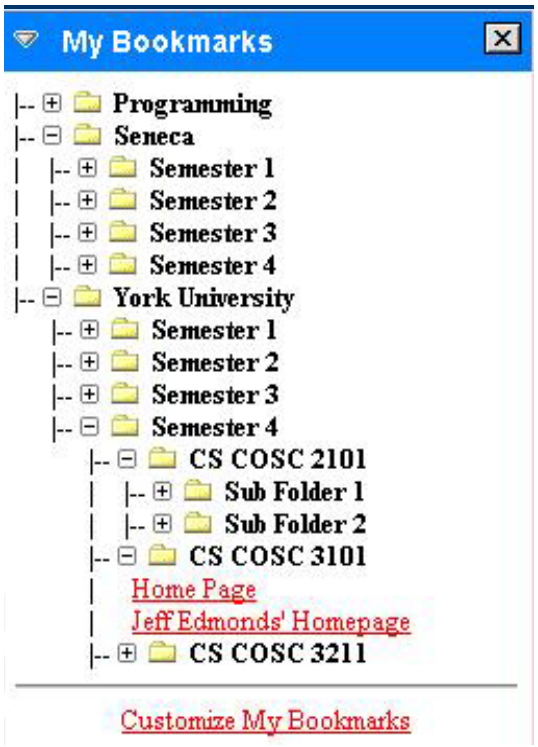
Lezioni di C



RAPPRESENTAZIONE GERARCHICA

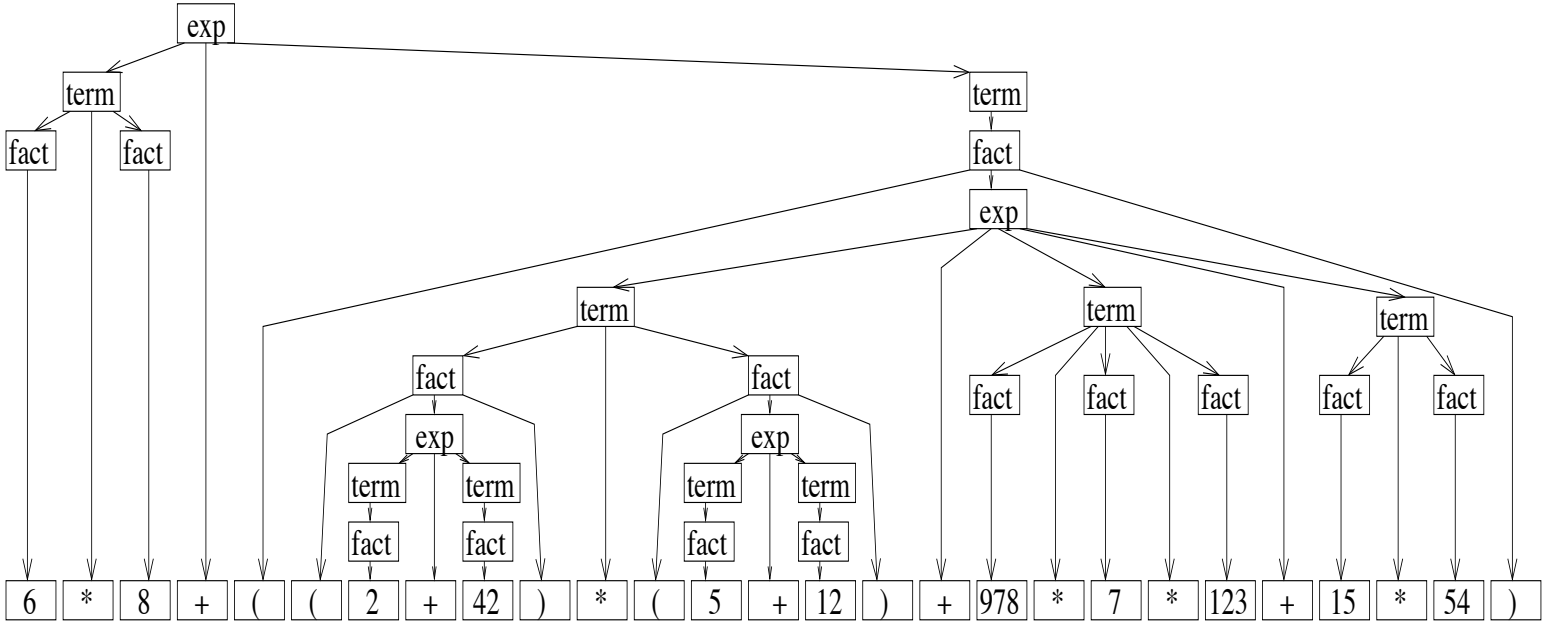
- L'albero e' un tipo astratto di dati usato per rappresentare relazioni gerarchiche.
  - struttura del file system
  - albero genealogico
  - organigramma
  - albero di decisione
  - ....





CODIFICA DI UNA ESPRESSIONE

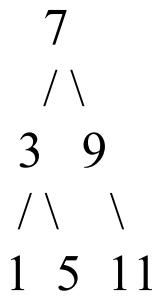
- *s*=6*8+((2+42)*(5+12)+987*7*123+15*54)



Osservazione: vale anche per il linguaggio naturale

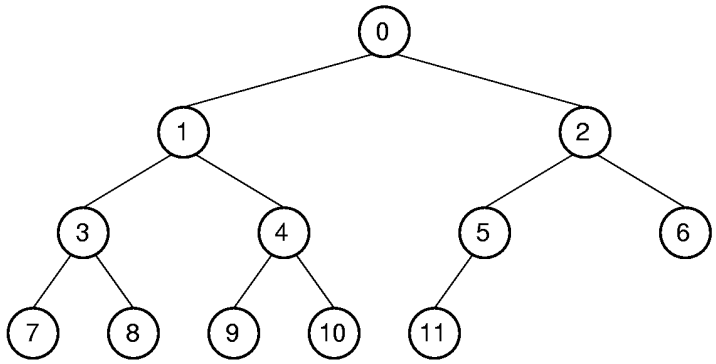
RAPPRESETAZIONE DI ALBERI

- Per gli alberi si possono avere 2 rappresentazioni*:*
  - *mediante array*
  - *a puntatori*
- Per rappresentare un albero binario di profondità n basta un array in cui riservare memoria per ogni nodo; nel caso di alberi sbilanciati i nodi mancanti avranno un valore di default (-1).



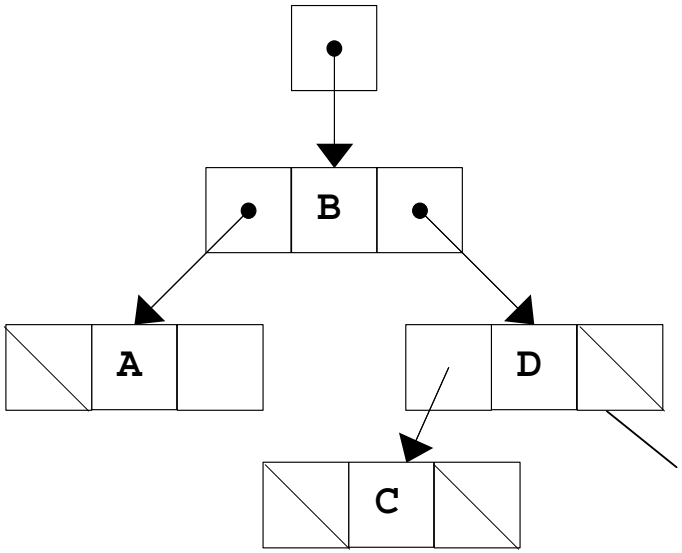
7	3	9	1	5	-1	11

IMPLEMENTAZIONE IN ARRAY 2D



posizione	0	1	2	3	4	5	6	7	8	9	10	11
genitore	--	0	0	1	1	2	2	3	3	4	4	5
Figlio sinistro	1	3	5	7	9	11	--	--	--	--	--	--
Figlio destro	2	4	6	8	10	--	--	--	--	--	--	--
Fratello sinistro	--	--	1	--	3	--	5	--	7	--	9	--
Fratello destro	--	2	--	4	--	6	--	8	--	10	--	--

RAPPRESENTAZIONE A PUNTATORI

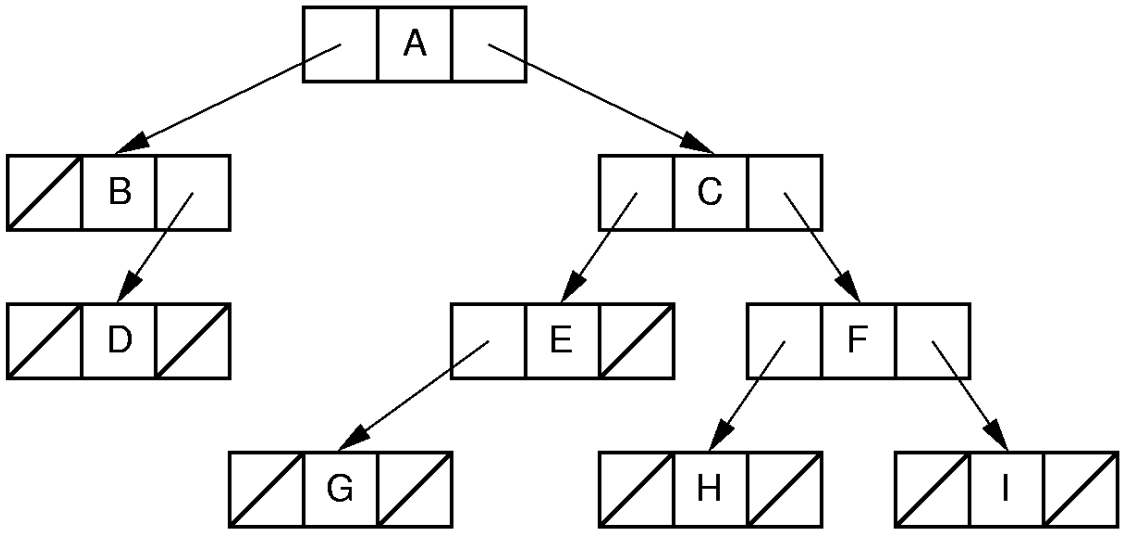



```
typedef struct Nodo {
      Tipo data;
      struct Nodo *left;
      struct Nodo *right;
} nodo;

typedef nodo * tree;
```

RAPPRESENTAZIONE A PUNTATORI

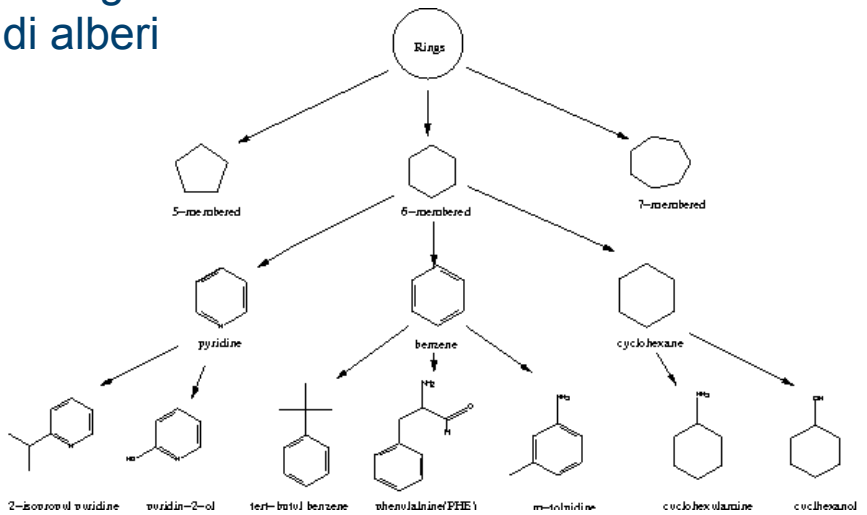
- Le foglie sono implementate come i nodi interni, quindi si spreca spazio per i puntatori nulli.



ALBERO N-ARIO

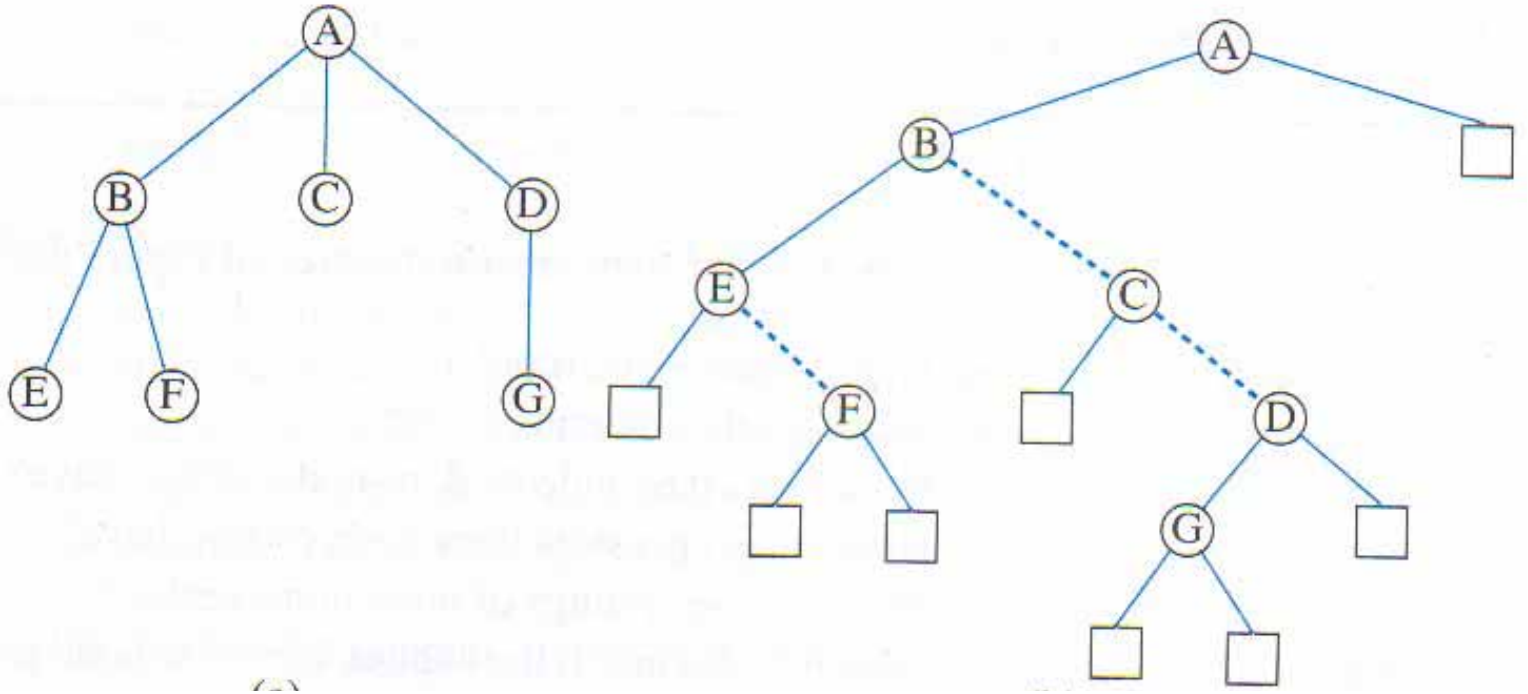
- Ogni nodo ha un numero arbitrario di figli
- Si usa ad esempio per rappresentare tassonomie e organizzazioni gerarchiche

definito ricorsivamente:
<albero> =ε | nodo seguito da
un numero finito di alberi

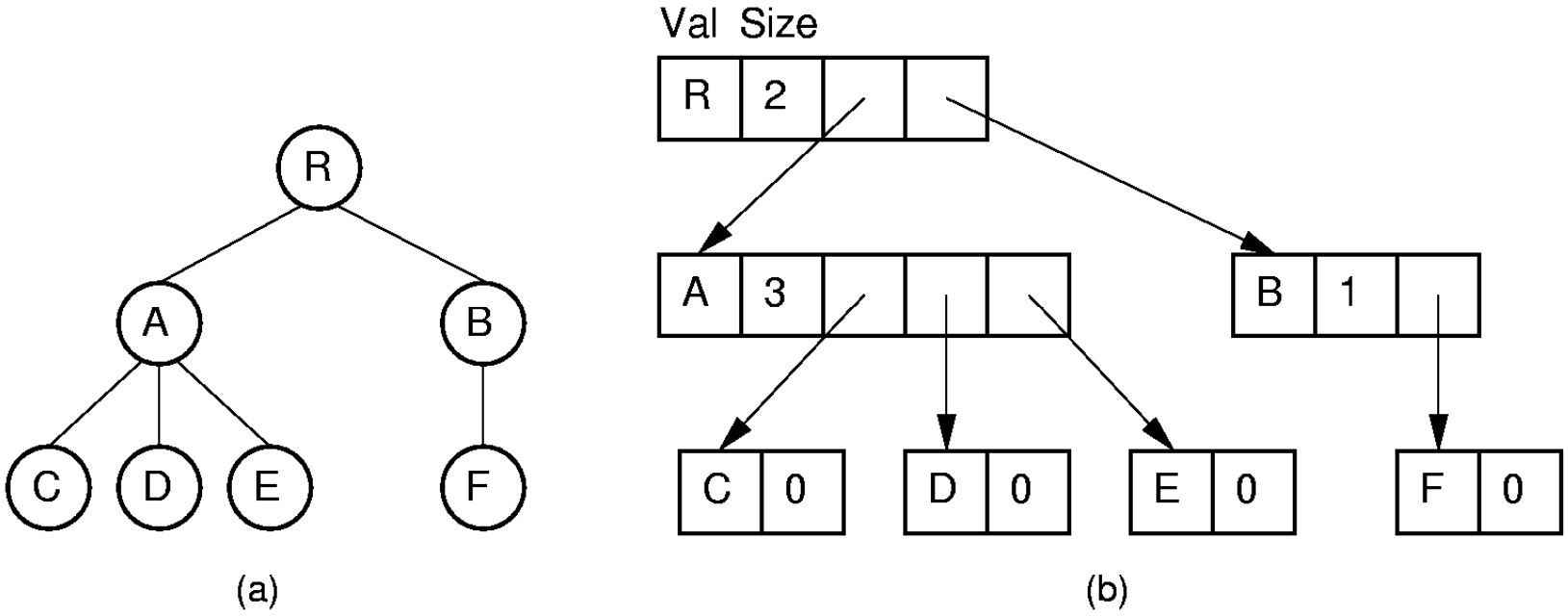


CONVERSIONE IN ALBERO BINARIO

- Il figlio di sinistra diventa sottoalbero sinistro;
- Il sottoalbero destro contiene il primo fratello verso destra



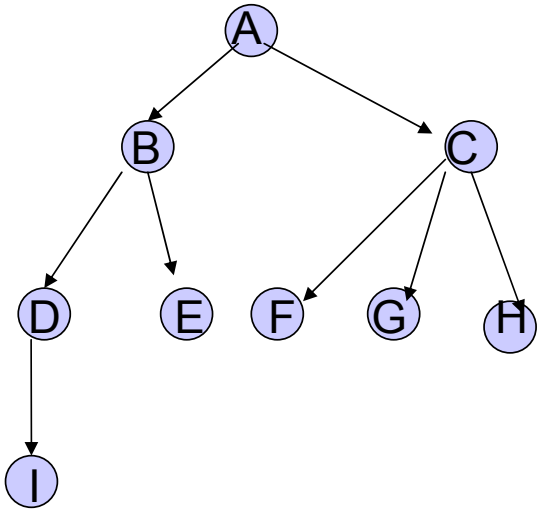
CON DIVERSO NUMERO DI PUNTATORI

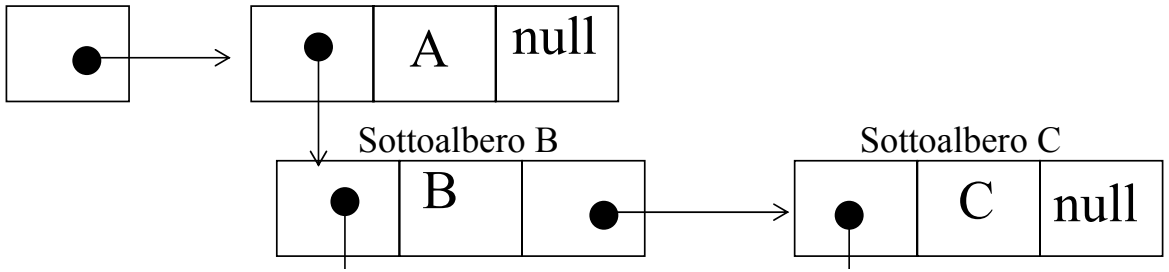


Ingestibile

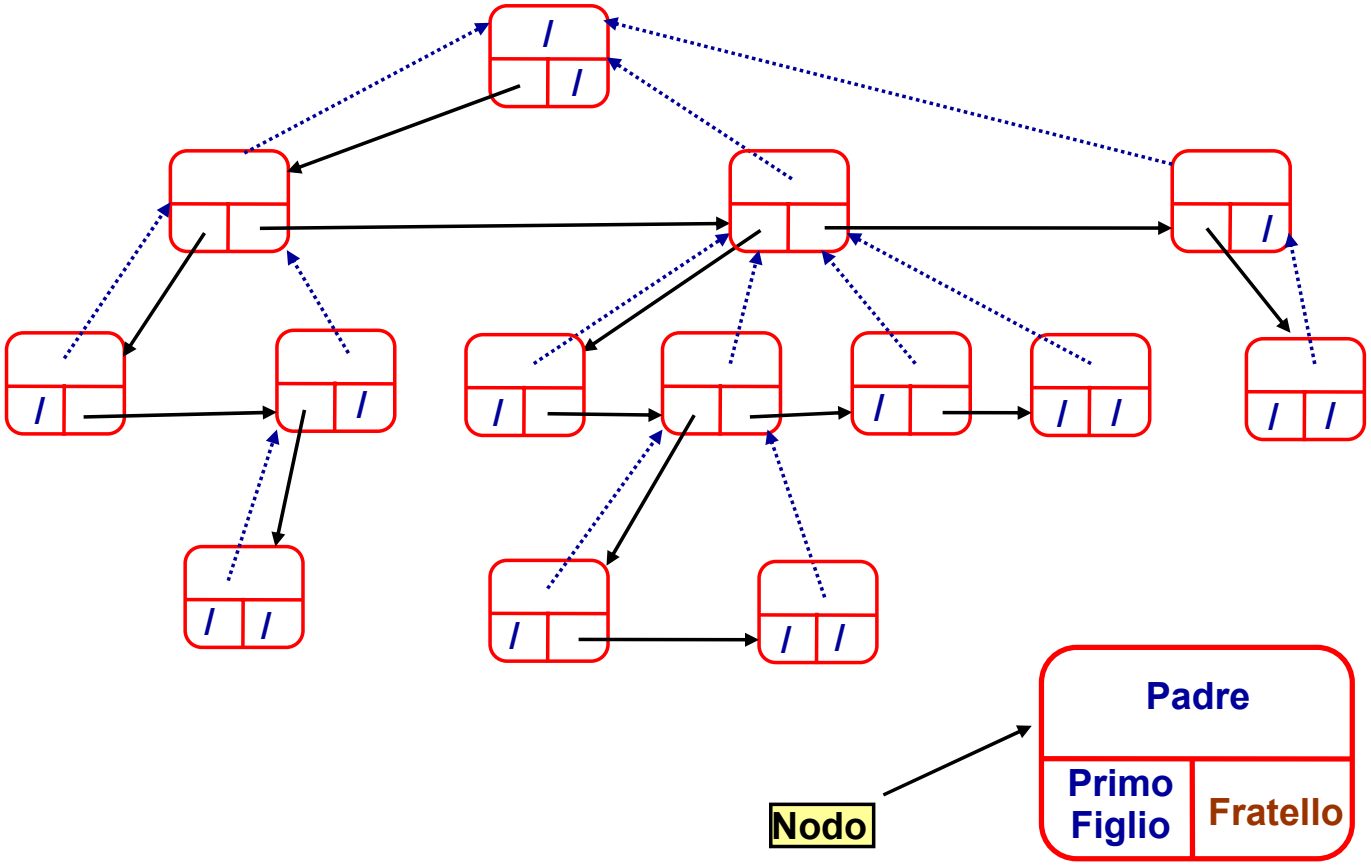
RAPPRESENTAZIONE CON 2 PUNTATORI

- Il nodo è una struttura con **3 campi**:
  - Informazione
  - Puntatore al **primo figlio**
  - Puntatore al **fratello destro**





RAPPRESENTAZIONE CON 3 PUNTATORI



STRUTTURA DATI

```
#ifndef NTree_h
#define NTree_h

#include <stdio.h>
#include<stdlib.h>

struct nodoalberon{
    struct nodoalberon *fratello;
    int elem;
    struct nodoalberon *primofiglio;
};

typedef struct nodoalberon TREENODEN;

typedef TREENODEN *ALBERONARIO;

#endif /* NTree_h */
```

FUNZIONI BASE (1/2)

```
/*costruzione di un albero nario inserito da input*/
ALBERONARIO creaalbero();

/*stampa in inorder i valori contenuti nell'albero*/
void stampapreordinealberon(ALBERONARIO);

/*calcola il massimo valore contenuto in un albero n-ario non vuoto*/
int maxalberon(ALBERONARIO);

/*verifica se un valore intero specificato sia o meno presente nell'albero */
int findalberon(ALBERONARIO, int);

/* verifico se tutti gli elementi di un albero n-ario sono pari*/
int tutti_pari_P (ALBERONARIO albero);
```

FUNZIONI BASE (2/2)

```
/* calcola l'altezza dell'albero*/
int altezza(ALBERONARIO T)

/* calcola il numero di foglie dell'albero generico*/
int numfoglie (ALBERONARIO T);

/* conta il numero di nodi con k figli */
int nodiKfigli (ALBERONARIO T, int k)

/* verifica se tutti i nodi pari hanno un numero dispari di figli */
int paridispari (ALBERONARIO T)
```

FUNZIONE 1

Si scriva una funzione che prende da Input coppie `<valore, numero di figli>` e costruisce un albero in `preordine`

```
/*Pre: l'albero che si costruisce e' NON vuoto*/

/*Post: costruisce in preordine un albero n-ario*/
```

FUNZIONE 1

```
ALBERONARIO creaalbero() {

    ALBERONARIO T, temp;

    int i,j,k;

    scanf("%d %d",&i,&j);

    T=malloc(sizeof(TREENODEN));

    if (T==NULL)
        printf("memoria non allocata\n");
```

FUNZIONE 1

```
else
    {
        T->elem = i;
        T->fratello = NULL;

        if (j==0)
            T->primofiglio = NULL;
        else
           {
            T->primofiglio = creaalbero();
```

FUNZIONE 1

```
            temp=T->primofiglio;

            for (k=0;k<j-1;k++)
            {
                /* printf("entrato nel ciclo con i pari a %d\n",i);*/
                temp->fratello=creaalbero();

                /*printf("valore mio e del fratello %d %d\n",
                           temp->elem, temp->fratello->elem);*/
                temp=temp->fratello;
            }
          }
        }
        return T;
}
```

FUNZIONE 2

Si scriva una procedura che dato un albero n-ario lo stampi in `preordine`

```
/*Post: stampa in preorder (prima la radice quindi il primo dei suoi
  sottoalberi, quindi il secondo dei suoi sottoalberi etc... i valori
  contenuti nei nodi dell'albero*/
```

FUNZIONE 2

```
void stampapreordinealberon(ALBERONARIO T)
{
    ALBERONARIO temp;
    if (T!=NULL)
    {
        printf("%d\n",T->elem);
        temp=T->primofiglio;
        while (temp!=NULL) {
            stampapreordinealberon(temp);
            temp=temp->fratello;
        }
    }
    return;
}
```

FUNZIONE 3

Si scriva una funzione che dato un albero n-ario calcoli il valore massimo contenuto nell'albero

```
/* Pre: albero non vuoto */
/* Post: restituisce il valore massimo contenuto nell'albero */
```

FUNZIONE 3

```
int maxalberon(ALBERONARIO T) {

    ALBERONARIO temp=T->primofiglio;
    int massimo = T->elem;
    int currmax;

    while (temp!=NULL){
        currmax=maxalberon(temp);

        if (currmax>massimo)
                massimo=currmax;

        temp=temp->fratello;
        }

    return massimo;

}
```

FUNZIONE 4

Si scriva una funzione che dato un albero n-ario ed un intero x verifichi se x compare tra i valori contenuti nell'albero

```
/*Post: restituisce i se il valore x e' contenuto nell'albero, 0 altrimenti */
```

FUNZIONE 4

```
int findalberon(ALBERONARIO T, int x) {

    ALBERONARIO temp;
    int trovato;
    if (T==NULL)  return 0;
    if (T->elem==x)  return 1;
    temp=T->primofiglio;
    trovato=0;

    while ((temp!=NULL)&& !trovato){
        if (findalberon(temp,x))
            trovato=1;
            temp=temp->fratello;
        }
    return trovato;
}
```

FUNZIONE 5

Si scriva una funzione che dato un albero n-ario verifichi se tutti i valori dei suoi nodi sono pari

FUNZIONE 5

```
/* verifico se tutti gli elementi di un albero n-ario sono pari*/
int tutti_pari_P (ALBERONARIO albero) {

    if (!albero) return 1;

    if (albero->elem % 2) return 0;

    else
        return (tutti_pari_P(albero->primofiglio)
                &&  tutti_pari_P(albero->fratello));
}
```

FUNZIONE 6

Si scriva una funzione che dato un albero n-ario calcoli la sua altezza

```
/* Post: restituisce l'altezza dell'albero generico */
```

FUNZIONE 6

```
int altezza(ALBERONARIO T) {
    ALBERONARIO temp;
    int h=0;
    int curralt;
    if (T==NULL) return 0;
    temp=T->primofiglio;

    while (temp!=NULL) {
        curralt=altezza(temp);

        if (curralt > h) h=curralt;

        temp=temp->fratello;
        }
    return (h+1);
}
```

FUNZIONE 7

Si scriva una funzione che dato un albero n-ario calcoli Il numero delle sue foglie

FUNZIONE 7

```
/* calcola il numero di foglie dell'albero generico*/

int numfoglie (ALBERONARIO T){
    if (T==NULL) return 0;

    if (T->primofiglio ==NULL)
        return 1+ numfoglie(T->fratello);

    else
        return (numfoglie (T->primofiglio)+
                numfoglie(T->fratello));
}
```

FUNZIONE 8

Si scriva una funzione che dato un albero n-ario calcoli il numero di nodi con k figli

FUNZIONE 8

```
int nodiKfigli (ALBERONARIO T, int k) {
    int k1=0;
    ALBERONARIO temp;
    if (T == NULL) return 0;
    else {
        temp=T->primofiglio;
        while (temp!=NULL){
            k1++;
            temp = temp->fratello;
        }
        if (k==k1)
            return 1+nodiKfigli(T->fratello,k)+ nodiKfigli(T->primofiglio,k);
        else
            return nodiKfigli(T->fratello,k)+nodiKfigli(T->primofiglio,k);

        }

    }
```

FUNZIONE 9

Scrivere funzione che dato un albero N-ario verifichi se ogni nodo pari ha solo figli dispari

FUNZIONE 9

```
int paridispari (ALBERONARIO T) {

    ALBERONARIO temp;
    int ris = 1;

    if (T == NULL) return 1;

    if (T->elem % 2) { /* caso in cui il nodo è dispari */
        temp = T->primofiglio;
        while (temp != NULL) {
            ris = ris && paridispari(temp);
            temp = temp->fratello;
        }
        return ris;
    }
```

FUNZIONE 9

```
 else { /* caso in cui il nodo radice è pari */
        temp = T->primofiglio;

        while (temp != NULL){
            ris = ris && (temp->elem % 2) && paridispari(temp);
            temp = temp->fratello;
        }

        return ris;
    }

}
```