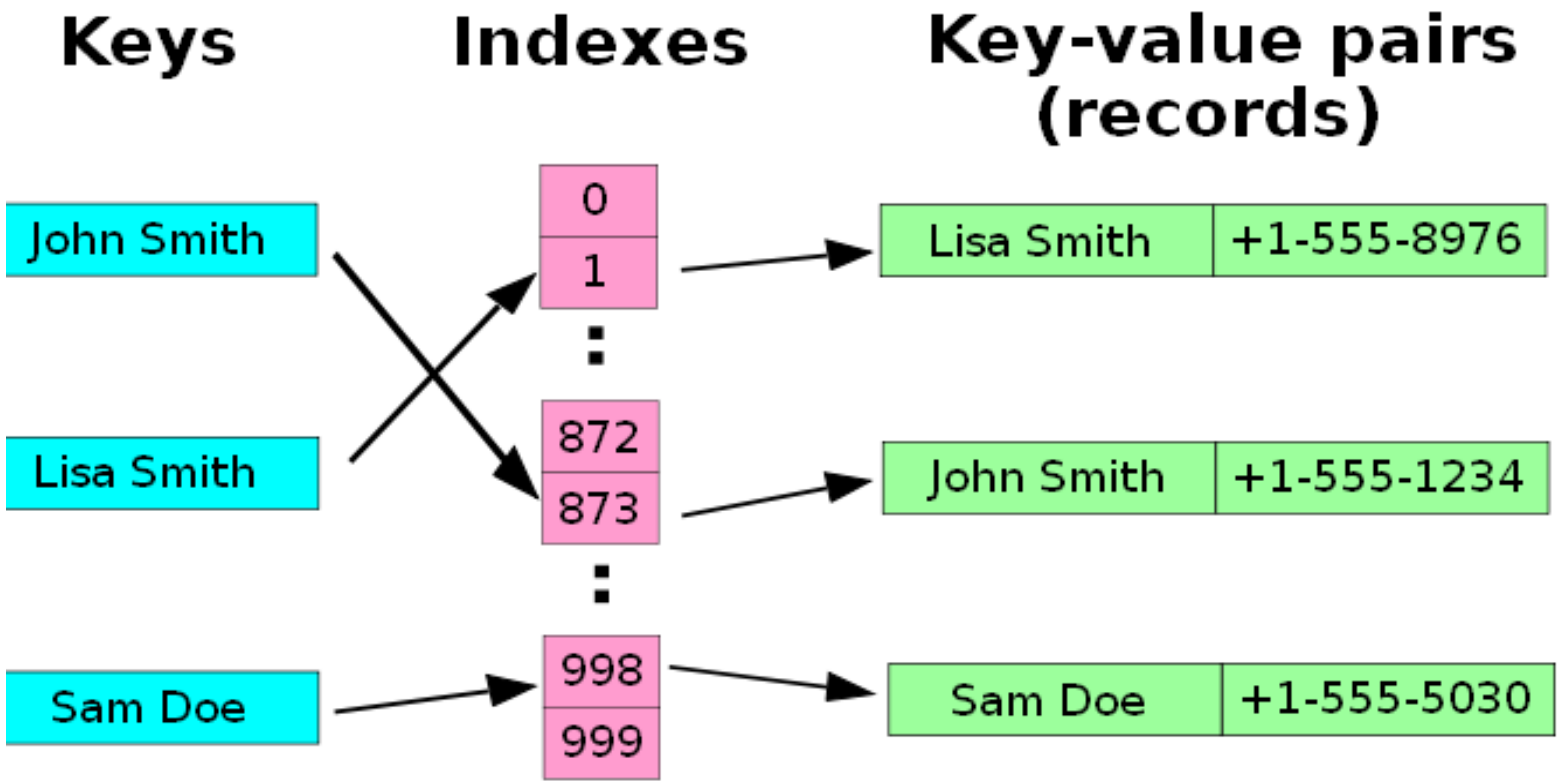
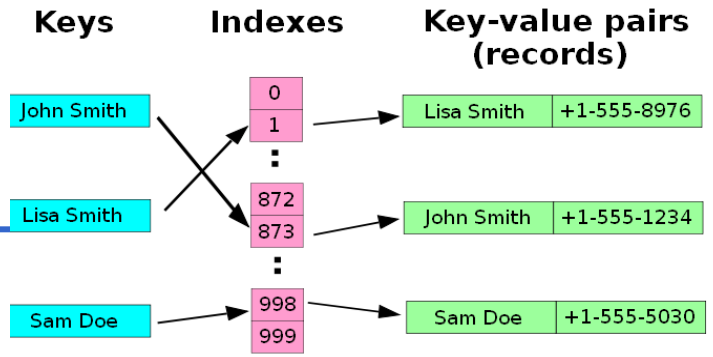
Indice come albero binario di ricerca

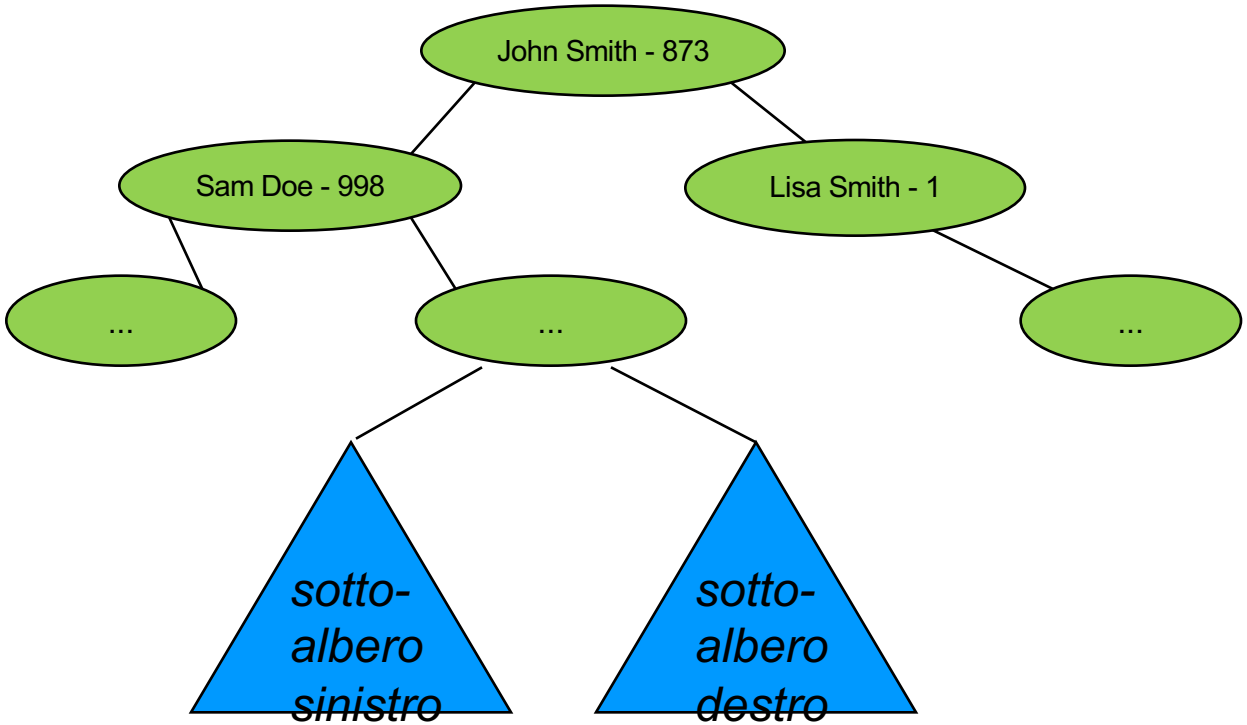
- Obiettivo:
  - Utilizzare una struttura dati del tipo albero binario di ricerca per realizzare un indice su file
  - Ricerca sull'indice in memoria centrale, poi accesso diretto alla posizione in memoria secondari (file)

Esempio indice - primary key:
cognome nome



Esempio indice

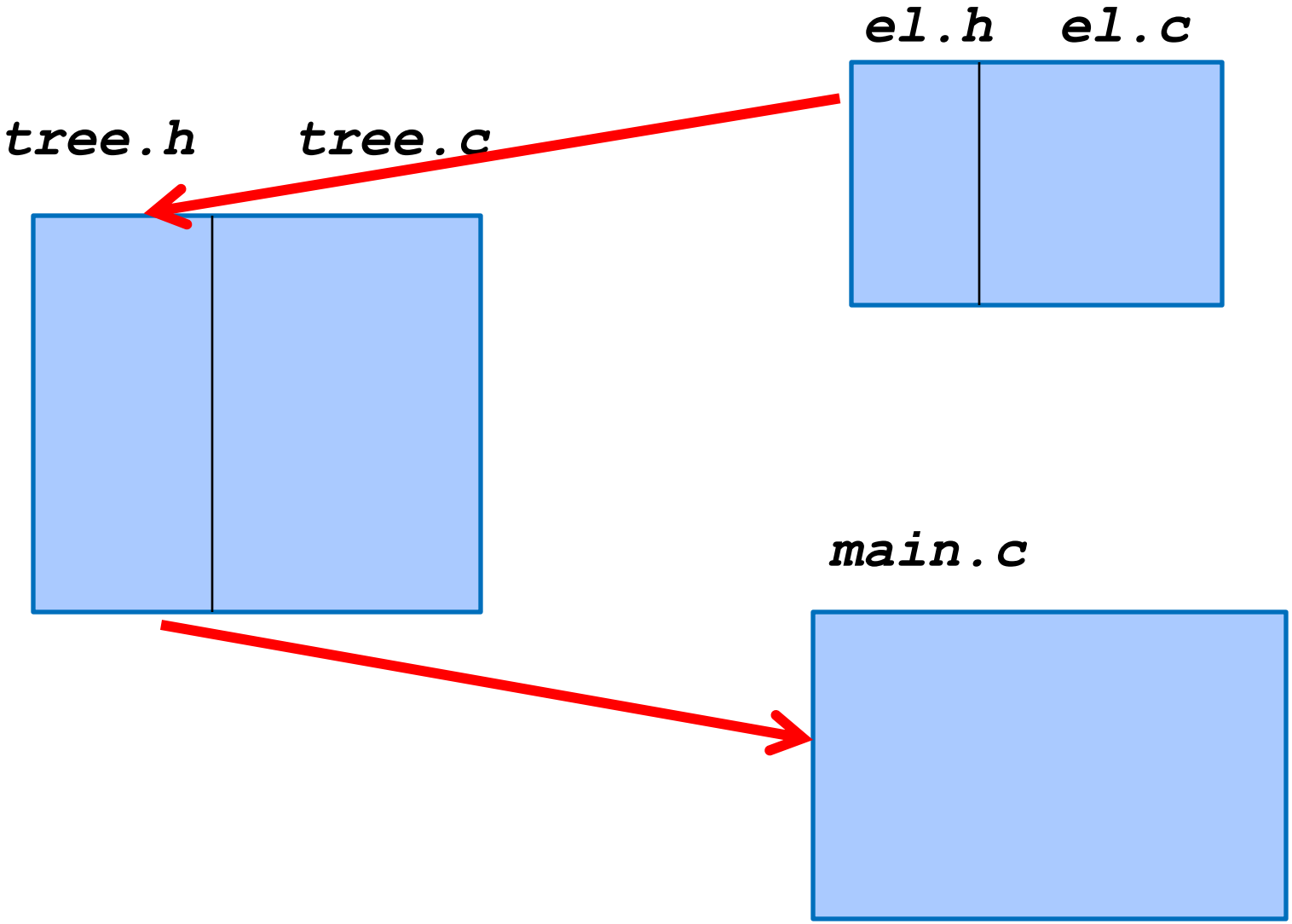




Esercizio: indice come BST

- Un file binario ad accesso diretto (PERSONE.DAT) contiene un elenco di informazioni su persone (cognome, nome, data di nascita, città di residenza). Il cognome di ciascuna persona costituisce la ***chiave unica*** di tale elenco. Per effettuare ricerche su questo elenco, si costruisca un ***albero binario di ricerca mantenendo, per ciascun cognome il numero di record corrispondente***. Si definisca un programma C che:
  a) costruisca la strutture dati in memoria principale a partire dal file PERSONE.DAT e ***stampi l'elenco ordinato dei cognomi***;
  b) calcoli ***l'altezza dell'albero*** così ottenuto e il ***numero dei suoi nodi***;
  c) letto un cognome a terminale, verifichi se esiste un elemento nell'indice un elemento con quella chiave e trovatolo nell'albero, acceda al file e legga il record corrispondente, visualizzando i campi nome, datan e città;
  d) letto un cognome a terminale, stampi ordinatamente a video i cognomi minori o uguali di quello letto.

COMPONENTI



ADT ELEMENT: el.h

```
/* ELEMENT TYPE - file el.h*/
typedef struct
      {     char nome[15];
            char cognome[15];
            char datan[7];
            char citta[15]; } record_type;

typedef struct {  char cognome[15];
                  int pos;} el_type;

typedef enum {false,true} boolean;

/* operatori esportabili */
boolean isequal(el_type, el_type);
boolean isless(el_type, el_type);
void showel(el_type);
```

ADT ELEMENT: el.c

```
/* ELEMENT TYPE - file el.c*/
#include <stdio.h>
#include <string.h>
#include "el.h"

boolean isequal(el_type e1, el_type e2)
/* due elementi uguali se stesso cognome */
{  if (strcmp(e1.cognome,e2.cognome)==0)
       return true;
    else return false; }

boolean isless(el_type e1, el_type e2)
{   if (strcmp(e1.cognome,e2.cognome) <0)
        return true;
    else  return false; }

void showel (el_type e)
{   printf("%s\t%d\n",e.cognome,e.pos); }
```

ADT TREE: tree.h

```
/* TREE INTERFACE  - file tree.h*/
#include "el.h“
typedef struct nodo { el_type value;
                      struct nodo *left, *right; } NODO;
typedef NODO * tree;
/* operatori esportabili */
tree  cons_tree(el_type e, tree l, tree r);
tree ord_ins(el_type e, tree t);
void inorder(tree t);
int height (tree t);
int nodi (tree t);
int member_ord(el_type e, tree t); /*NOTA TIPO RESTITUITO */
void stampa_minori(tree t, el_type e);
int max_int(int, int);
```

TO DO

Si modifichino i file *main.c e tree.c già disponibili (vedi lezioni e esercitazioni precedenti)* implementando le operazioni richieste in funzione di quelle esportate dall'ADT degli elementi

Il *main* da realizzare deve: a) leggere il contenuto del file ricordando il numero del blocco letto e inserendo chiave, posizione come elemento in un *albero binario di ricerca,* e stampare l'elenco ordinato dei cognomi;
b) calcolarne numero di nodi e altezza;
c) letto un cognome e accedendo all'albero recuperare le informazioni complete di quel cognome dal file;
d) stampare i cognomi minori di quello letto.

main.c *(1 - stub)*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "tree.h"
main() {
    tree indice=NULL;
    int pos;
    FILE *f;
    record_type REC;
    el_type EL;
    char COGNOME[15];

    f = fopen("PERSONE.DAT", "rb");
    if(f==NULL){ printf("errore apertura file\n");
                 exit(-1);       }
    pos=0; /* CICLO DI SCANSIONE DEL FILE */
    while(fread(&REC,sizeof(record_type),1,f)>0){
        // "definizione" campi dell'elemento EL
        // inserimento EL nell'albero  indice
        // da implementare
    }
```

main.c *(2 - stub)*

```
    fclose(f);
    printf("Chiusura del file\n");

    printf("Stampa dell'albero\n");
    inorder(indice);


    /* CALCOLO ALTEZZA E NUM.NODI DELL'ALBERO */

    printf("Altezza dell'albero: %d\n", height(indice));
    printf("Num. Nodi dell'albero: %d\n", nodi(indice));
```

TO DO – passo 1

Si modifichino i file *main.c e tree.c già disponibili (vedi lezioni e esercitazioni precedenti)* implementando le operazioni richieste in funzione di quelle esportate dall'ADT degli elementi

Il *main* da realizzare deve: a) leggere il contenuto del file ricordando il numero del blocco letto e inserendo chiave, posizione come elemento in un *albero binario di ricerca,* e stampare l'elenco ordinato dei cognomi;
b) calcolarne numero di nodi e altezza;
c) letto un cognome e accedendo all'albero recuperare le informazioni complete di quel cognome dal file;
d) stampare i cognomi minori di quello letto.

tree.c (1)

```
#include <stdlib.h>
#include "tree.h"

tree  cons_tree(el_type e, tree l, tree r)
/* costruisce un albero che ha nella
 radice e; per sottoalberi sinistro e
 destro l ed r rispettivamente    */
{ tree t;
  t = (NODO *) malloc(sizeof(NODO));
  t-> value = e;
  t-> left = l;
  t-> right = r;
  return (t);  }
```

tree.c (2)

```
void inorder(tree t)
{ if (t!=NULL)
      {  inorder(t->left);
         showel( t->value);
         inorder(t->right);     }
}
```

tree.c (ricorsiva)

```
tree ord_ins(el_type e, tree t)
/* albero binario di ricerca senza duplicazioni */
{ if (t==NULL)    /* inserimento in albero vuoto */
      return cons_tree(e,NULL,NULL);
  else
    { if (isless(e,t->value))
            t->left = ord_ins(e,t->left);
      else  t->right = ord_ins(e,t->right);
      return t;
    }
}
```

Inserimento iterativo

```
tree ordins_it(el_type e,tree root)
{tree p=NULL, t=root;
 if (root==NULL) return cons_tree(e,NULL,NULL);
 else
    { while (t!=NULL)
        if (isLess(e,t->value))
          {p=t;  t=t->left;}
        else
          {p=t;  t=t->right;}
    }                                    //p punta al padre
      if (isLess(e,p->value))
          p->left = cons_tree(e,NULL,NULL);
      else
          p->right = cons_tree(e,NULL,NULL); }
 return root;  }
```

main.c *(soluzione)*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "tree.h"
main() {
    tree indice=NULL;
    int pos;                FILE *f;
    record_type REC;        el_type EL;      char COGNOME[15];
    f = fopen("PERSONE.DAT", "rb");
    if(f==NULL){ printf("errore apertura file\n");
                 exit(-1);       }
    pos=0; /* CICLO DI SCANSIONE DEL FILE */
    while(fread(&REC,sizeof(record_type),1,f)>0)
     {  strcpy(EL.cognome,REC.cognome);
        EL.pos=pos;
        /* INSERIMENTO NELL'ALBERO */
        printf("Inserimento nell'albero di %s
         con posizione %d\n", EL.cognome, pos);
        indice = ord_ins(EL,indice);
        pos++;          }
```

main.c *(2 - soluzione)*

```
    fclose(f);
    printf("Chiusura del file\n");

    printf("Stampa dell'albero\n");
    inorder(indice);


    /* CALCOLO ALTEZZA E NUM.NODI DELL'ALBERO */

    printf("Altezza dell'albero: %d\n", height(indice));
    printf("Num. Nodi dell'albero: %d\n", nodi(indice));
```

TO DO – passo 2

Si modifichino i file *main.c e tree.c già disponibili*, implementando le operazioni richieste in funzione di quelle esportate dall'ADT degli elementi

Il *main* da realizzare deve:

a) leggere il contenuto del file ricordando il numero del blocco letto e inserendo chiave, posizione come elemento in un *albero binario di ricerca,* e stampare l'elenco ordinato dei cognomi;

b) calcolarne numero di nodi e altezza;

c) letto un cognome e accedendo all'albero recuperare le informazioni complete di quel cognome dal file;

d) stampare i cognomi minori di quello letto.

tree.c (7)

```
int nnodi(tree t)
{ if (t==NULL) return 0;
  else
   return (1+nnodi(t->left)+nnodi(t->right));
}
```

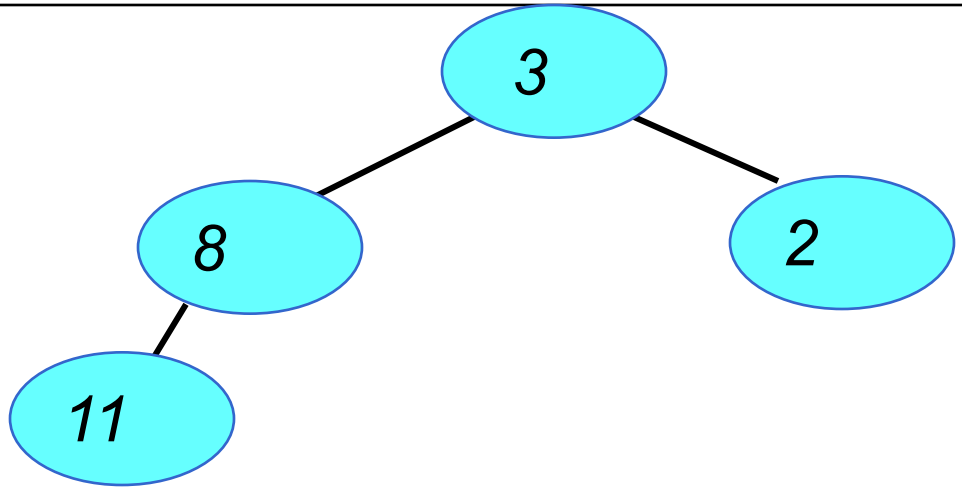
- E' tail ricorsiva?

To Do: altezza di un albero

- Altezza di un albero, lunghezza del cammino più lungo dalla radice a una delle foglie

Algoritmo:

- se l'albero è vuoto, o ha solo un nodo (radice) la sua altezza è 0
- altrimenti, è 1 +
  max(altezza del figlio sinistro , altezza del figlio destro)



tree.c (8 - height)

```
int height (tree t)
{ if (t==NULL) return 0;
   else  return max_int( height_aux(t->left),
                         height_aux(t->right)) ;
}

int height_aux(tree t)
{ if (t==NULL) return 0;
  else return (1 + max_int( height_aux(t->left),
                            height_aux(t->right) ));
}
```

- E' tail ricorsiva?

TO DO – passo 3

Si modifichino i file *main.c e tree.c già disponibili*, implementando le operazioni richieste in funzione di quelle esportate dall'ADT degli elementi

Il *main* da realizzare deve:

a) leggere il contenuto del file ricordando il numero del blocco letto e inserendo chiave, posizione come elemento in un *albero binario di ricerca,* e stampare l'elenco ordinato dei cognomi;

b) calcolarne numero di nodi e altezza;

c) letto un cognome e accedendo all'albero recuperare le informazioni complete di quel cognome dal file;

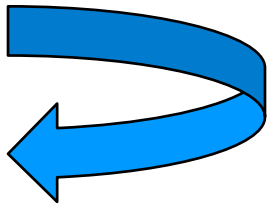
d) stampare i cognomi minori di quello letto.

main.c *(3 - stub)*

```
    /* VISUALIZZAZIONE RECORD */
     printf("Inserisci un cognome: ");
     scanf("%s", COGNOME);
     f = fopen("PERSONE.DAT", "rb");

     if(f==NULL){
         printf("errore apertura file\n");
         exit(-1);
     }

     // invocare opportunamente member_ord,
     // trovare il record sul file con fseek
     // e stampare risultato

     // invocare opportunamente stampa_minori
}
```

```
    /* VISUALIZZAZIONE RECORD */
     printf("Inserisci un cognome: ");
     scanf("%s", COGNOME);
     f = fopen("PERSONE.DAT", "rb");
         if(f==NULL){
         printf("errore apertura file\n");
         exit(-1);
     }
     strcpy(EL.cognome, COGNOME);
     pos = member_ord(EL, indice);

     if (pos>=0){
         fseek(f, pos*sizeof(record_type), SEEK_SET);
         if(fread(&REC,sizeof(record_type),1,f)>0){
             printf("Posizione %d:
                    Cognome:%s\tNome:%s\tDatan:
                    %s\tCitta:%s\n",
   pos,REC.cognome, REC.nome, REC.datan, REC.citta);
         }
         stampa_minori(indice, EL);
     }
}
```

```
    /* VISUALIZZAZIONE RECORD */
     printf("Inserisci un cognome: ");
     scanf("%s", COGNOME);
     f = fopen("PERSONE.DAT", "rb");
         if(f==NULL){
         printf("errore apertura file\n");
         exit(-1);
     }
     strcpy(EL.cognome, COGNOME);
     pos = member_ord(EL, indice);

     if (pos>=0){
         fseek(f, pos*sizeof(record_type), SEEK_SET);
         if(fread(&REC,sizeof(record_type),1,f)>0){
             printf("Posizione %d:
                    Cognome:%s\tNome:%s\tDatan:
                    %s\tCitta:%s\n",
   pos,REC.cognome, REC.nome, REC.datan, REC.citta);
         }
       //  stampa_minori(indice, EL);
     }
}
```



tree.c (7)

```
int member_ord(el_type e,tree t)
{ if (t==NULL) return -1;
  else
    if (isequal(e,t->value)
              return (t->value).pos;
    else
      if (isless(e,t->value)
            return member_ord(e,t->left);
      else return member_ord(e,t->right) ;
}
```

tree.c (6 - member_ord iterativa)

```
int  member_ord(el_type e, tree t)
{ while (t!=NULL){
        if (isequal(e,t->value))
                return (t->value).pos;
        else
             if (isless(e,t->value))
                        t=t->left;
             else
                        t=t->right;
     }
   return -1;
}
```

Esercizio

Si modifichino i file *main.c e tree.c già disponibili*, implementando le operazioni richieste in funzione di quelle esportate dall'ADT degli elementi

Il *main* da realizzare deve:

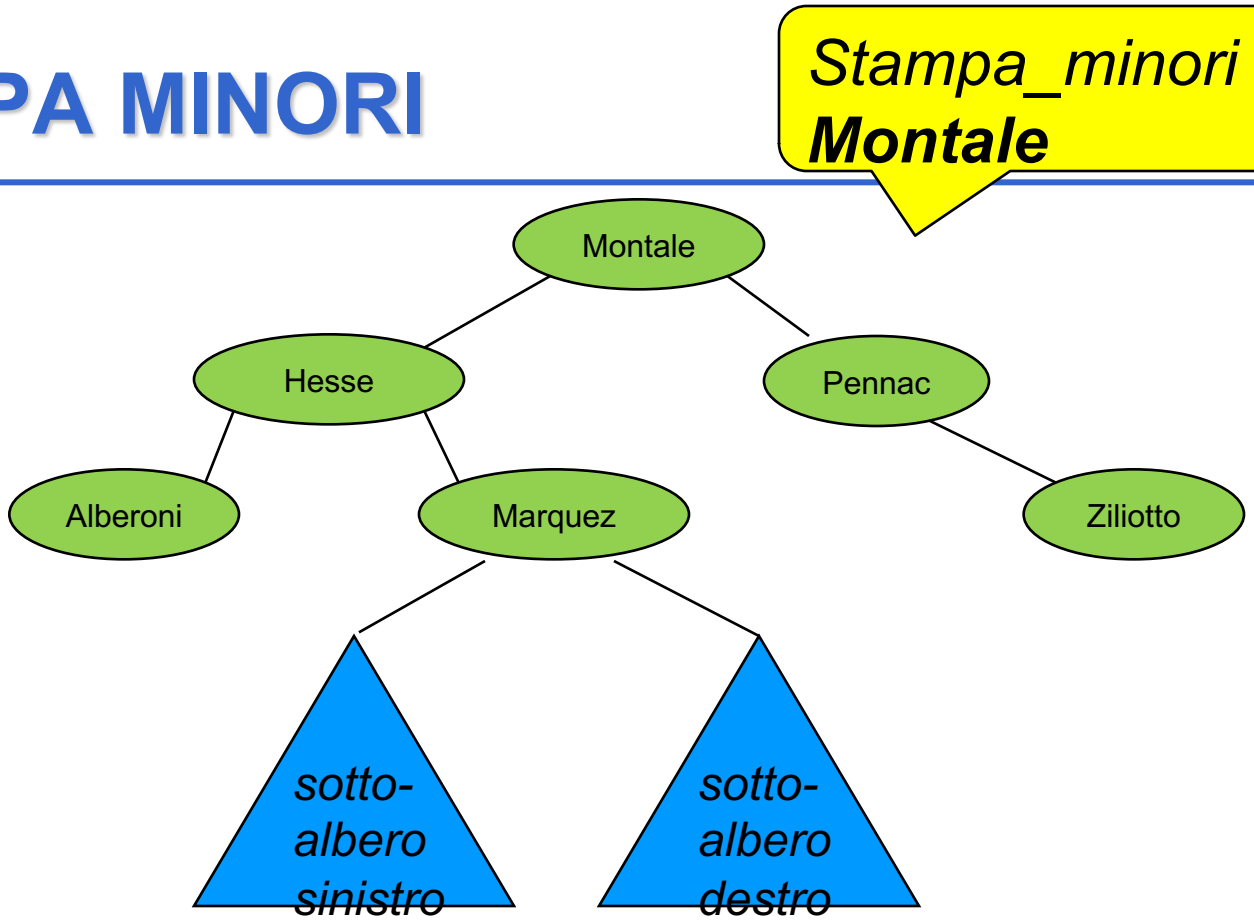
a) leggere il contenuto del file ricordando il numero del blocco letto e inserendo chiave, posizione come elemento in un *albero binario di ricerca,* e stampare l'elenco ordinato dei cognomi;

b) calcolarne numero di nodi e altezza;

c) letto un cognome e accedendo all'albero recuperare le informazioni complete di quel cognome dal file;

d) stampare i cognomi minori di quello letto.

STAMPA MINORI



- Modifica della visita inorder; stampo il contenuto del nodo solo se il cognome precede quello dato
- Non efficientissima (visito tutti i nodi così, anche quelli con cognome sicuramente maggiore)

tree.c

```
void  stampa_minori(tree t, el_type e)
/* ricorsiva - visita in ordine simmetrico */
{
  if (t!=NULL)

    { stampa_minori(t->left,e);
      if (isless(t->value,e))
                          showel(t->value);
      stampa_minori(t->right,e);
    }
}
```

Indice come BST – conclusioni

- Gli indici in memoria centrale a strutture dati in memoria di massa sono realizzati con alberi
- Caso BST è il più semplice
- Importanza del bilanciamento dell'albero, per BST bilanciato la ricerca binaria nel caso peggiore ha complessità $O(\log_2 N)$, dove N numero dei nodi