Laboratorio: Attività Preparatoria e Primi Esercizi

Sommario: 8 Marzo, 2019

- Esercizi Propedeutici:
  - Espressività del Linguaggio.
    Struttura di un Programma: Versionamento
  - Allocazione Dinamica
    Quando, Come, Perchè: Studio di Casi

Espressività del Linguaggio

- **Struttura di un Programma: Versionamento**.
  - Siamo Partiti da un L. di Programmazione e da un algoritmo, che ci sono stati "imposti", per:
    - Un problema a cui dare una soluzione automatizzata
    - Una funzione calcolabile di cui fornire un programma che la definisca
    - Una Computer Application da realizzare ed usare

    3 modi di "vedere" l'Ordinamento con QuickSort in C
  - Alla fine ciascuno di noi ha fornito un Programma in C che risponde alla richiesta.
  - Ottenendo ben 15 versioni diverse tra loro:
    - nei costrutti utilizzati per definire le principali fasi del procedimento;
    - nelle strutture dei dati utilizzati;
    - nello spazio dei nomi introdotti (variabili, costanti)
  - Cosa considerare per confrontare le diverse versioni e
    - individuare la migliore ?
    - e/o migliorare ancora la versione scelta?

Attività

- **Discutere ed Individuare dei criteri di confronto;**.
  soluzione (discussa in aula)
  Espressività intesa come Uso dei Costrutti del Linguaggio
  - piú adatti a esprimere il comportamento voluto in ogni sezione del programma
  - Uso dei Costrutti per ridurre il ricorso a vincoli di uso del programma

  Leggibilità intesa come capacità di:
  - mostrare le fasi essenziali del procedimento (stiamo usando programmazione prescrittiva)
  - localizzare il codice di ogni fase in procedure;
  - usare una struttura di codice gerarchica che incapsuli i dettagli e li mostri solo nel livello adatto.
- **Applicare i criteri di confronto scelti alle versioni QSort, QSort2, QSort3, QSort4 e QSort5** (vedi ListingA2.1 e slides successive)
  soluzione discussa in aula ...
- **Discutere possibili miglioramenti della versione individuata come migliore.**
  soluzione discussa in aula ...

Attività: Confronto QSort vs. QSort2 (a destra)

```c
/*
 Esercizio L1.1 del 28/2/2019.
 Scrivere un programma C che calcoli la funzione "ordinamento di sequenza"
 implementando l'algoritmo QuickSort. Allo scopo, si completi il programma
 delle operazioni per lettura e stampa di sequenze e si aggiungano gli even-
 tuali vincoli che si ritengano necessari.
 **
 * Marco Bellia
 **
 Soluzione.
 Vincoli.
 1) Esistenza bound, N, alla dimensione della sequenza da ordinare. Il bound è
    trattato come Parametro di Programma.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 3) Lettura numero elementi K (≤N) effettivi dela sequenza (da buffer di input)
 4) Lettura elementi sequenza di sequito a lettura di valore K (da buffer input)
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.
 **/

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

const int N = 15; // Parametro di programma (non modificabile)
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<J){
            int scambia=A[I];
            A[I]=A[J];
            A[J]=scambia;
            }
        if (I<=J){
            I++; J--;
            }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    // Allocaione Sequenza da ordinare
    int Seq[N];
    int k;
    //lettura sequenza k<=N interi da ordinare;
    int i=0;
    while (i<N && scanf("%d",&Seq[i])==1)i++;
    k=i-1;
     //ordinamento QuickSort
    QuickSort(0,k,Seq);
    //stampa sequenza ordinata;
    for (int i=0; i<k; i++) printf("%d,",Seq[i]);
    printf("%d\n",Seq[k]);
    return(1);
}
```

```c
 Soluzione.
 Vincoli.
 QSort2 migliora QSort nell'espressività riducendo i vincoli di uso.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 3) Lettura numero elementi K effettivi dela sequenza (da buffer di input)
 4) Lettura elementi sequenza di seguito a lettura di valore K (da buffer input)
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort2 migliora QSort nella scrittura del controllo di sequenza per lo scambio dei
      valori in QuickSort
 */

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

void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici crescenti in [Left..Right]
    int J=Right; //indici decrescenti in [Left..Right]
    int temp=A[(Left+Right)/2];  //elemento di separazione
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            int scambia=A[I];
            A[I]=A[J];
            A[J]=scambia;
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    // Allocaione Sequenza da ordinare
    int *Seq;
    int k;
    //lettura dimensione sequenza;
    printf("dimensione e sequenza: ");
    scanf("%d\n",&k);
    Seq = malloc(k*sizeof(int));
    for(int i; i<k; i++) scanf("%d",&Seq[i]);
    QuickSort(0,k-1,Seq);
    //stampa sequenza ordinata;
    for (int i=0; i<k; i++) printf("%d,",Seq[i]);
    printf("\n");
    return(1);
}
```

Attività: Confronto QSort2 vs. QSort3 (a destra)

```c
 Soluzione.
 Vincoli.
 QSort2 migliora QSort nell'espressività riducendo i vincoli di uso.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 3) Lettura numero elementi K effettivi dela sequenza (da buffer di input)
 4) Lettura elementi sequenza di seguito a lettura di valore K (da buffer input)
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort2 migliora QSort nella scrittura del controllo di sequenza per lo scambio dei
      valori in QuickSort
 */

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

void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici crescenti in [Left..Right]
    int J=Right; //indici decrescenti in [Left..Right]
    int temp=A[(Left+Right)/2];  //elemento di separazione
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            int scambia=A[I];
            A[I]=A[J];
            A[J]=scambia;
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    // Allocaione Sequenza da ordinare
    int *Seq;
    int k;
    //lettura dimensione sequenza;
    printf("dimensione e sequenza: ");
    scanf("%d\n",&k);
    Seq = malloc(k*sizeof(int));
    for(int i; i<k; i++) scanf("%d",&Seq[i]);
    QuickSort(0,k-1,Seq);
    //stampa sequenza ordinata;
    for (int i=0; i<k; i++) printf("%d,",Seq[i]);
    printf("\n");
    return(1);
}
```

```c
 Soluzione.
 Vincoli.
 QSort3 e QSort2: Differente scelta nei vincoli sulla sequenza da leggere
 1) Esistenza bound, N, alla dimensione della sequenza da ordinare. Il
    bound è trattato come Parametro di Programma.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort3 migliora QSort2 nella scrittura della struttura del programma in-
 dividuando alcune funzionalità e incapsulandole in procedure indipendendi:
 Scambia, leggiSeq, stampaSeq, QuickSort
 Ciò rende il comportamento del main molto più chiaro, e la veifica, modifi-
 ca del programma più semplice
*/

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

const int N = 15; // Parametro di programma (non modificabile)
void Scambia(int *x, int *y){
    int scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(int Seq[], int *upIndex){
    int i = 0;
    while (i<N && scanf("%d",&Seq[i])==1)i++;
    *upIndex = i-1;
}
void stampaSeq(int Seq[], int upIndex){
    for (int i=0; i<upIndex; i++) printf("%d,",Seq[i]);
    printf("%d\n",Seq[upIndex]);
}
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    int Seq[N];
    int k;
    leggiSeq(Seq,&k);     //lettura sequenza
    QuickSort(0,k,Seq);   //ordinamento
    stampaSeq(Seq,k);     //stampa sequenza
    return(1);
}
```

Attività: Confronto QSort3 vs. QSort4 (a destra)

```c
 Soluzione.
 Vincoli.
 QSort3 e QSort2: Differente scelta nei vincoli sulla sequenza da leggere
 1) Esistenza bound, N, alla dimensione della sequenza da ordinare. Il
    bound è trattato come Parametro di Programma.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort3 migliora QSort2 nella scrittura della struttura del programma in-
 dividuando alcune funzionalità e incapsulandole in procedure indipendendi:
 Scambia, leggiSeq, stampaSeq, QuickSort
 Ciò rende il comportamento del main molto più chiaro, e la veifica, modifi-
 ca del programma più semplice
*/

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

const int N = 15; // Parametro di programma (non modificabile)
void Scambia(int *x, int *y){
    int scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(int Seq[], int *upIndex){
    int i = 0;
    while (i<N && scanf("%d",&Seq[i])==1)i++;
    *upIndex = i-1;
}
void stampaSeq(int Seq[], int upIndex){
    for (int i=0; i<upIndex; i++) printf("%d,",Seq[i]);
    printf("%d\n",Seq[upIndex]);
}
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}

int main(void){
    int Seq[N];
    int k;
    leggiSeq(Seq,&k);     //lettura sequenza
    QuickSort(0,k,Seq);   //ordinamento
    stampaSeq(Seq,k);     //stampa sequenza
    return(1);
}
```

```c
 Soluzione.
 Vincoli.
 QSort4 migliora QSort3 nell'espressività riducendo i vincoli di uso.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort4 mantiene la struttura di QSort3.
 Ciò rende il comportamento del main molto più chiaro, e la veifica, modifi-
 ca del programma più semplice: In particolare, il miglioramento è dovuto al
 semplice di rimpiazzamento di leggiSeq con una versione migliore.
*/

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

void Scambia(int *x, int *y){
    int scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(int **rSeq, int *upIndex){
    int maxdim = 8; // potenza del 2
    int i = 0;
    int *Seq;
    Seq = malloc(maxdim*sizeof(int));
    int temp;
    while (scanf("%d",&temp)==1){
        if (i==maxdim){
            maxdim = 2 * maxdim;
            Seq = realloc(Seq,maxdim*sizeof(int));
        }
        Seq[i] = temp;
        i++;
    }
    *upIndex = i;
    *rSeq = Seq;
}
void stampaSeq(int *Seq, int upIndex){
    for (int i=0; i<upIndex; i++) printf("%d,",Seq[i]);
    printf("end.\n");
}
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}
int main(void){
    int *Seq;
    int k;
    leggiSeq(&Seq,&k);    //lettura sequenza
    QuickSort(0,k-1,Seq); //ordinamento
    stampaSeq(Seq,k);     //stampa sequenza
    return(1);
}
```

Attività: Confronto QSort4 vs. Magic QSort5 (a destra)

```c
 Soluzione.
 Vincoli.
 QSort4 migliora QSort3 nell'espressività riducendo i vincoli di uso.
 2) Elementi della sequenza di tipo noto. Il tipo è int.
 5) Relazione d'ordine assunta come operazione nota. Relazione < su int.

 Versionamento
 QSort4 mantiene la struttura di QSort3.
 Ciò rende il comportamento del main molto più chiaro, e la veifica, modifi-
 ca del programma più semplice: In particolare, il miglioramento è dovuto al
 semplice di rimpiazzamento di leggiSeq con una versione migliore.
*/

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

void Scambia(int *x, int *y){
    int scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(int **rSeq, int *upIndex){
    int maxdim = 8; // potenza del 2
    int i = 0;
    int *Seq;
    Seq = malloc(maxdim*sizeof(int));
    int temp;
    while (scanf("%d",&temp)==1){
        if (i==maxdim){
            maxdim = 2 * maxdim;
            Seq = realloc(Seq,maxdim*sizeof(int));
        }
        Seq[i] = temp;
        i++;
    }
    *upIndex = i;
    *rSeq = Seq;
}
void stampaSeq(int *Seq, int upIndex){
    for (int i=0; i<upIndex; i++) printf("%d,",Seq[i]);
    printf("end.\n");
}
void QuickSort(int Left, int Right, int A[]){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    int temp=A[(Left+Right)/2];
    while (I<=J) {
        while (A[I]<temp) I++;
        while (A[J]>temp) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}
int main(void){
    int *Seq;
    int k;
    leggiSeq(&Seq,&k);     //lettura sequenza
    QuickSort(0,k-1,Seq); //ordinamento
    stampaSeq(Seq,k);      //stampa sequenza
    return(1);
}
```

```c
 Soluzione
 Questo codice è "fasullo": C non ha i meccanismi coinvolti:
    Polimorfismo Generico
    Higher Order
 Esistono Linguaggi di Programmazione che hanno tutti i mecannismi necessari:
 Haskell, OCaml, C#, Java e molti altri

 Vincoli.
 QSort5 migliora QSort4 nell'espressività eliminando ogni vincoli di uso.

 Versionamento
 QSort5 mantiene la struttura di QSort4.
*/

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

void Scambia(T *x, T *y){
    T scambia = *x;
    *x = *y;
    *y = scambia;
}
void leggiSeq(T **rSeq, int *upIndex){
    int maxdim = 8; // potenza del 2
    int i = 0;
    T *Seq;
    Seq = malloc(maxdim*sizeof(T));
    T temp;
    while (scanf("????",&temp)==1){
        if (i==maxdim){
            maxdim = 2 * maxdim;
            Seq = realloc(Seq,maxdim*sizeof(T));
        }
        Seq[i] = temp;
        i++;
    }
    *upIndex = i;
    *rSeq = Seq;
}
void stampaSeq(T *Seq, int upIndex){
    for (int i=0; i<upIndex; i++) printf("???",Seq[i]);
    printf("end.\n");
}
void QuickSort(T Left, T Right, T A[], T (*ord) (T x, T y)){// procedura ricorsiva
    int I=Left;  //indici in [Left..Right]
    int J=Right; //indici in [Left..Right]
    T temp=A[(Left+Right)/2];
    while (I<=J) {
        while (ord(A[I],temp)) I++;
        while (ord(temp,A[J])) J--;
        if (I<=J){
            Scambia(&A[I],&A[J]);
            I++; J--;
        }
    }
    if (Left<J) QuickSort(Left,J,A);
    if (I<Right) QuickSort(I,Right,A);
}
int main(void){
    int *Seq;
    int k;
    leggiSeq(&Seq,&k);     //lettura sequenza
    QuickSort(0,k-1,Seq); //ordinamento
    stampaSeq(Seq,k);      //stampa sequenza
    return(1);
}
```

Esercizio (semplificato)

Scrivere un programma C che introduca una implementazione per i parseTree che abbiamo visto nella lezione del 6/3/2019. Allo scopo si fornisca l'implementazione di 2 nuovi tipi di dato e delle operazioni a loro applicabili ... Si completi il programma con opportuni casi di test.

Esercizio (Completo)

Scrivere un programma C che calcoli la funzione $\Rightarrow^{\texttt{Tree}}$ *(deriva) su parse tree, data una grammatica libera (vedi lucidi relativi). Allo scopo, si completi il testo con le assunzioni e i vincoli che si ritenga necessari apportare.*
Si fornisca infine, il codice per applicare il programma ad un caso concreto, quale la costruzione del parse tree ottenuto dalla derivazione dell'espressione $3 \star \texttt{x} + 10$ *con la grammatica vista a lezione, utilizzando l'opportuno lessico.*

Attività: Definizione, Progettazione e Implementazione

- **Riesaminare proprietà dei parseTree** Definizione, Uso, anche rispetto alle operazioni richieste (vedi ListingA2.2 e/o slide successiva)
  soluzione (discussa in aula)...

- **Definire una struttura per la rappresentazione dei ParseTree**. Definizione prima astratta, poi formale in C
  soluzione (discussa in aula)...

- **Definire ciascuna operazione richiesta.** Fornendo ed applicando le soluzioni parziali a casi di test durante l'intero sviluppo.
  soluzione da completare e discutere ...

Attività: Progett. e Implementaz. in C dei ParseTree

parseTree.h e' un modulo che fornisce l'implementazione di 3 nuovi tipi di dato e la specifica delle operazioni applicabili (Listing A2.2)

- Il tipo **parseTree** con le seguenti operazioni (di interfaccia):
  - makeEmpty
  - makeLeaf
  - makeRooted
  - isEmpty
  - isLeaf
  - isRooted
  - getLabel
  - getSons
  - printlnTree
- Il tipo **treeList** con le seguenti operazioni (di interfaccia)
  - nil
  - cons
  - split
- Il tipo **bool** senza operazioni con due valori: {`false`, `true`}

Attività: La struttura di Nodo proposta in parseTree.h

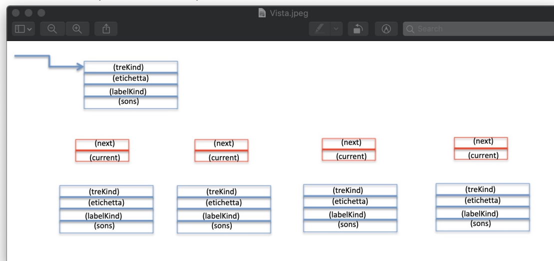
Il modulo parseTree.h introduce "parseTreeStruct" per rappresentare i nodi di un parseTree.

```
//Strutture per rappresentare alberi parseTree
struct parseTreeStruct{
    int treeKind; //0=leaf; 1=rooted; pointer null per emptyTree.
    char * etichetta;
    int labelKind; //0=NonTerminale; 1=Terminale.
    struct treeListStruct * sons;
};
struct treeListStruct{
    struct parseTreeStruct * currentSon;
    struct treeListStruct * next;
};

//Tipi per alberi, liste di alberi, booleani
typedef struct parseTreeStruct * parseTree;
typedef struct treeListStruct * treeList;
typedef enum{false,true} bool;
```

Il tipo **parseTree** con le seguenti operazioni (di interfaccia):

- makeEmpty
- makeLeaf
- makeRooted
- isEmpty
- isLeaf
- isRooted
- getLabel
- getSons
- printlnTree



Attività: Signature Operazioni parseTree.h

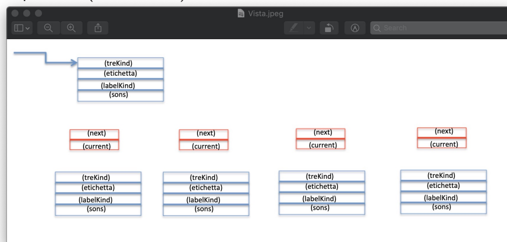
Il modulo parseTree.h fornisce la signature delle operazioni da definire sui parseTree.

```
//Le operazioni di parseTree
parseTree makeEmpty();
parseTree makeLeaf(tLabel labelKind, char * etichetta);
parseTree makeRooted(char * etichetta, treeList sons);
//Le operazioni: Ispettori e Selettori di parseTree
bool isEmpty(parseTree t);
bool isLeaf(parseTree t);
bool isRooted(parseTree t);
bool getLabel(parseTree t, char** r);
treeList getSons(parseTree t);
//-- operazioni per la stampa --
void printlnTree(parseTree t);

//Le operazioni: Costruttori di treeList
treeList nil();
treeList cons(parseTree val, treeList list);
//Le operazioni: Ispettori e Selettori di treeList
bool split(treeList list, parseTree * treeVal, treeList *
    listVal);
```

Il tipo **parseTree** con le seguenti operazioni (di interfaccia):

- makeEmpty
- makeLeaf
- makeRooted
- isEmpty
- isLeaf
- isRooted
- getLabel
- getSons
- printlnTree



Attività: Completiamo parseTree.c

- Inseriamo in parseTree.c le definizioni delle operazioni sui parseTree
- Queste operazioni hanno la segnatura definita nel modulo parseTree.h
- Queste operazioni devono creare alberi rappresentati come nella slide precedente.

```c
/*
 parseTree.c e' un modulo che fornisce l'implementazione di 3 nuovi tipi di dato.
 Il tipo parseTree con le seguenti operazioni (di interfaccia):
 - makeEmpty
 - makeLeaf
 - makeRooted
 - isEmpty
 - isLeaf
 - isRooted
 - getLabel
 - getSons
 - printTree
 Il tipo treeList con le seguenti operazioni (di interfaccia)
 - nil:
 - cons:
 - split
 Il tipo bool senza operazioni con due valori literal
 + false
 + true

 ** AUTHOR: Marco Bellia
 ** LPL - Matematica
 ** Pisa 2016

 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parseTree.h"

//Le operazioni: Costruttori di treeList
//scrivere qui sotto le definizioni di: nil, cons

//Le operazioni: Ispettori e Selettori di treeList
//scrivere qui sotto le definizioni di: isNull, hd, tail, split

//Le operazioni: Costruttori di parseTree
//scrivere qui sotto le definizioni di: makeEmpty, makeLeaf, makeRooted

//Le operazioni: Ispettori e Selettori di parseTree
//scrivere qui sotto le definizioni di: isEmpty, isLeaf, isRooted, getLabel, getSons

//-- operazioni per la stampa --
//scrivere qui sotto le definizioni di: printTree,....
```