# Laboratorio A2: Attività Preparatoria e Primi Esercizi

**Sommario: 13 Marzo, 2020**

- 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 alla versione QSort** (in ListingA2.1) e ad una vostra versione)
  **soluzione** da discutere in aula ...

- **Annotare i miglioramenti della versione individuata come migliore.**
  **soluzione** da discutere in aula ...????

# Allocazione Dinamica e Moduli in C

**Esercizio (semplificato)**

*Scrivere un programma C che introduca una implementazione per i parseTree che abbiamo visto nella lezione del Marzo 4, 2020. 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 Parse Tree**. 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

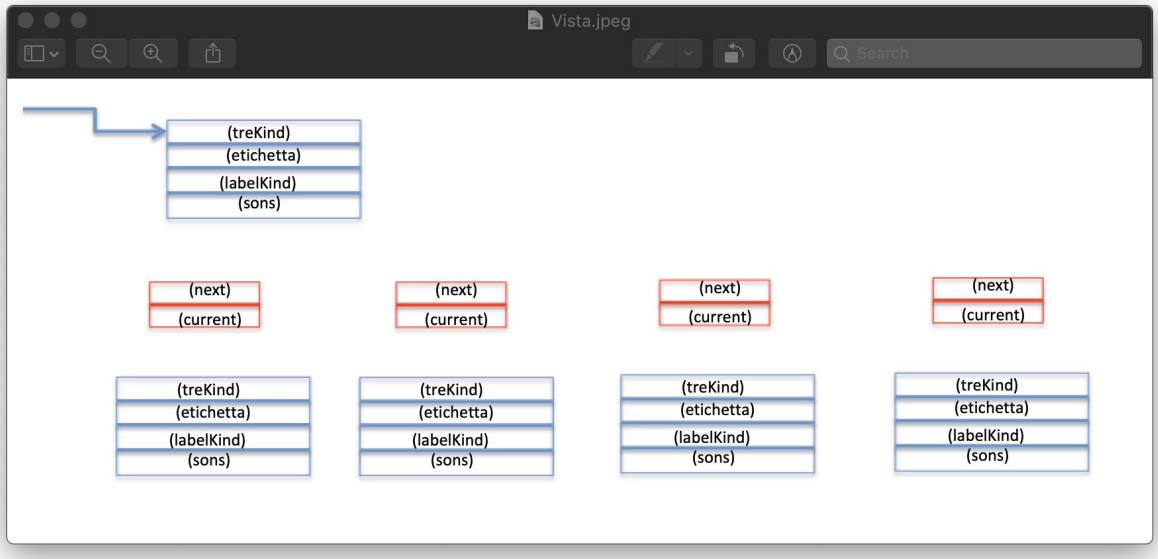
Il modulo parseTree.h introduce "parseTreeStruct" per rappresentare i nodi di un parseTree.

```
//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
- printInTree



# Attività: Signature Operazioni parseTree.h

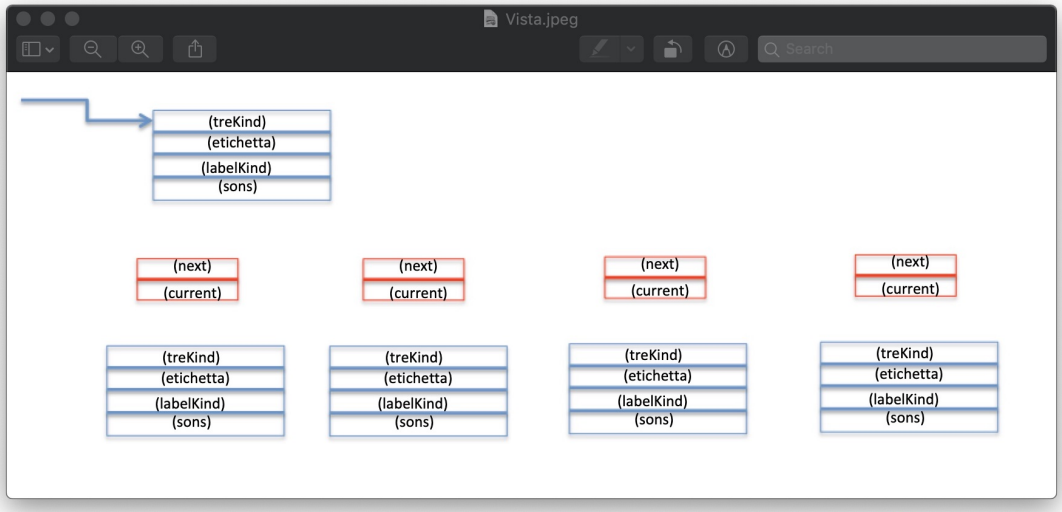
Il modulo parseTree.h fornisce la signature delle operazioni da definire sui parseTree.

```
//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.

```
/*
 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,....
```