# Ingegneria degli Algoritmi (A.A. 2014-2015)

Corsi di Laurea in Ingegneria Informatica e Automatica, Ingegneria dei Sistemi Informatici, e Laurea Magistrale in Ingegneria Informatica

*Sapienza Università di Roma*

**Esame di laboratorio (24/07/2015) – Durata 2h 00'**

---

**Albero binario di ricerca non bilanciato con elementi di tipo generico**

In un **albero binario di ricerca**, se x è l'elemento contenuto in un nodo, il sottoalbero sinistro di quel nodo contiene elementi minori (o uguali) a x, e il sottoalbero destro contiene elementi maggiori (o uguali) a x.

Si richiede di completare le funzioni di un albero binario di ricerca (non bilanciato) con elementi di tipo generico realizzato mediante strutture e puntatori in C, con i seguenti prototipi dichiarati nella header `stree.h`:

```
stree* stree_new(size_t elem_size,
                 int (*comp)(const void*, const void*));
```

Crea un albero di ricerca vuoto:

1. `elem_size`: numero di byte di un elemento dell'albero;
2. `comp`: puntatore a funzione comparatore per gli elementi dell'albero;
3. la funzione deve restituire il puntatore all'albero allocato.

```
void stree_delete(stree* q);
```

Dealloca un albero:

1. `t`: puntatore all'albero da deallocare.

```
int stree_isempty(const stree* q);
```

Verifica se l'albero è vuoto:

1. `t`: puntatore all'albero;
2. la funzione deve restituire 1 se l'albero è vuoto e 0 altrimenti.

```
void stree_insert (queue* q, void* elem);
```

Aggiunge un elemento nell'albero di ricerca:

3. `t`: puntatore all'albero;
4. `elem`: puntatore a un oggetto che contiene l'elemento da inserire.

```
int stree_find(const stree* t, const void* elem);
```

Cerca un elemento nell'albero [**da realizzare**]:

1. `t`: puntatore all'albero;
2. `elem`: puntatore a un oggetto che contiene l'elemento da cercare;
3. la funzione deve restituire 1 se l'elemento è presente, e 0 altrimenti.

```
void stree_clear(stree* t);
```

Rimuove tutti gli elementi dall'albero [**da realizzare**]:

- `t`: puntatore all'albero da svuotare.

Completare le funzioni `stree_find()` e `stree_clear()` nel file `stree.c`. E' utile esaminare il main di prova `main.c` per vedere un esempio di operazioni su un albero.

Completare inoltre la funzione comparatore `comp()` nel file `main.c` in modo che possa essere usata nel main di prova:

```
int comp(const void* a, const void* b)
```

Confronta due elementi generici [**da realizzare**]:

1. `a`: puntatore al primo elemento;
2. `b`: puntatore al secondo elemento;
3. la funzione deve restituire:
    a. -1 se `a` è minore di `b`
    b. 0 se `a` è uguale a `b`
    c. +1 se `a` è maggiore di `b`

**Compilazione e test.**

- *Compilazione programma di test*: dare il comando `make`, che genera il file eseguibile `stree` contenente un programma di test
- *Test correttezza*: eseguire il programma di test `./stree`.

E' consigliabile modificare il main per fare ulteriori test e verificare la correttezza delle funzioni scritte, purché non si modifichino i prototipi delle funzioni dichiarate in `stree.h`.

**Nota bene**: testare la correttezza dell'uso della memoria usando il tool Valgrind:

`valgrind ./stree`