

# Relazione Progetto Binary Search Tree

- **Nome:** Daniel
- **Cognome:** Mateus Cardona
- **Matricola:** 844615
- **Mail:** d.mateuscardona@campus.unimib.it

## Indice

[Indice](#)

[Classe BST](#)

[Struttura](#)

[Tipo di dato](#)

[Implementazione](#)

[Attributi principali](#)

[Funzionalità e metodi principali](#)

[Altre Scelte implementative](#)

[Valgrind e Make](#)

## Classe BST

La classe rappresenta un **albero binario di ricerca** (Binary Search Tree, BST) in C++.

La classe è implementata come un template, dove il parametro T definisce il tipo di dato dei nodi dell' albero. I funtori di confronto, equal e compare, garantiscono l'ordinamento degli elementi all'interno dell'albero.

## Struttura

L'albero è rappresentato tramite un **lista di nodi concatenati ordinata**.

Ogni nodo è una struct che contiene un dato di tipo T e puntatori ai nodi parent, sinistro e destro.

## Tipo di dato

Per rappresentare un nodo del BST è stata creata la struttura di supporto node:

```
typedef T value_type; ///< Type of values stored in the BST

struct node
    value_type value; ///< The value stored in the node
    node *parent;      ///< Pointer to the parent node
    node *left;        ///< Pointer to the left child node
    node *right;       ///< Pointer to the right child node
```

La struct è privata e definisce il tipo di dato contenuto nel BST. Contiene un valore generico T (che è il dato vero e proprio), e puntatori ad altri nodi per il collegamento / disposizione nell' albero.

## Implementazione

Per costruire un oggetto BST, è necessario specificare tre tipi:

1. T: Il tipo di valore da memorizzare nell'albero.
2. C: Un funtore di comparazione per ordinare gli elementi.
3. E: Un funtore di uguaglianza per confrontare gli elementi.

Questi sono gli argomenti necessari per mantenere la generalità della classe (dato che è templata).

Gli altri metodi fondamentali classici ne permettono invece la copia (cc), l'assegnament (operator=) e la distruzione.

## Attributi principali

Gli attributi principali della classe sono:

- `_root` : È la testa della lista concatenata di nodi. Rappresenta la radice dell'albero
- `_size`: Restituisce il numero di nodi presenti nell'albero
- `_compare`: Funtore che confronta due nodi e determina "chi diventa figlio sinistro (o destro) di chi."
- `_equals`: Funtore che stabilisce se due nodi sono uguali o meno



La policy di `compare` funziona secondo la logica di inserimento dei BST: se il funtore restituisce `true`, il secondo nodo passato come argomento è figlio sinistro del primo, altrimenti diventa figlio destro.

## Funzionalità e metodi principali

- Costruzione:
  - Costruttore di default: Crea un BST vuoto.
  - Costruttore da range: Costruisce un BST da una sequenza di elementi.
  - Costruttore di copia: Crea una copia profonda di un altro BST.
- Distruzione:
  - Il distruttore libera tutta la memoria allocata per i nodi dell'albero.
- Funzionalità principali:
  - Inserimento (`add`): Aggiunge un nuovo elemento all'albero mantenendo l'ordinamento.
  - Ricerca (`exists`): Verifica se un elemento è presente nell'albero.
  - Estrazione di sottoalbero (`subtree`): Crea un nuovo BST a partire da un nodo specifico.
  - `begin()` ed `end()`: Forniscono un iteratore costante per attraversare l'albero in ordine.

- Altre operazioni:
  - Operatore di assegnamento: Permette di copiare un BST in un altro.
  - Stampa (operator<<): Consente di stampare il contenuto dell'albero.
  - Stampa condizionale (printf): Stampa elementi che soddisfano una condizione specifica.
- Gestione degli errori:
  - Lancia eccezioni in caso di operazioni non valide (es. inserimento di duplicati).

## Altre Scelte implementative

- Implementazioni ricorsive vs iterative

Ho scelto (molto spesso) nel codice, di implementare i metodi nella loro versione ricorsiva e non iterativa. Questa scelta è stata fatta volontariamente per leggibilità di codice (soprattutto in fase di lettura). Non essendo specificato nel testo dell' esame di soffermarsi sulle prestazioni, ho deciso di sacrificare la prestazione dell' iterativo per l' eleganza ( mia personale opinione), compattezza e leggibilità del ricorsivo.

- Algoritmi ricorsivi "classici" per la gestione del BST

La maggior parte dei metodi si rifanno alla versione ricorsiva di algoritmi noti per la gestione dei BST. Nella mia implementazione ho seguito questi algoritmi, utilizzando occasionalmente metodi ausiliari per facilitare la comprensione del codice. Per quanto organizzate in maniera diversa, le logiche di implementazione non si discostano dalla letteratura sul BST.

- Gestione Eccezioni nel ricorsivo

Come spiegato nel corso, un ciclo di allocazioni in memoria dinamica deve portare il programmatore a prevederne il fallimento, gestire il data recovery e

catchare l' eccezione. Avendo prediletto il ricorsivo non mi è stato possibile gestire "in maniera classica" i cicli di allocazioni (non essendoci).

Ho comunque cercato di gestire le chiamate ricorsive: ho messo in parallelo un ciclo di iterazione con una sotto chiamata ricorsiva.

Ogni funzione ricorsiva nel codice è stata lasciata libera di propagare gli errori ed eccezioni, ed è stata "circondata" da try catch e relativa gestione **nel chiamante**.

Non è quindi possibile, con fallimenti parziali della ricorsività, non arrivare ad una gestione delle eccezioni. La responsabilità invece di fare questi controlli per i metodi ricorsivi pubblici invece è stata documentata e delegata al chiamante di tali metodi.

- Ordinamento in-order di lettura


Ho deciso di fare un attraversamento dell' albero in-order per la lettura (metodo privato goNext() del forward iterator). La scelta è stata casuale.

## Valgrind e Make

L' architettura su cui ho implementato il progetto è la seguente

```
...-!:::-...
.-MMMMMMMMMMMMMMMM-
.-MMMM`...:::-..`MMMM-
.:MMMM.:MMMMMMMMMMMMMMMM.:MMMM:
-MMM-M--MMMMMMMMMMMMMMMMMMMM.MMM-
`:MMM:MM` :MMM:.....-MMM:MM:`
:MMM:MM` :MM:` `` `` `:MMM:MM:
.MMM.MMMM` :MM. -MM. .MM- `MMM.MMM.
:MMM:MMM` :MM. -MM- .MM: `MMM-MMM:
:MMM:MMM` :MM. -MM- .MM: `MMM:MM:
:MMM:MMM` :MM. -MM- .MM: `MMM-MMM:
.MMM.MMMM` :MM.--:MM.--:MM: `MMM.MMM.
:MMM:MM- `MMMMMMMMMMMMMM-` -MM-MMM:
:MMM:MM:` `:MMM:MM:
.MMM.MMM:-----:MMM.MMM.
'-MMMM.-MMMMMMMMMMMMMM-.MMMM-'
'.-MMMM`--:::-`MMMM-.'
'-MMMMMMMMMMMMMM-'
`--:::-`
`--:::-`

daniel@daniel-Modern-14-B11MOU
-----
OS: Linux Mint 21.1 x86_64
Host: Modern 14 B11MOU REV:1.0
Kernel: 5.15.0-105-generic
Uptime: 2 hours, 50 mins
Packages: 2513 (dpkg), 4 (snap)
Shell: zsh 5.8.1
Resolution: 1920x1080, 1920x1080
DE: Cinnamon 5.6.8
WM: Muttter (Muffin)
WM Theme: Mint-Y-Dark-Purple (Mint-Y)
Theme: Mint-Y-Dark-Purple [GTK2/3]
Icons: Mint-Y-Dark-Purple [GTK2/3]
Terminal: vscode
CPU: 11th Gen Intel i3-1115G4 (4) @ 4.100GHz
GPU: Intel Device 9a78
Memory: 2052MiB / 7641MiB
```



Ed il risultato di Valgrind è il seguente:

```

[~/Scrivania/844615]
• daniel valgrind ./main.exe
==15810== Memcheck, a memory error detector
==15810== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15810== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15810== Command: ./main.exe
==15810==
Testing constructors...
Constructor tests passed!
Testing assignment operator...
Assignment operator test passed!
Testing add method...
Add method test passed!
Testing subtree method...
Subtree method test passed!
Testing iterators...
Iterator test passed!
Testing BST of BST...
BST of BST test passed!
Testing printIF function...
Original BST: 1 2 3 4 5 6 7 8 9 10
Even numbers: 2 4 6 8 10
Numbers greater than 5: 6 7 8 9 10
printIF test passed!
Testing complex person comparison...
Complex person comparison test passed!
Testing complex point comparison...
Complex point comparison test passed!
All tests passed successfully!
==15810==
==15810== HEAP SUMMARY:
==15810==    in use at exit: 0 bytes in 0 blocks
==15810== total heap usage: 138 allocs, 138 frees, 79,628 bytes allocated
==15810==
==15810== All heap blocks were freed -- no leaks are possible
==15810==
==15810== For lists of detected and suppressed errors, rerun with: -s
==15810== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Anche la Make funziona senza problemi:

```

[~/Scrivania/844615]
• daniel make
g++ -std=c++0x -I. -c main.cpp -o main.o
g++ -std=c++0x main.o -o main.exe

```