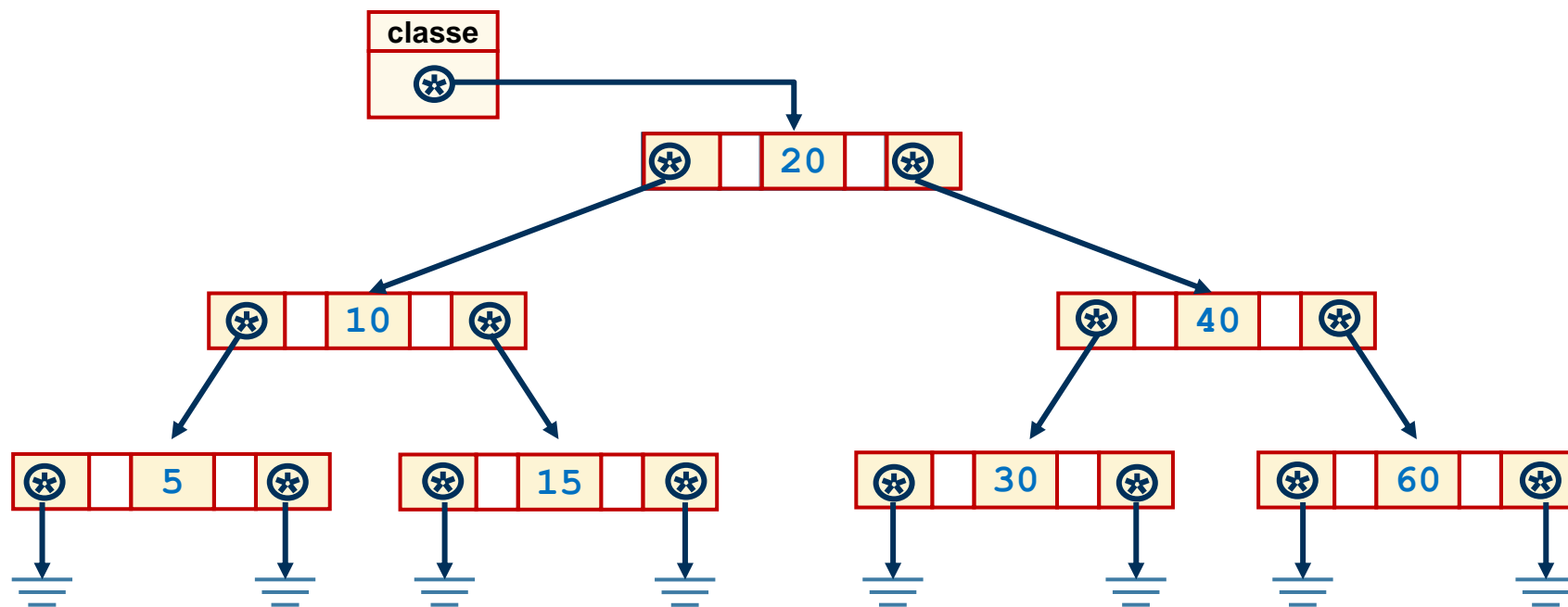




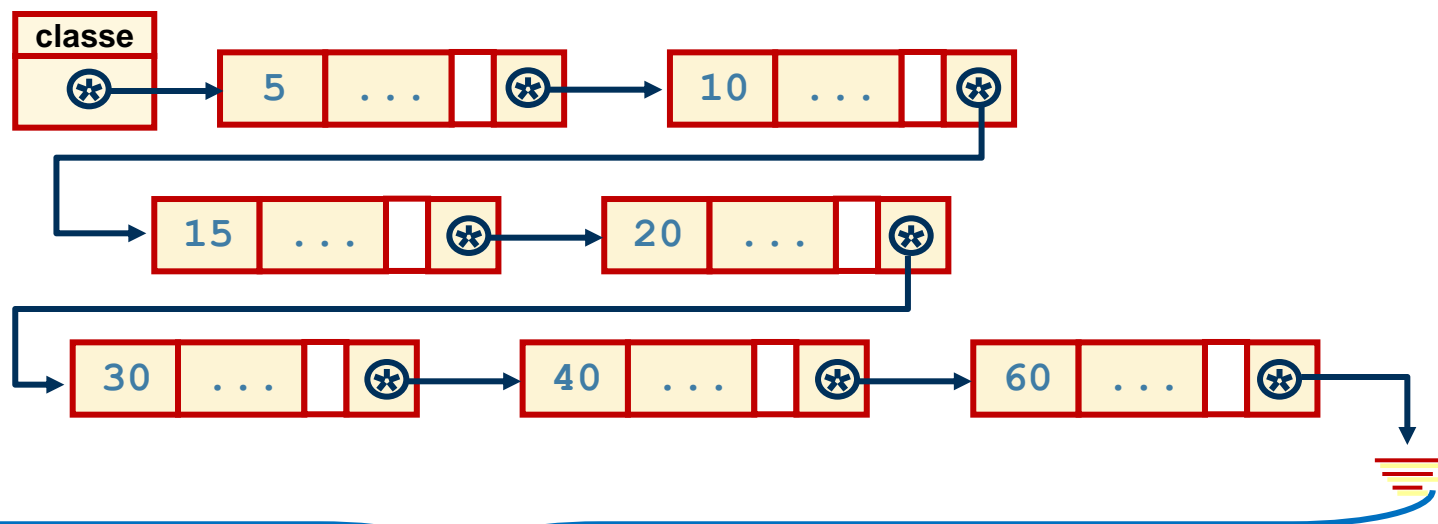
POLITECNICO  
DI MILANO

# INFORMATICA

Gestione di tabelle ad  
albero binario, con uso  
di classi.



classe		
num		elenco
7	0	5 ...
	1	10 ...
	2	15 ...
	3	20 ...
	4	30 ...
	5	40 ...
	6	60 ...
	7	

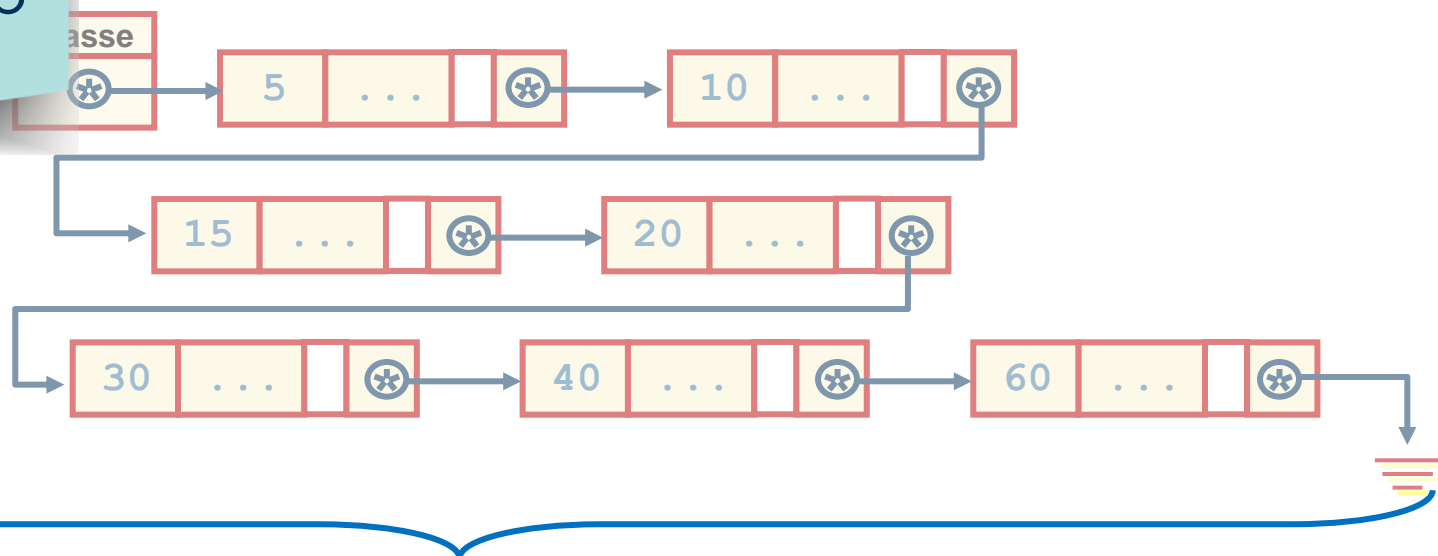


*funzioni*

*main()*

Tabella in stato  
consistente

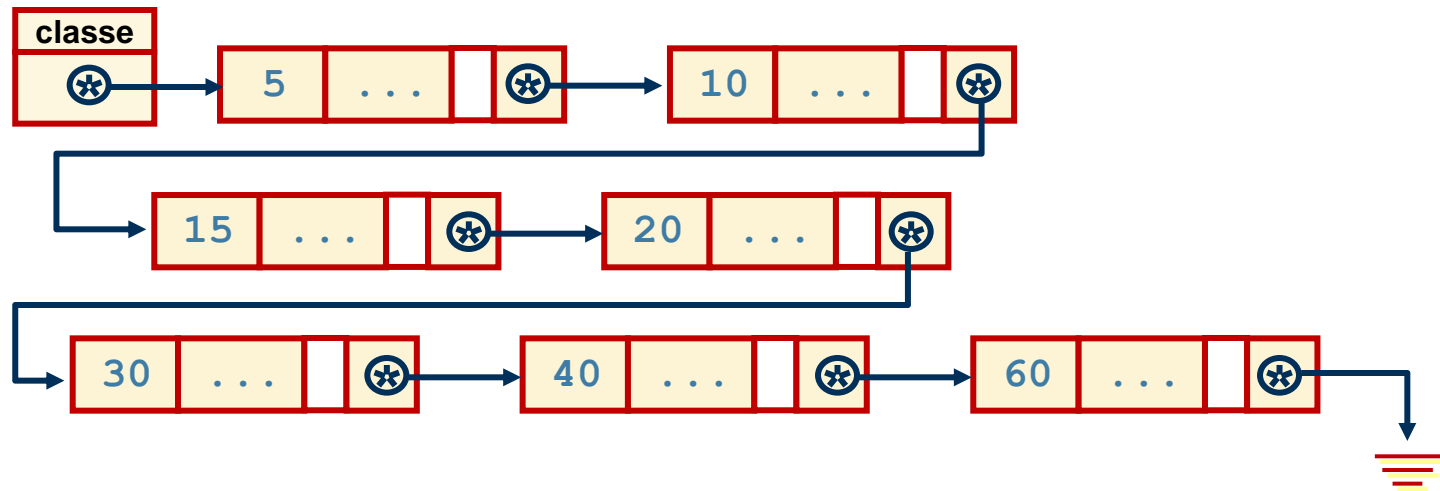
nu		
0		
1	10	...
2	15	...
3	20	...
4	30	...
5	40	...
6	60	...
7		



*inizializzazione, inserimento,  
eliminazione, stampa,*

*main()*

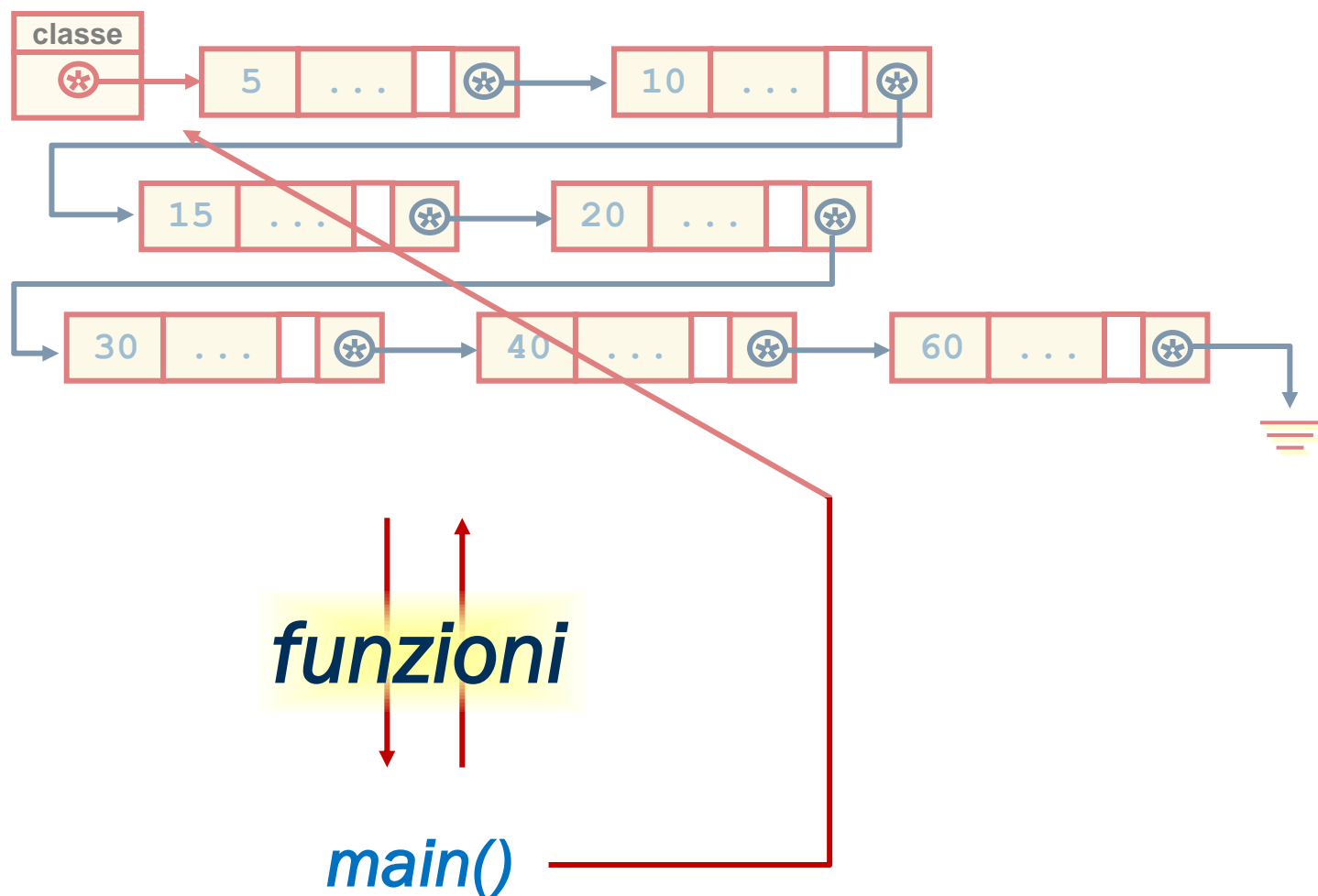
classe		
num		elenco
7	0	5 ...
	1	10 ...
	2	15 ...
	3	20 ...
	4	30 ...
	5	40 ...
	6	60 ...
	7	



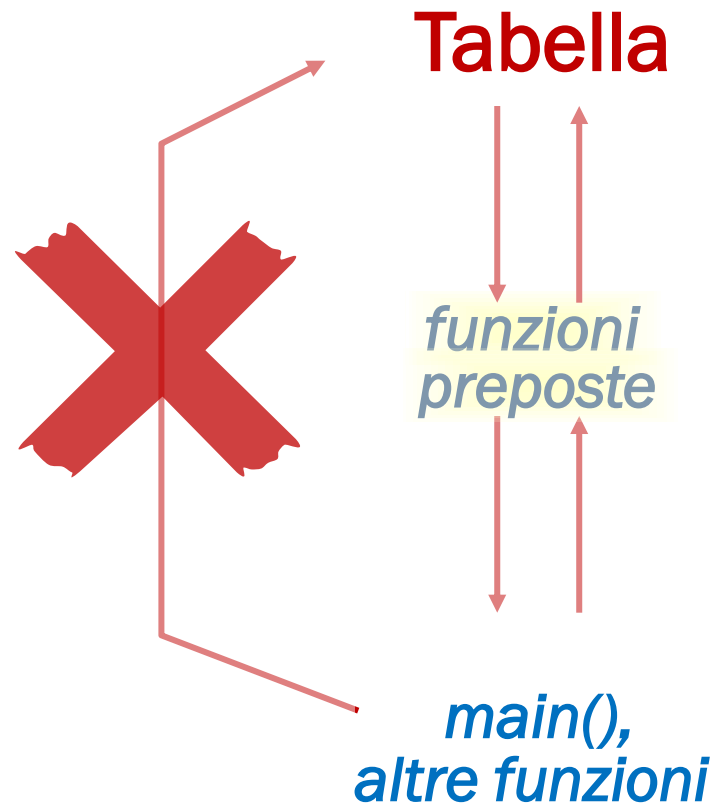
*funzioni*

*main()*

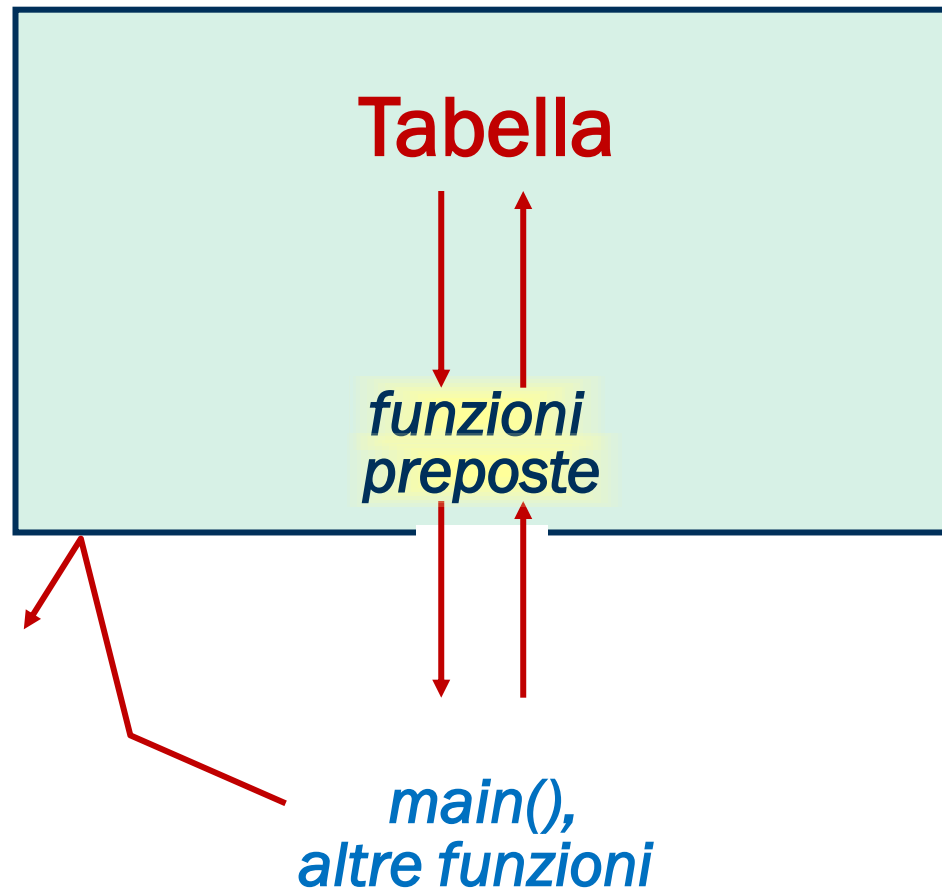
classe		
num		elenco
7	0	5 ...
	1	10 ...
	2	15 ...
	3	20 ...
	4	30 ...
	5	40 ...
	6	60 ...
	7	



## Autodisciplina

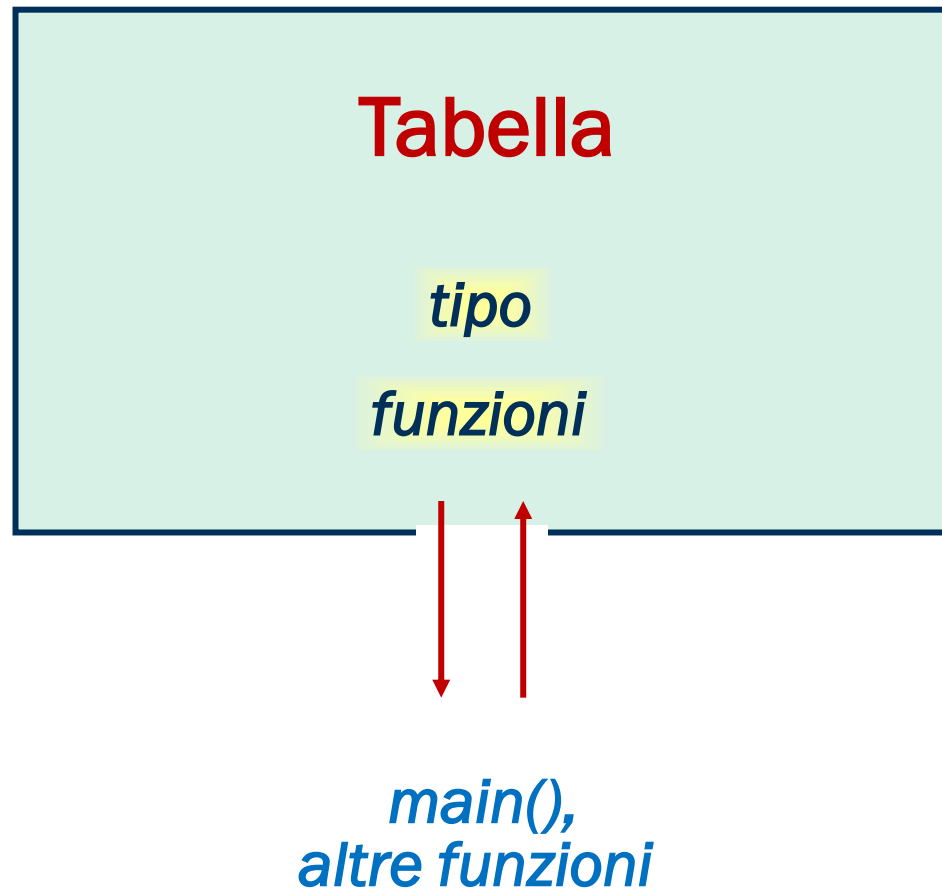


Classe





Gestione "privata"



`classe`

```
{ elementi che  
  definiscono il tipo  
  funzioni che agiscono  
  sugli oggetti del tipo  
};
```

```
class Tree
{
    public:
        Tree ();
        void inserisciSeNonEsiste (Studente);

    private:
        TreeNode *rootPtr;
};
```

```
class Tree
```

```
{
```

```
public:
```

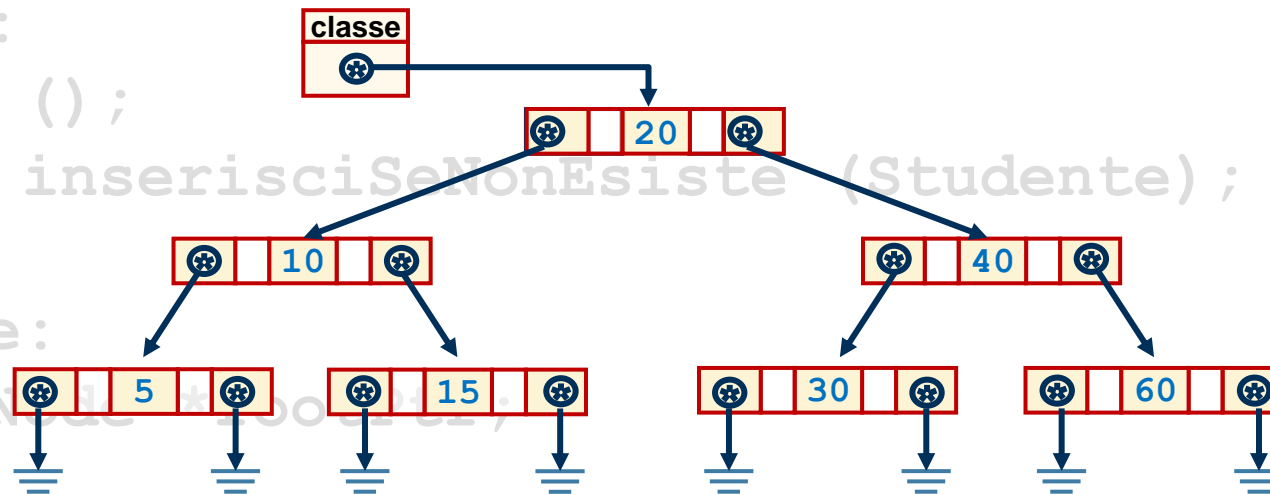
```
    Tree ();
```

```
    void inserisciSeNonEsiste (Studente);
```

```
private:
```

```
    TreeNode *root;
```

```
};
```



```
class Tree
{
    public:
        Tree ();
        void inserisciSeNonEsiste (Studente);

    private:
        TreeNode *rootPtr;
};
```

```
class Tree
{
public:
    Tree ();
    void inserisciSeNonEsiste (Studente);

private:
    insertSeNonEsiste
    TreeNode *rootPtr;
};
```


The diagram illustrates the encapsulation of the `insertSeNonEsiste` method within the `Tree` class. A blue box labeled **Tree** represents the public interface, containing the `private` access specifier and the `inserisciSeNonEsiste (Studente);` method signature. A green box below it, with a dashed border, represents the private implementation, containing the `insertSeNonEsiste` method name and the `TreeNode *rootPtr;` member variable. Red arrows indicate the flow of information: one arrow points from the `private` access specifier down to the implementation box, and another points from the `insertSeNonEsiste` method name in the implementation box up to the `private` access specifier. A third arrow points from the `insertSeNonEsiste` method name in the implementation box down to the `TreeNode *rootPtr;` member variable, and a fourth arrow points from the `TreeNode *rootPtr;` member variable up to the `insertSeNonEsiste` method name in the implementation box.

```
class Tree
{
    public:
        Tree ();
        void inserisciSeNonEsiste (Studente);

    private:
        TreeNode *rootPtr;
};
```

```
class Tree
{
    public:
        Tree ();
        void inserisciSeNonEsiste (Studente);

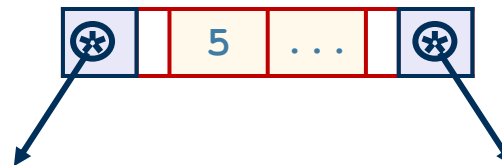
    private:
        TreeNode *rootPtr;
}; allocazione dinamica
```



*Tree classe;*



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



```
class TreeNode
{
    friend class Tree;
    public:
        TreeNode (Studiante) ;
        Studiante getData() ;
    private:
        Studiante datiStud;
        TreeNode *leftPtr;
        TreeNode *rightPtr;
};
```

## Inizializzazione

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode(Studente);
    Studente getData();
private:
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

**TreeNode**

*inizializzazione*

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData();
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



The diagram shows a class named **TreeNode** in a blue box. Below it is a dashed box representing the private section, containing the constructor **TreeNode ()**.

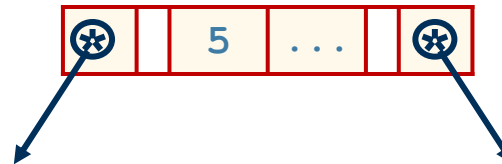
**TreeNode****TreeNode ()**

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class TreeNode
{
    friend class Tree;
public:
    void TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

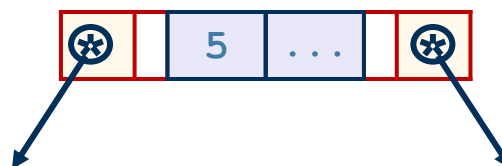
Costruttore





## Parametri

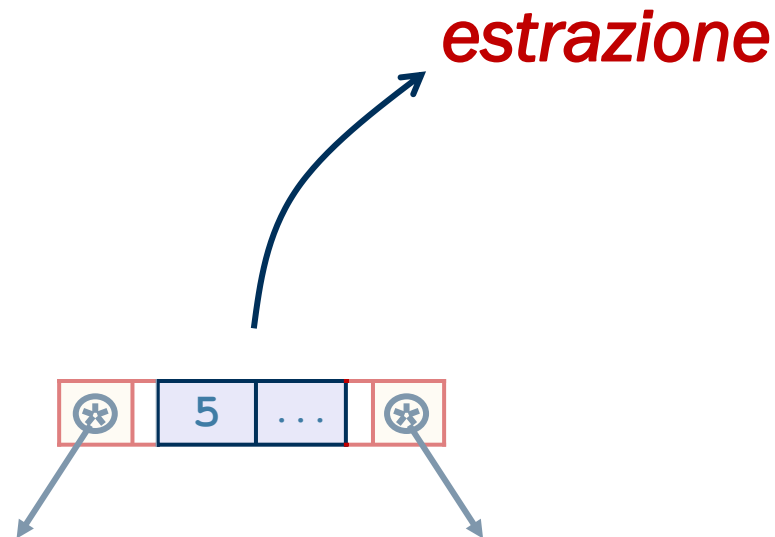
```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studente);
    Studente getData();
private:
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



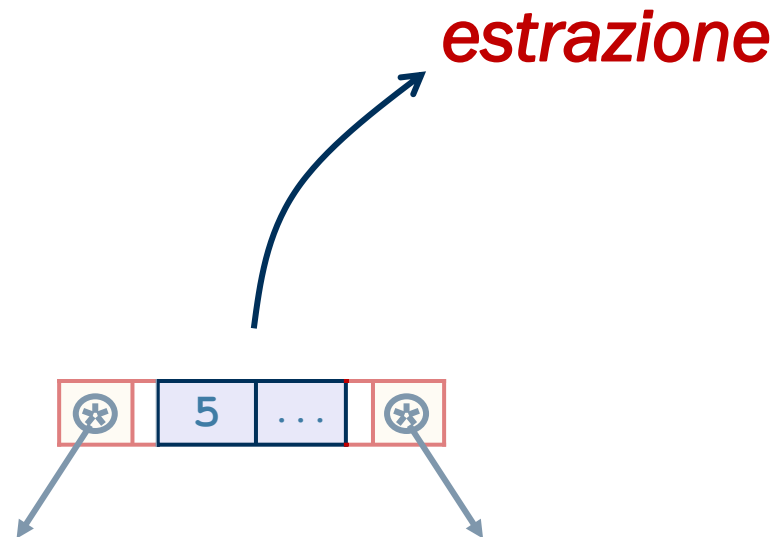
## Parametri

```
void Tree :: inserisciConRicorsione(TreeNode *&ptr;  
                                     Studente nuovoStudiante)  
{  
    if (ptr == 0)  
    { ptr = new TreeNode(nuovoStudiante) ;  
    }  
    else if (nuovoStudiante.matricola < ptr->datiStud.matricola)  
        inserisciConRicorsione (ptr->leftPtr, nuovoStudiante)
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData () ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData();
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```



## Membro di una classe

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
alfa = new TreeNode (nuovoStudiante)
```

**alfa**

## Dot notation

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData () ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

alfa .getData ()



## Prototipi

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class MiaClasse
{
    ...
};
```

```
TreeNode (Studiante nuovoStudiante)
{
    ...
};
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
class Tree
{
    ...
};
```

```
TreeNode (Studiante nuovoStudiante)
{
    ...
};
```

```
class TreeNode
```

```
{  
    friend class Tree;  
public:  
    TreeNode (Studiante) ;  
    Studiante getData() ;  
private:  
    Studiante datiStud;  
    TreeNode *leftPtr;  
    TreeNode *rightPtr;  
};
```

```
class Tree  
{  
    ...  
};
```

```
TreeNode (Studiante nuovoStudiante)
```

```
{  
    ...  
};
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData () ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

Doppi due punti ::

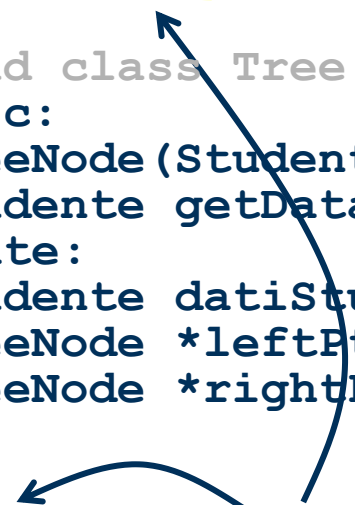
```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
class TreeNode
```

```
{  
    friend class Tree;  
public:  
    TreeNode(Studente);  
    Studente getData();  
private:  
    Studente datiStud;  
    TreeNode *leftPtr;  
    TreeNode *rightPtr;  
};
```

```
TreeNode :: TreeNode(Studente nuovoStudente)  
{  
    datiStud = nuovoStudente;  
    leftPtr = rightPtr = 0;  
}
```





```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```


```
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

```
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

Studiante TreeNode :: getData ()
{
}
```

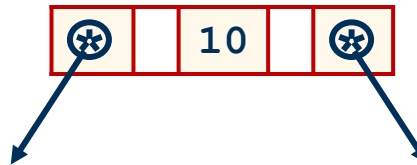
```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

Studiante TreeNode :: getData()
{
    return datiStud;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode(Studiante);
    Studiante getData();
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};
```

***TreeNode***



```
TreeNode :: TreeNode(Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}
```

```
Studiante TreeNode :: getData()
{
    return datiStud;
}
```

**Tree**

```

class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

```

```

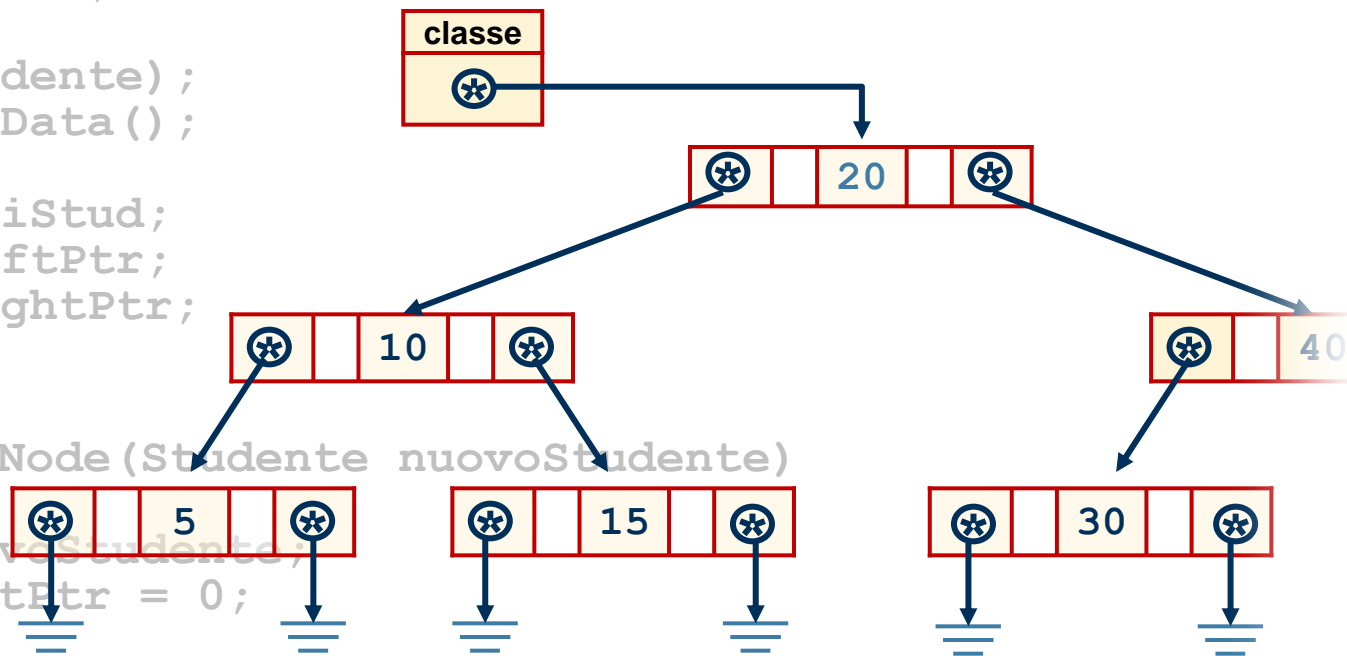
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

```

```

Studiante TreeNode :: getData()
{
    return datiStud;
}

```





```

class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante);
    Studiante getData();
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

```

```

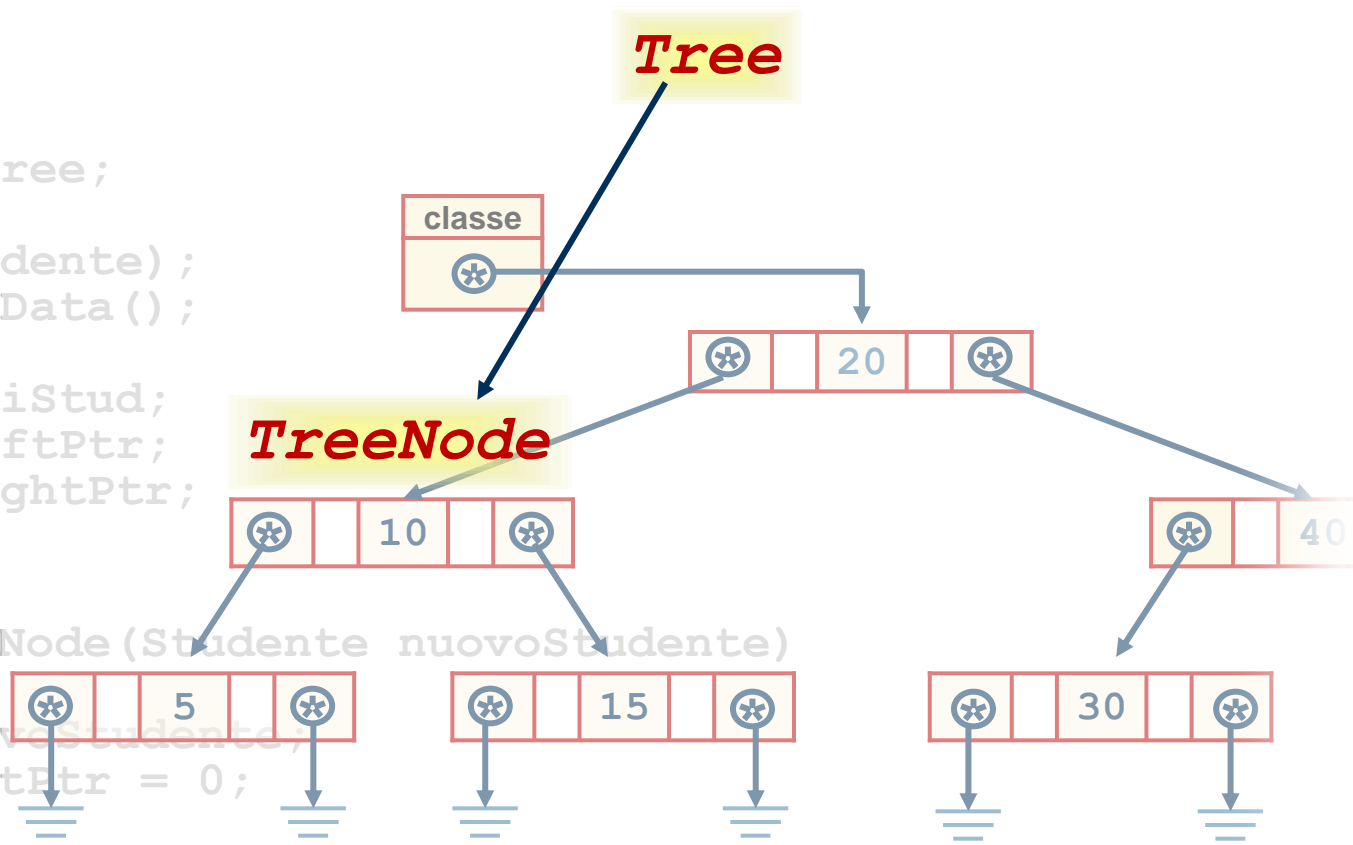
TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

```

```

Studiante TreeNode :: getData()
{
    return datiStud;
}

```



```
class TreeNode
```

```
{
```

```
    friend class Tree;
```

```
public:
```

```
    TreeNode (Studiante) ;
```

```
    Studiante getData() ;
```

```
private:
```

```
    Studiante datiStud;
```

```
    TreeNode *leftPtr;
```

```
    TreeNode *rightPtr;
```

```
};
```

```
public:
```

```
TreeNode :: TreeNode (Studiante nuovoStudiante)
```

```
{
```

```
    datiStud = nuovoStudiante;
```

```
    leftPtr = rightPtr = 0;
```

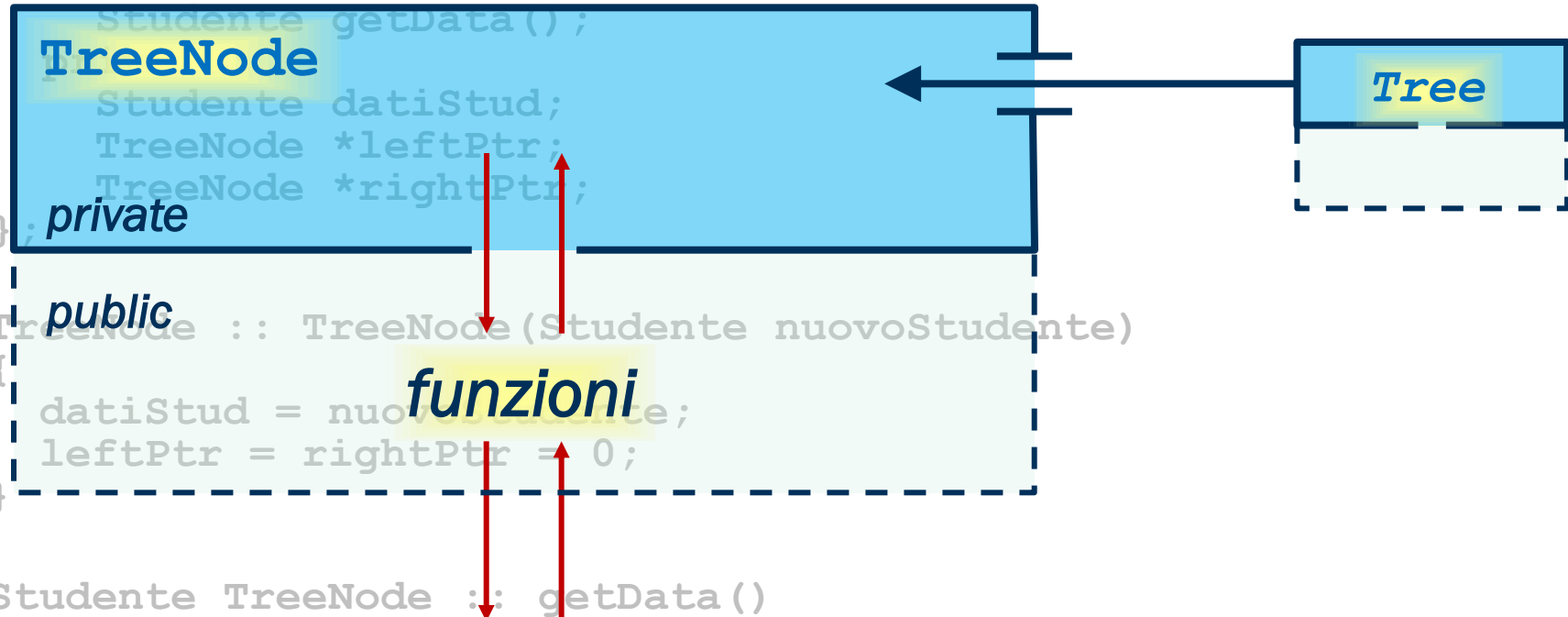
```
}
```

```
Studiante TreeNode :: getData()
```

```
{
```

```
    return datiStud;
```

```
}
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

Studiante TreeNode :: getData()
{
    return datiStud;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

Studiante TreeNode :: getData()
{
    return datiStud;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode(Studente);
    Studente getData();
private:
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode(Studente nuovoStudente)
{
    datiStud = nuovoStudente;
    leftPtr = rightPtr = 0;
}

Studente TreeNode :: getData()
{
    return datiStud;
}
```

```
class TreeNode
```

```
{  
    friend class Tree;  
public:  
    TreeNode (Studiante) ;  
    Studiante getData() ;  
    Studiante datiStud;  
    TreeNode *leftPtr;  
    TreeNode *rightPtr;  
};  
  
private  
public  
    TreeNode :: TreeNode (Studiante nuovoStudiante)  
    {  
        datiStud = nuovoStudiante;  
        leftPtr = rightPtr = 0;  
    }  
  
    Studiante TreeNode :: getData ()  
    {  
        return datiStud;  
    }  
}
```

**funzioni**



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studiante) ;
    Studiante getData() ;
private:
    Studiante datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studiante nuovoStudiante)
{
    datiStud = nuovoStudiante;
    leftPtr = rightPtr = 0;
}

Studiante TreeNode :: getData()
{
    return datiStud;
}
```

```
class TreeNode
{
    friend class Tree;
public:
    TreeNode (Studente) ;
    Studente getData()
private:
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode (Studente nuovoStudente)
{
    datiStud = nuovoStudente;
    leftPtr = rightPtr = 0;
}

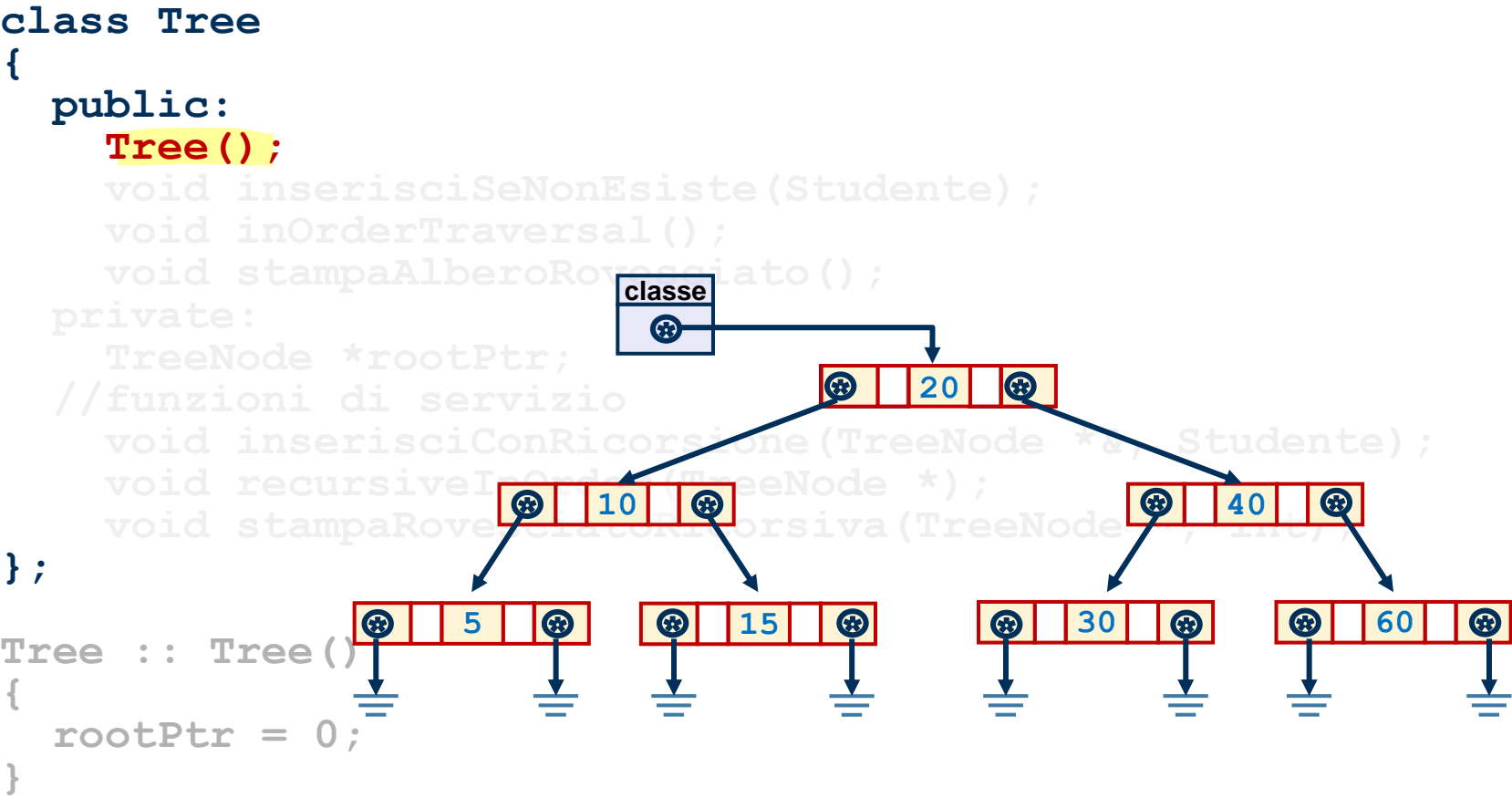
Studente TreeNode :: getData ()
{
    return datiStud;
}
```



```
class TreeNode
{
    friend class Tree;
public:
    TreeNode(Studente);
    Studente getData();
private:
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
};

TreeNode :: TreeNode(Studente nuovoStudente)
{
    datiStud = nuovoStudente;
    leftPtr = rightPtr = 0;
}

Studente TreeNode :: getData()
{
    return datiStud;
}
```



```
class Tree
{
```

```
    public:
```

```
        Tree();
```

```
        void inserisciSeNonEsiste(Studente);
```

```
        void inOrderTraversa();
```

```
        void stampaAlberoRovesciato();
```

```
    private:
```

```
        TreeNode *rootPtr;
```

```
        //funzioni di servizio
```

```
        void inserisciCognomeOne(TreeNode *nodo, Studente studente);
```

```
        void recursiveInOrder(TreeNode *);
```

```
        void stampaRovesciatoRicerca(TreeNode *);
```

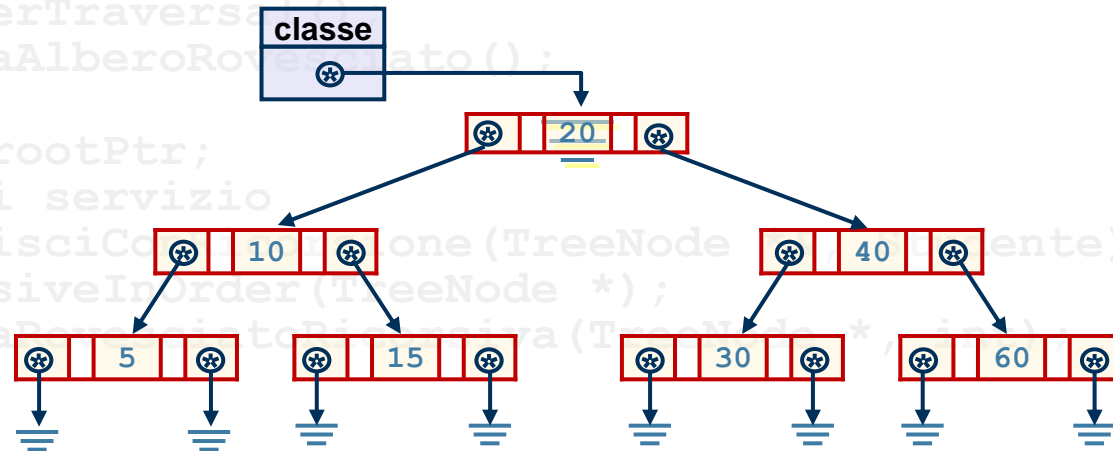
```
};
```

```
Tree :: Tree()
```

```
{
```

```
    rootPtr = 0;
```

```
}
```



```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

```
class Tree
{
```

```
    public:
```

```
        Tree();
```

```
        void inserisciSeNonEsiste(Studente);
```

```
        void inOrderTraversal();
```

```
        void stampaAlberoRovesciato();
```

```
    private:
```

```
        TreeNode *rootPtr;
```

```
        //funzioni di servizio
```

```
        void inserisciConRicorsione(TreeNode *&, Studente);
```

```
        void recursiveInOrder(TreeNode *);
```

```
        void stampaRovesciatoRicorsiva(TreeNode *, int);
```

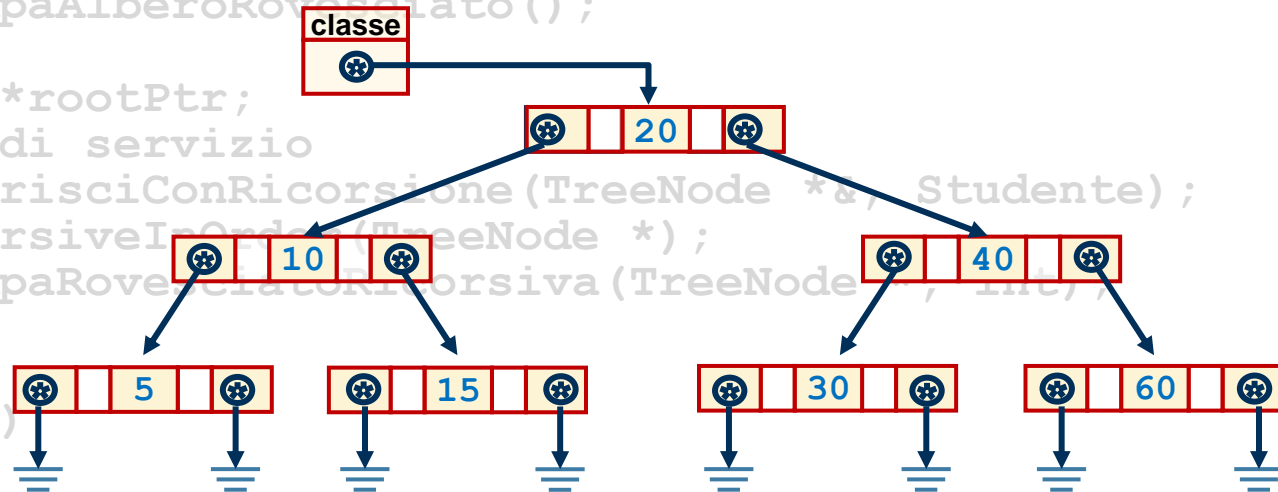
```
};
```

```
Tree :: Tree()
```

```
{
```

```
    rootPtr = 0;
```

```
}
```



```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studiante);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        classe.inserisciSeNonEsiste(nuovoStudiante);
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studiante);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studiante);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studiante);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```



```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicerca(TreeNode *, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

**~~eliminaSeEsiste~~**

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
}
```

5 10 15 20 30 40 60

```
class Tree  
{
```

```
public:
```

```
    Tree();
```

```
    void inserisciSeNonEsiste(Studente);
```

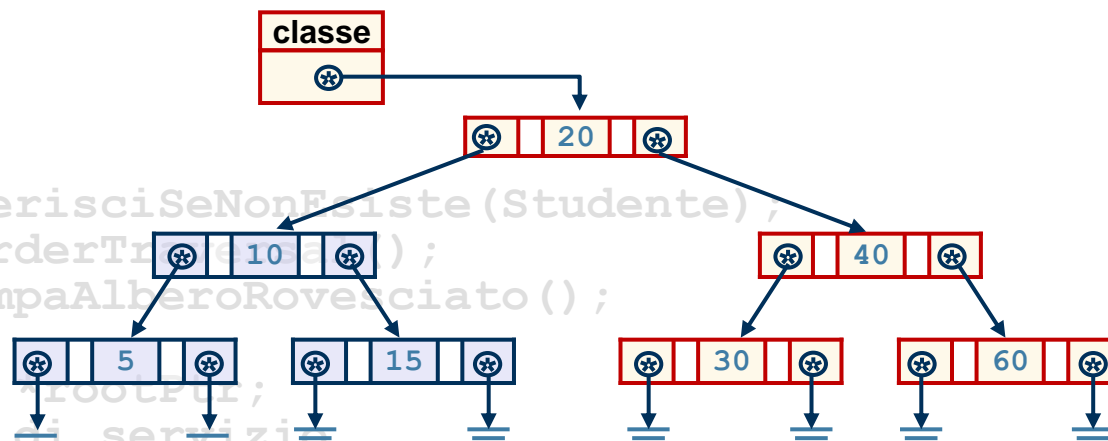
```
    void inOrderTraverse();
```

```
    void stampaAlberoRovesciato();
```

```
private:
```

```
    TreeNode *rootPtr;
```

```
    //funzioni di servizio
```



5 10 15 20 30 40 60

```
class Tree  
{
```

```
public:
```

```
    Tree();
```

```
    void inserisciSeNonEsiste(Studente);
```

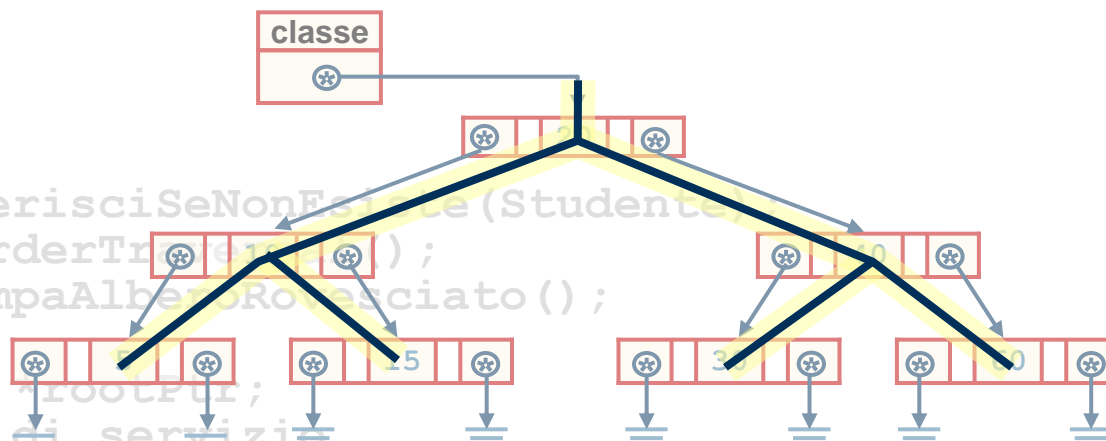
```
    void inOrderTraverse();
```

```
    void stampaAlberoRovesciato();
```

```
private:
```

```
    TreeNode *rootPtr;
```

```
    //funzioni di servizio
```



```
class Tree  
{
```

```
public:
```

```
    Tree();
```

```
    void inserisciSeNonEsiste(Studente);
```

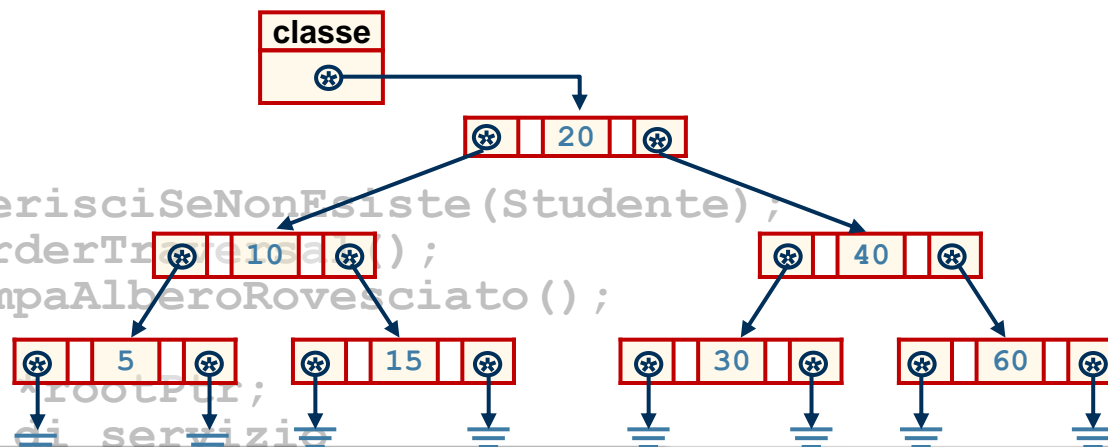
```
    void inOrderTraverse();
```

```
    void stampaAlberoRovesciato();
```

```
private:
```

```
    TreeNode *rootPtr;
```

```
    //funzioni di servizio
```



60  
40  
20  
30  
15  
10  
5

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
}
```

```
graph TD
    60 --> 40
    60 --> 30
    40 --> 20
    40 --> 10
    20 --> 10
    10 --> 5
```



```
class Tree
{
```

```
public:
```

```
    Tree();
```

```
    void inserisciSeNonEsiste(Studente);
```

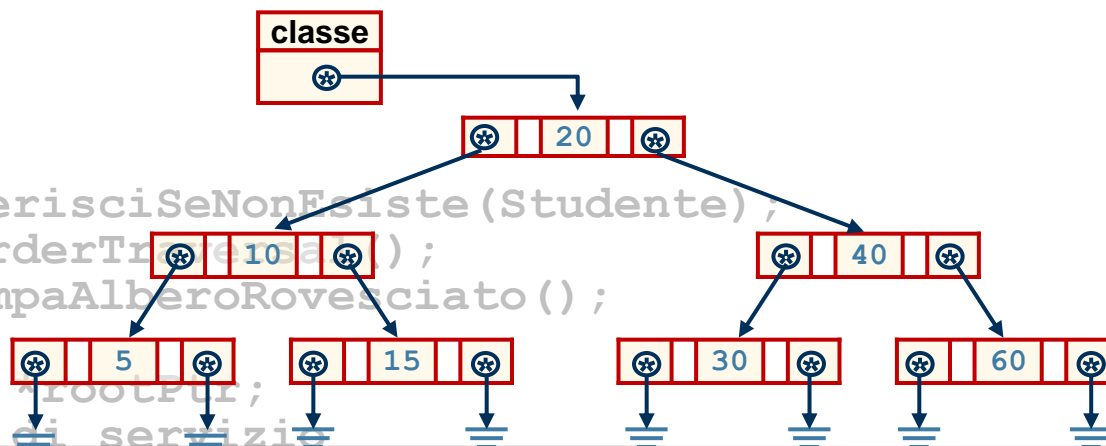
```
    void inOrderTraverse();
```

```
    void stampaAlberoRovesciato();
```

```
private:
```

```
    TreeNode *rootPtr;
```

```
    //funzioni di servizio
```

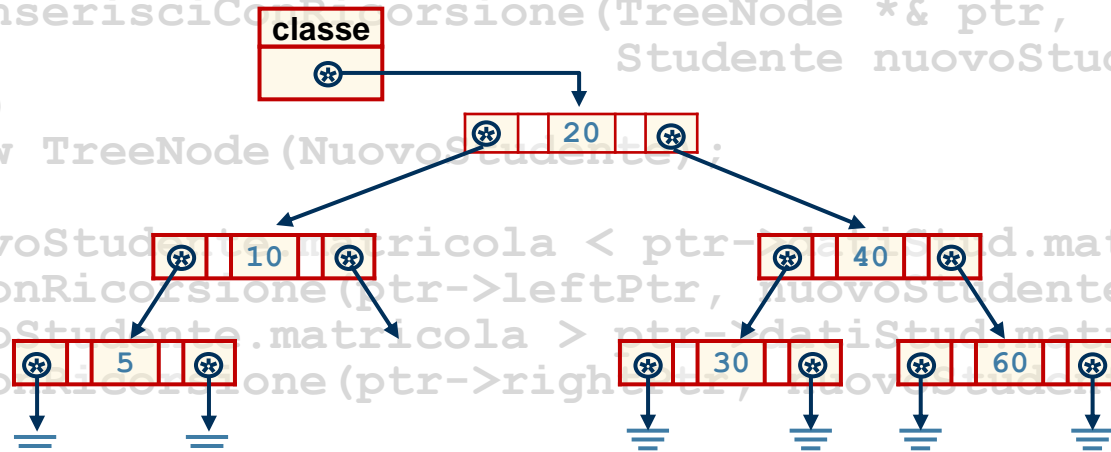


20  
10 40  
5 15 30 60

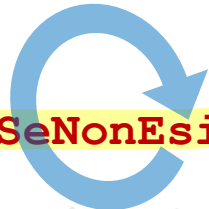
## Funzione ricorsiva

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```

```
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{
    if (ptr == 0)
    {
        ptr = new TreeNode(NuovoStudente);
    }
    else if (nuovoStudente.matricola < ptr->datiStudente.matricola)
        inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > ptr->datiStudente.matricola)
        inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
    else
    {
        cout << "studente di matricola "
              << nuovoStudente.matricola
              << " già presente" << endl;
    }
}
```



## Funzione ricorsiva

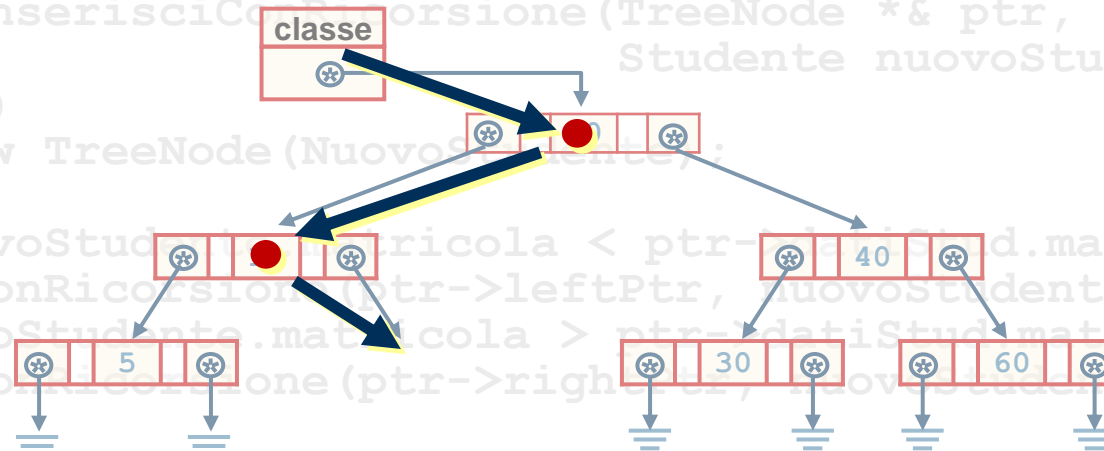


```
void Tree :: inserisciSeNonEsiste (Studente nuovoStudente)
```

```
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```

```
void Tree :: inserisciConRicorsione (TreeNode *& ptr,
                                     Studente nuovoStudente)
```

```
{ if (ptr == 0)
  { ptr = new TreeNode(NuovoStudente);
  }
  else if (nuovoStudente.matricola < ptr->datiStudente.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStudente.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
  }
```



```
inserisciSeNonEsiste(TreeNode *&tab, Studente nuovoStudente)
{
    if (tab == 0)
    {
        tab = new TreeNode;
        tab->datiStud = nuovoStudente;
        tab->leftPtr = 0;
        tab->rightPtr = 0;
    }
    else if (nuovoStudente.matricola < tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->rightPtr, nuovoStudente);
    else
        cout << nuovoStudente.matricola << " duplicato" << endl;
}
```



## Funzione ricorsiva

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

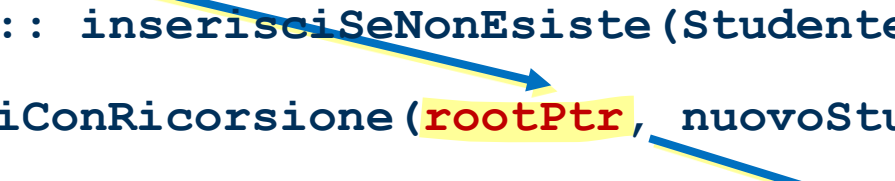
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{
    if (ptr == 0)
    {
        ptr = new TreeNode(NuovoStudente);
    }
    else if (nuovoStudente.matricola < ptr->datiStud.matricola)
        inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > ptr->datiStud.matricola)
        inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
    else
        cout << "studente di matricola "
              << nuovoStudente.matricola
              << " già presente" << endl;
}
```

```
class Tree
{
    public:
        Tree();
        void inserisciSeNonEsiste(Studente);
        void inOrderTraversal();
        void stampaAlberoRovesciato();
    private:
        TreeNode *rootPtr;
        //funzioni di servizio
        void inserisciConRicorsione(TreeNode *&, Studente);
        void recursiveInOrder(TreeNode *);
        void stampaRovesciatoRicorsiva(TreeNode *, int);
};

Tree :: Tree()
{
    rootPtr = 0;
}
```

*classe.inserisciSeNonEsiste(nuovoStudente);*

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```



```
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                   Studente nuovoStudente)
{ if (ptr == 0)
  { ptr = new TreeNode(NuovoStudente);
  }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
}
```



```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{ if (ptr == 0)
    { ptr = new TreeNode(NuovoStudente);
    }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
}
```

```
inserisciSeNonEsiste(TreeNode *&tab, Studente nuovoStudente)
{
    if (tab == 0)
    {
        tab = new TreeNode;
        tab->datiStud = nuovoStudente;
        tab->leftPtr = 0;
        tab->rightPtr = 0;
    }
    else if (nuovoStudente.matricola < tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->rightPtr, nuovoStudente);
    else
        cout << nuovoStudente.matricola << " duplicato" << endl;
}
```

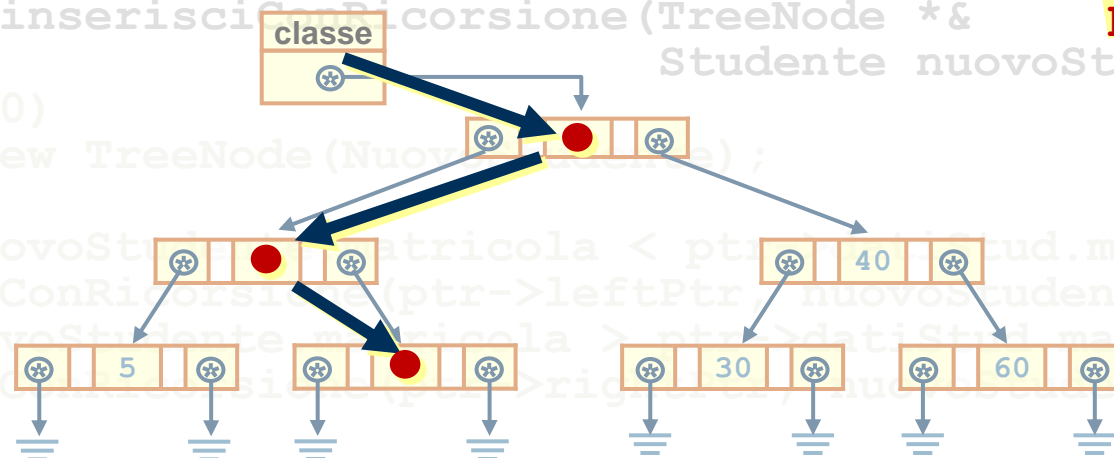
```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{
    if (ptr == 0)
    {
        ptr = new TreeNode(NuovoStudente);
    }
    else if (nuovoStudente.matricola < ptr->datiStud.matricola)
        inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > ptr->datiStud.matricola)
        inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
    else
        cout << "studente di matricola "
              << nuovoStudente.matricola
              << " già presente" << endl;
}
```

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```

```
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
```

```
{ if (ptr == 0)
  { ptr = new TreeNode(NuovoStudente);
  }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
  }
```



```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{ if (ptr == 0)
    { ptr = new TreeNode(NuovoStudente);
    }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
}
```

```
inserisciSeNonEsiste(TreeNode *&tab, Studente nuovoStudente)
{
    if (tab == 0)
    {
        tab = new TreeNode;
        tab->datiStud = nuovoStudente;
        tab->leftPtr = 0;
        tab->rightPtr = 0;
    }
    else if (nuovoStudente.matricola < tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->rightPtr, nuovoStudente);
    else
        cout << nuovoStudente.matricola << " duplicato" << endl;
}
```

```
inserisciSeNonEsiste(TreeNode *&tab, Studente nuovoStudente)
{
    if (tab == 0)
    {
        tab = new TreeNode;
        tab->datiStud = nuovoStudente;
        tab->leftPtr = 0;
        tab->rightPtr = 0;
    }
    else if (nuovoStudente.matricola < tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->leftPtr, nuovoStudente);
    else if (nuovoStudente.matricola > tab->datiStud.matricola)
        inserisciSeNonEsiste(tab->rightPtr, nuovoStudente);
    else
        cout << nuovoStudente.matricola << " duplicato" << endl;
}
```

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{ if (ptr == 0)
    { ptr = new TreeNode(nuovoStudente);
    }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola "
          << nuovoStudente.matricola
          << " già presente" << endl;
}
```



```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```

```
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
```

```
{ if (ptr == 0)
  { ptr = new TreeNode(nuovoStudente);
  }
```

```
else if (nuovoStudente.datiStud.matricola < ptr->datiStud.matricola)
```

```
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
```

```
else if (nuovoStudente.datiStud.matricola > ptr->datiStud.matricola)
```

```
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
```

```
else
```

```
    cout << "studente di matricola " << nuovoStudente.datiStud.matricola
```

```
    << " già presente nel database.\n";
```

```
    return;
```

```
}
```

```
class TreeNode
```

```
{
```

```
public:
```

```
    TreeNode(Studente);
```

```
private:
```

```
    Studente datiStud;
```

```
    TreeNode *leftPtr;
```

```
    TreeNode *rightPtr;
```

```
}
```

**classe.inserisciSeNonEsiste(nuovoStudente);**

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}
```

```
void Tree :: inserisciConRicorsione(TreeNode *& ptr,
    Studente nuovoStudente)
```

```
{ if (ptr == 0)
  { ptr = new TreeNode(nuovoStudente);
  }
```

```
else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
else
```

```
    cout << "studente di matricola " << nuovoStudente.matricola
    << " già presente" << endl;
```

```
private:
```

```
    Studente datiStud;
    TreeNode *leftPtr;
    TreeNode *rightPtr;
```

```
}
```

```
void Tree :: inserisciSeNonEsiste(Studente nuovoStudente)
{
    inserisciConRicorsione(rootPtr, nuovoStudente);
}

void Tree :: inserisciConRicorsione(TreeNode *& ptr,
                                     Studente nuovoStudente)
{ if (ptr == 0)
  { ptr = new TreeNode(nuovoStudente);
  }
  else if (nuovoStudente.matricola < ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->leftPtr, nuovoStudente);
  else if (nuovoStudente.matricola > ptr->datiStud.matricola)
    inserisciConRicorsione(ptr->rightPtr, nuovoStudente);
  else
    cout << "studente di matricola " << nuovoStudente.matricola
          << " già presente" << endl;
}
```

**class TreeNode**

**public:**

**TreeNode (Studente) ;**

**private:**

**Studente datiStud;**

**TreeNode \*leftPtr;**

**TreeNode \*rightPtr;**

**}**