Data Structures Second Assignment

DICTONARY ,TREE

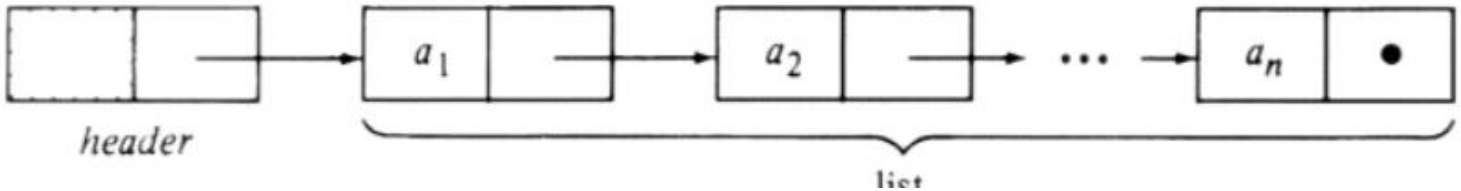
Carlos Garrido Junco 02570033J

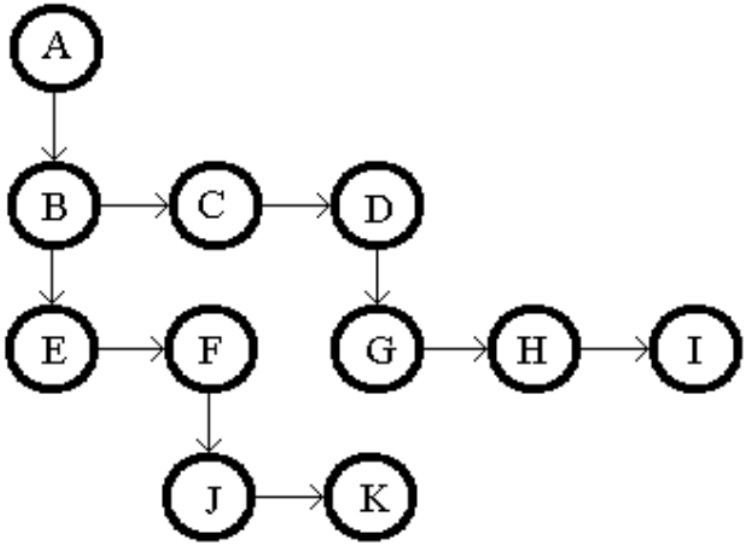
Contents

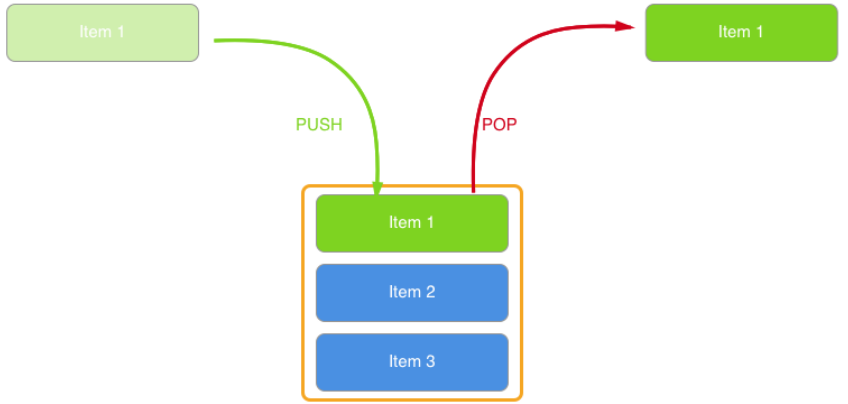
ANALYSIS

ADT specification

- The project will use list, queue, stacks and tree.







ANALYSIS

Definition of the operations of the ADT

```
spec TREE[NODE]
     genres tree, node, label
     operations
          parent: node tree -> node
          leftmost_child: node tree -> node
          right_sibling: node tree -> node
          label: node tree -> label
          create: label tree tree -> tree
          root: tree -> node
          makenull: tree -> tree
endspec
```

```
spec QUEUE[ITEM]
    genres queue, item
    operations
        enqueue: queue item->queue
        dequeue: queue->item
        front: queue->item
        makenull: queue->queue
        empty: queue->boolean
endspec
```



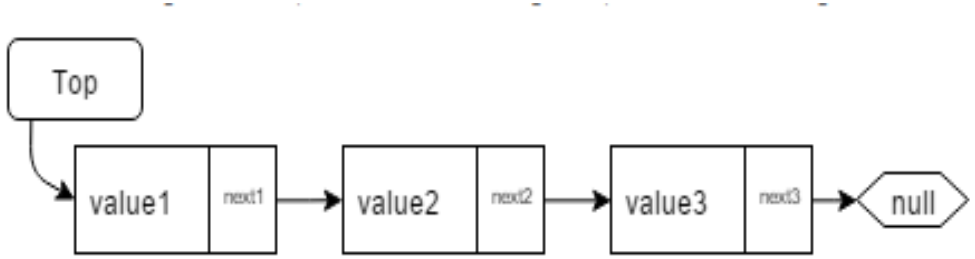
```
spec STACK[ITEM]
    genres stack, item
    operations
        push: stack item->stack
        pop: stack->item
        top: stack->item
        makenull: stack->stack
        empty: stack->boolean
endspec
```

```
spec LIST[ITEM]
     genres list, item, position
     operations
          insert:item position list->list
          delete:position list->list
          locate:item list->position
          retrieve:position list->item
          next:position list->item
          previous:position list->item
          makenull:list->list
          empty:list->bool
endspec
```

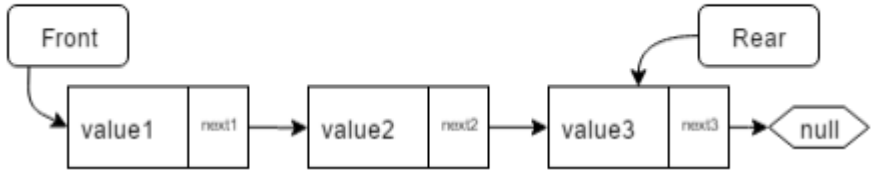
DESIGN

Diagram of the representation of the ADT in the memory of the computer

- Representation of the stack.



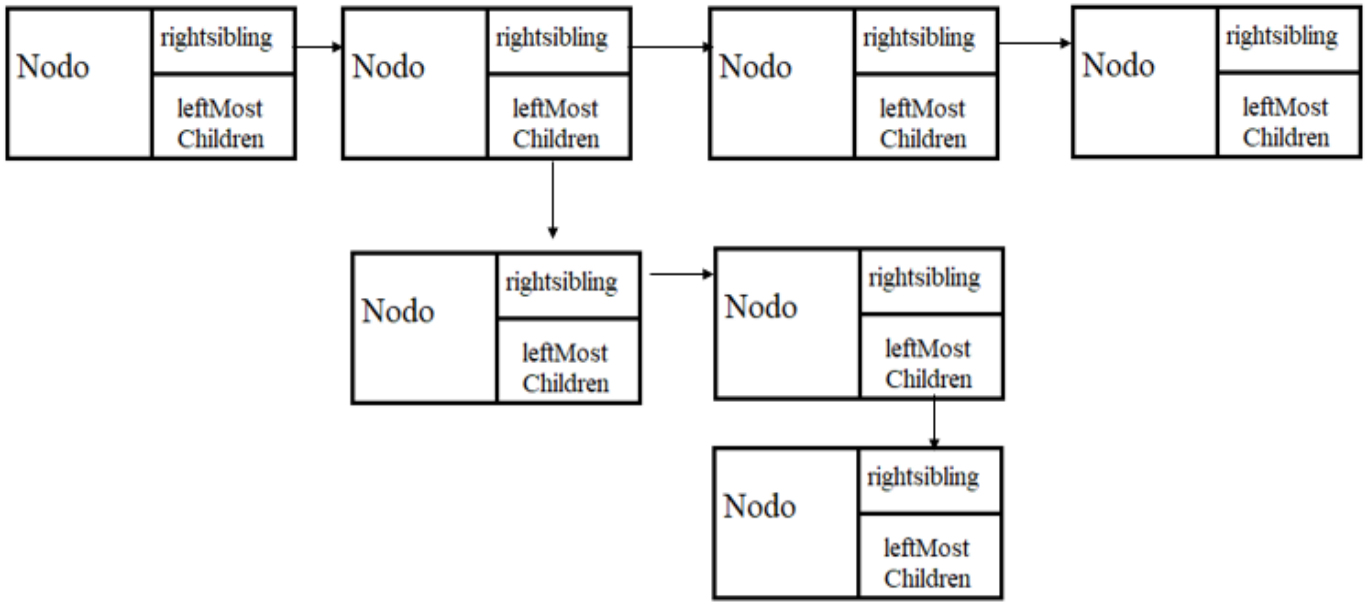
- Representation of the list and the queue.



DESIGN

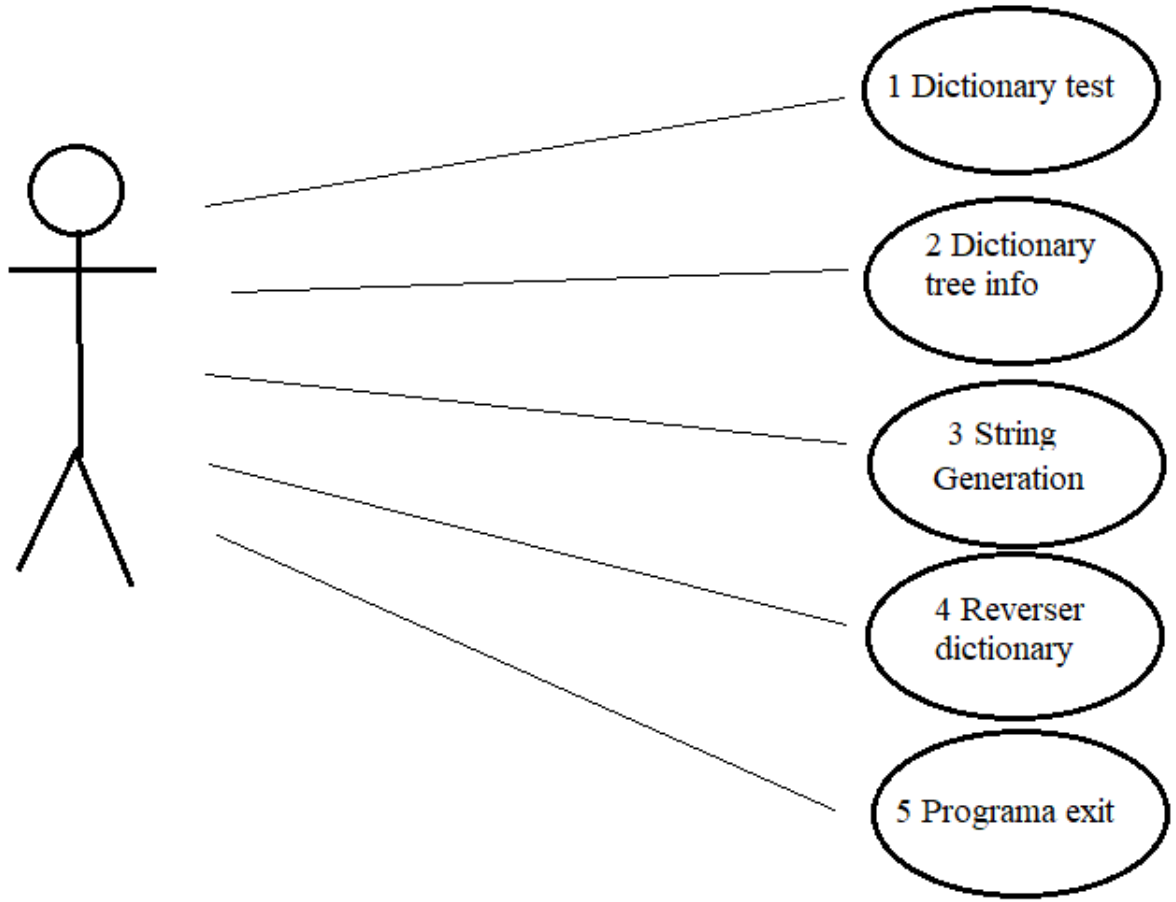
Diagram of the representation of the ADT in the memory of the computer

- Representation of the tree.



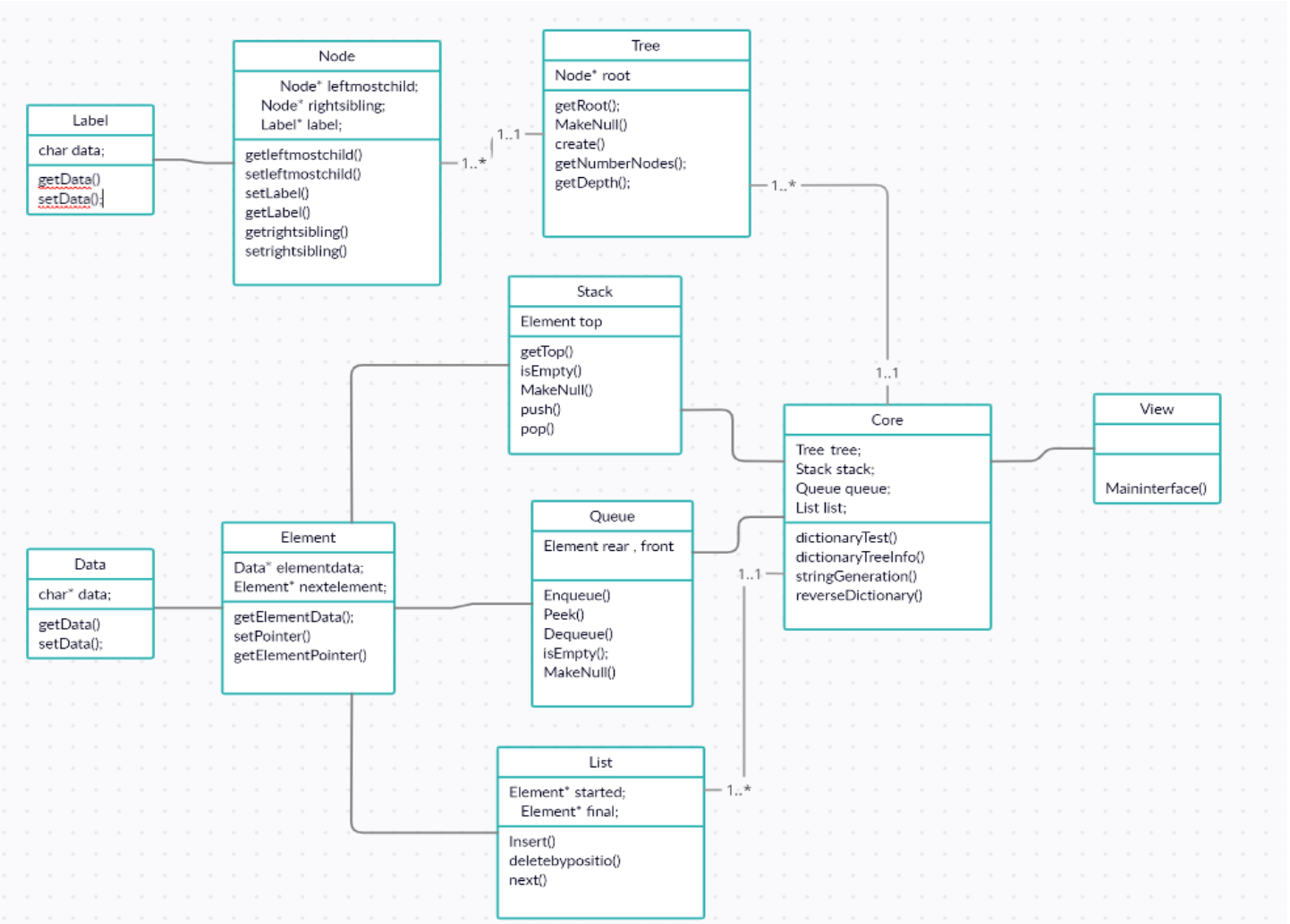
DESIGN

UML Diagram Use case -diagram



DESIGN

UML Diagram Class diagram

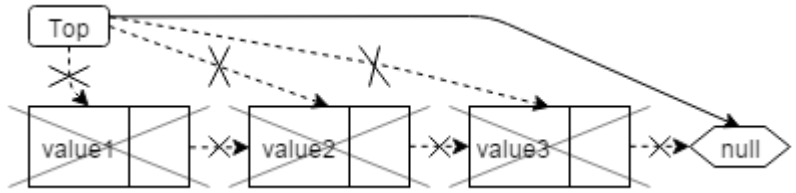


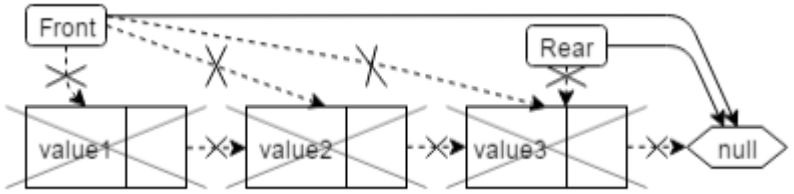
Explanation of the class

Explanation of ADT methods:

Shared methods:

- Makenull(): delete all structure and erase its memory space.
- IsEmpty() check if the structure is empty.



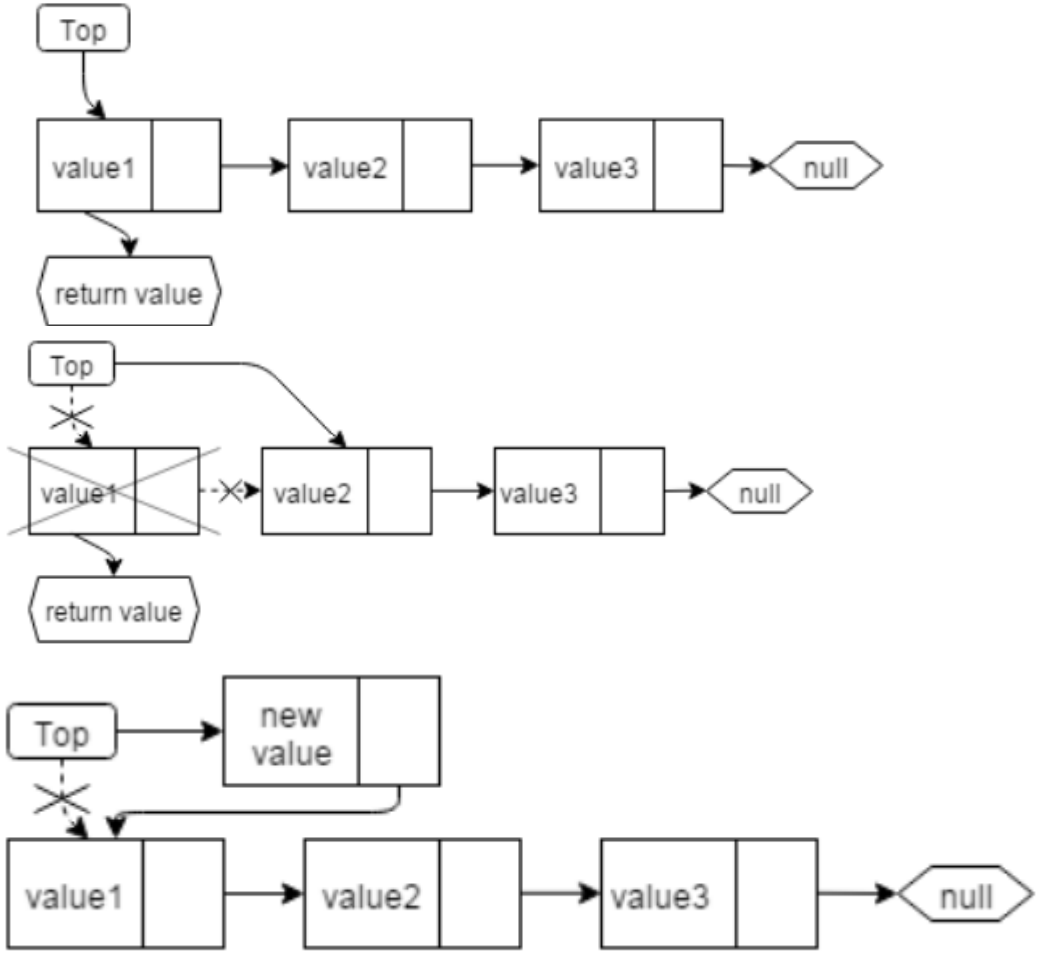


Explanation of the class

Explanation of ADT methods:

Stack methods:

- Top:return the top value
- Pop:return one element and delete it from the stack.
- Push insert a new element in the top

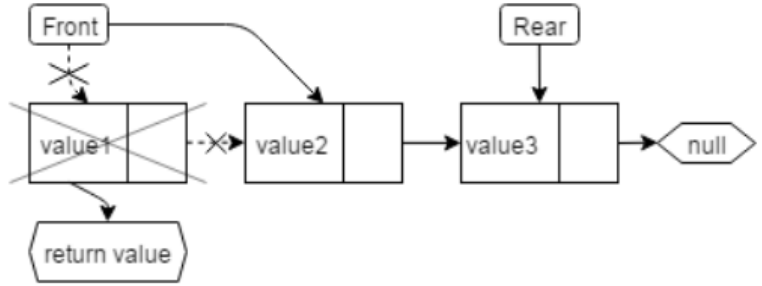


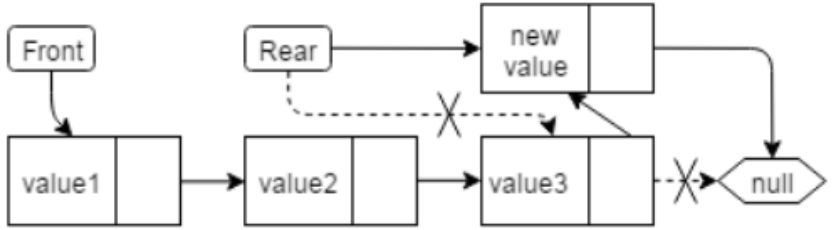
Explanation of the class

Explanation of ADT methods:

Queue methods:

- Dequeue: Extract the element pointed by front
- Enqueue:insert new element in the queue



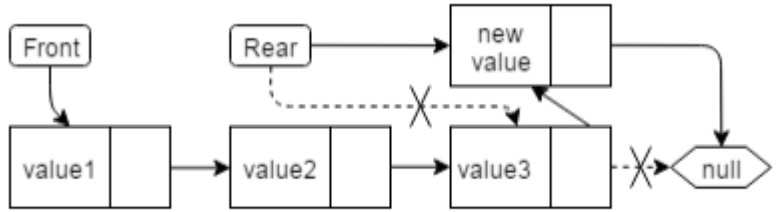


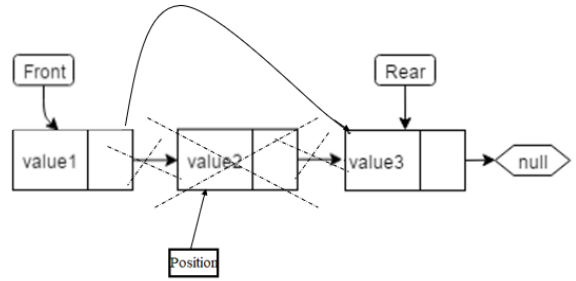
Explanation of the class

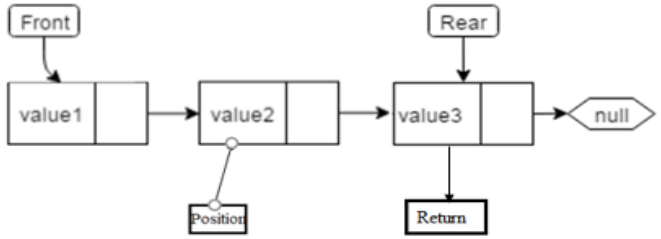
Explanation of ADT methods:

List methods:

- Insert:insert one element.
- Deletebyposition,delete the element idicated by position.
- Next return the next element indicate by position.





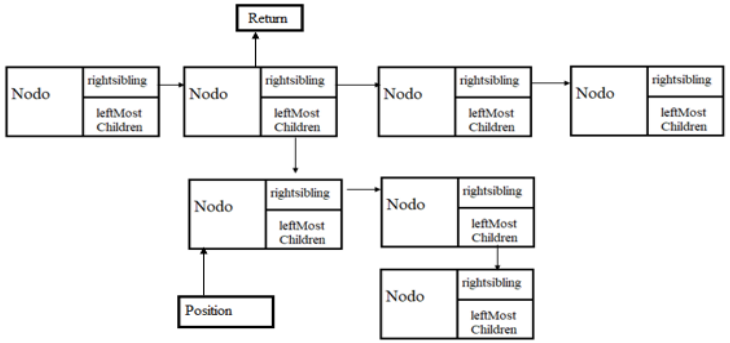


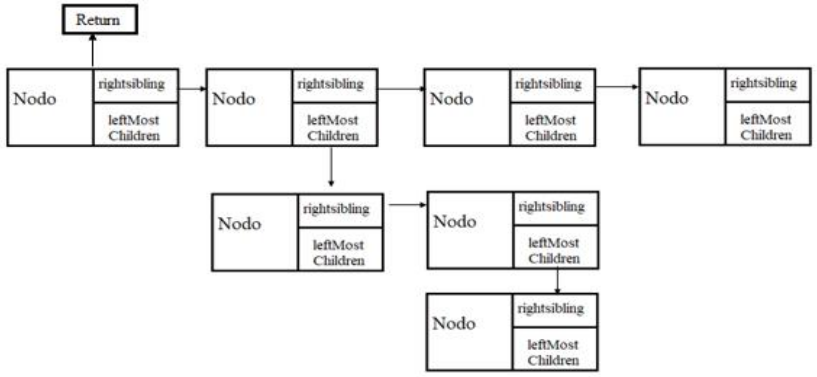
Explanation of the class

Explanation of ADT methods:

Tree methods:

- Parent:return the father of one node.
- Label:return the value the one node.
- Root:return the root of the tree.



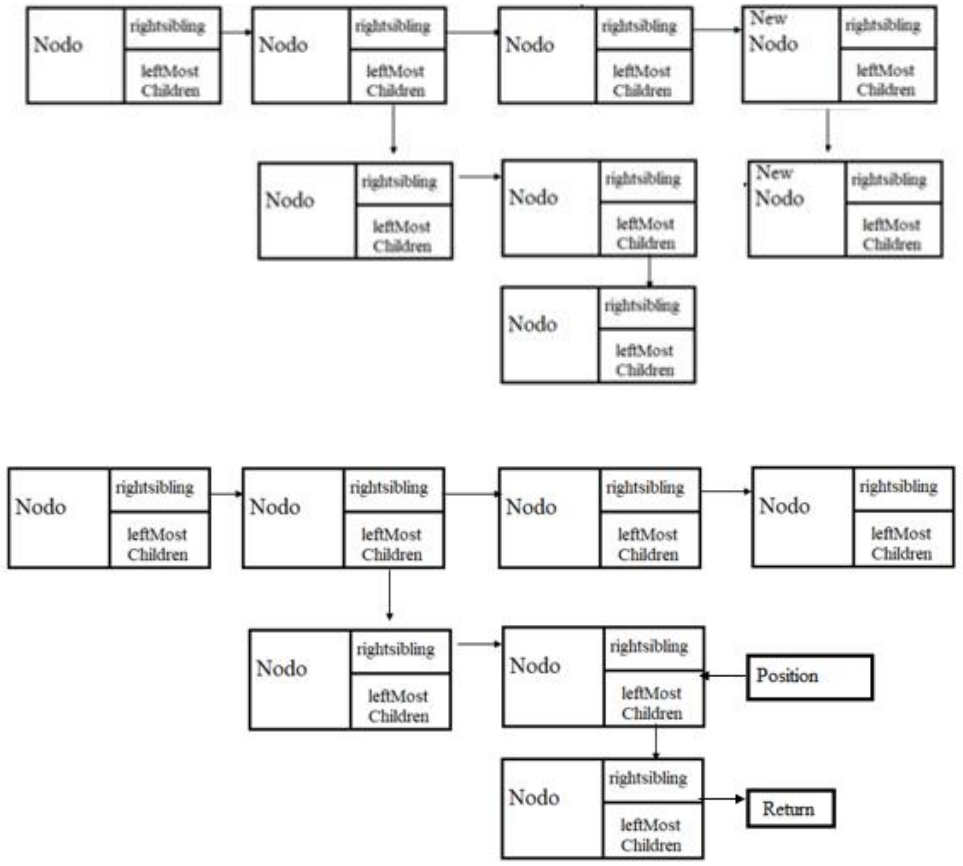


Explanation of the class

Explanation of ADT methods:

Tree methods:

- Create. Insert new nodo with its children
- LeftMostChildren:return the children

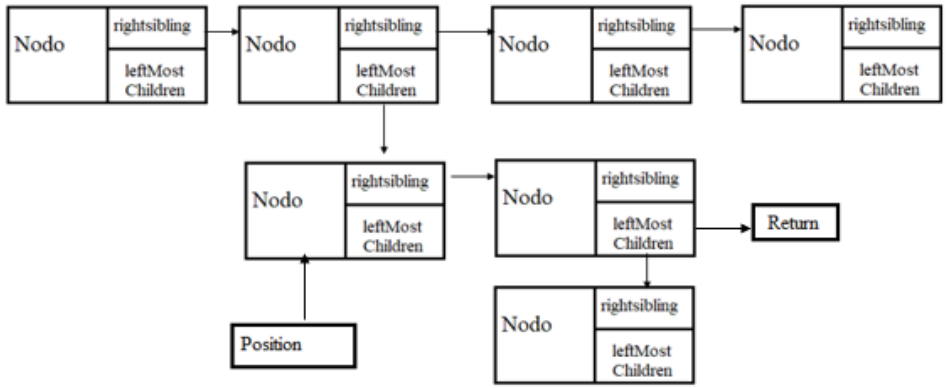


Explanation of the class

Explanation of ADT methods:

Tree methods:

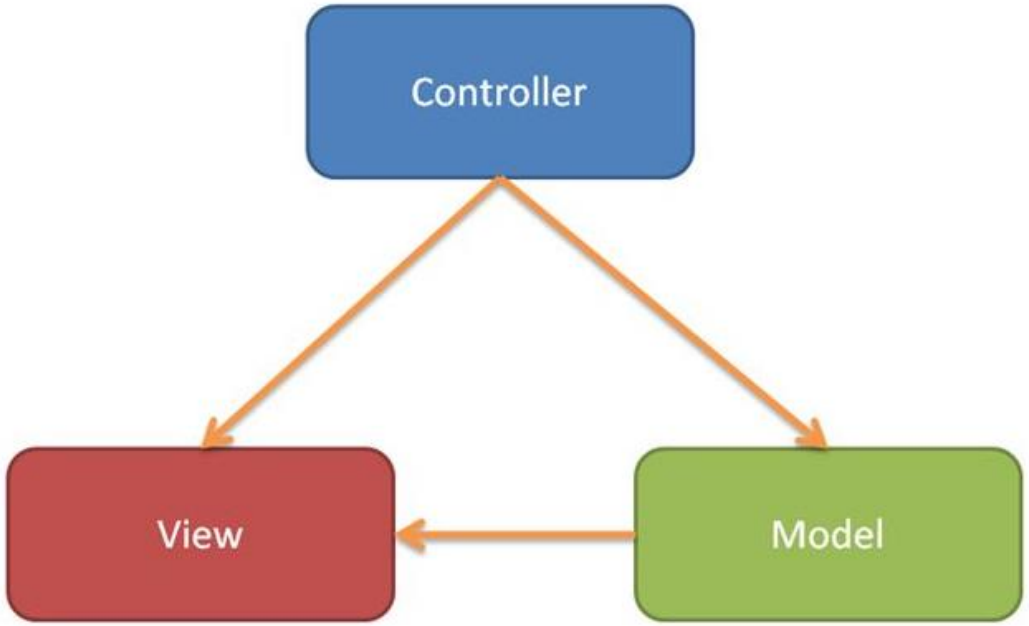
getrightsibling :return the right sibling of the node.



IMPLEMENTATION

Explanation of every difficult section of the program

The Project is splited in three parts(Model,View,Controller) following patter MVC.

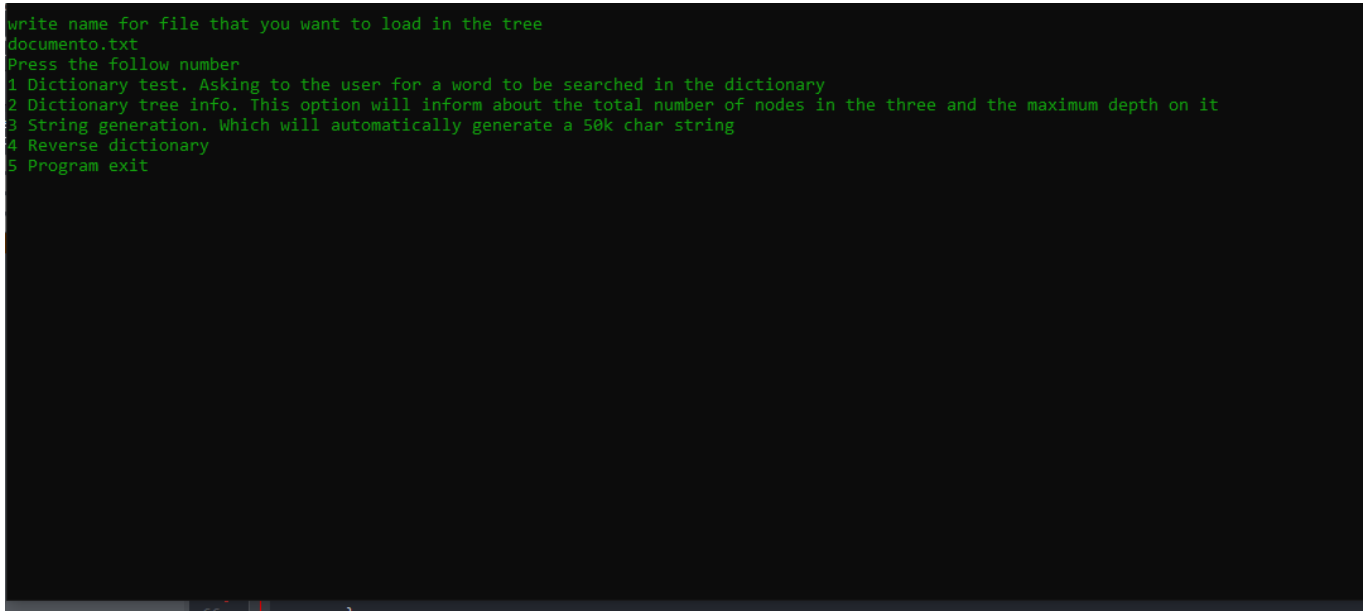


IMPLEMENTATION

Explanation of every difficult section of the program

Difficults in the View:

- How to generate a graphical interface, easy for the user and intuitive.

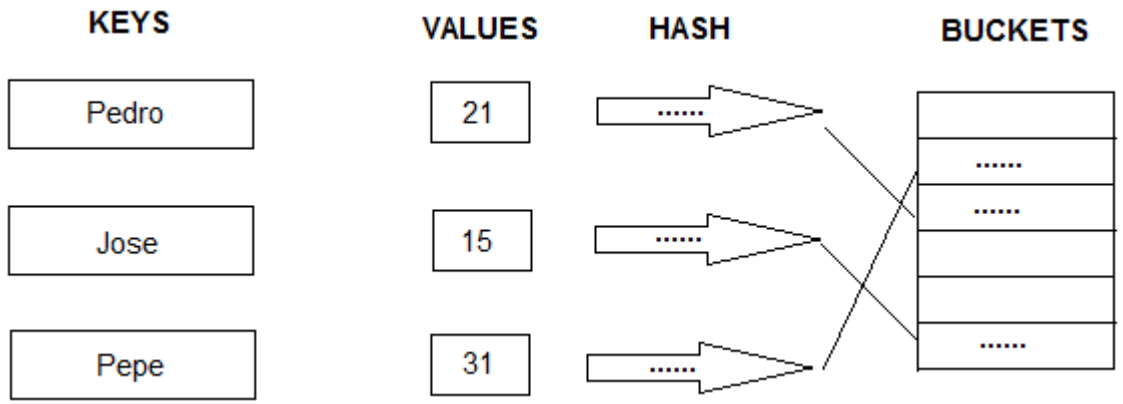


IMPLEMENTATION

Explanation of every difficult section of the program

Difficults in the Control:

- Generate the string without having to save all the letters in an array
- Count the duplicate words, of which have been generated, for this it has been decided to use an open hashmap.
- How to go through the tree to be able to reverse it.
- How to pass the file name as a parameter to the save or open function.
- Deal with SIGEVN errors when trying to read from null memory locations.



IMPLEMENTATION

Explanation of every difficult section of the program

Difficults in the Model:

- how to structure the tree and create its implementation
- The used the char * as an argument for the queue, stack and list methods, the memory reference was passed to it and if you wanted to continue inserting

Review

Running time of operations

- **Label**:
- getData()-> O(1)
- setData(char)-> O(1)
- **Node**
- getleftmostchild()-> O(1)
- setleftmostchild(char)-> O(1)
- setLabel(char) -> O(1)
- getLabel()-> O(1)
- getrightsibling()-> O(1)
- setrightsibling(char)-> O(1)

Tree

getRoot();-> O(1)
MakeNull()-> O(n)
create()-> O(logn)
getNumberNodes();-> O(1)
getDepth();-> O(1)

Data:

getData()-> O(1)

setData(char *)-> O(1)

Element

getElementData();
setPointer(Element*)
getElementPointer()

Stack

getTop()-> O(1)
isEmpty()-> O(1)
MakeNull()-> O(n)
push(char*)-> O(1)
pop()-> O(1)

Queue

Enqueue(char*)-> O(1)
Peek()-> O(1)
Dequeue()-> O(1)
isEmpty();-> O(1)
MakeNull()-> O(n)

Queue

Enqueue(char*)-> O(1)
Peek()-> O(1)
Dequeue()-> O(1)
isEmpty();-> O(1)
MakeNull()-> O(n)

List

Insert(Element *)-> O(1)
deletebyposition(int) -> O(1)
next(Element*)-> O(1)

Core

dictionaryTest()-> O(log)
dictionaryTreeInfo()-> O(log)
stringGeneration()-> O(n)
reverseDictionary()-> O(n)