RADAR STATION

DATA STRUCTURES ASSIGNMENT II

BY: GROUP PROFESSOR BTW

- IVÁN GARCÍA GARCÍA
- AARÓN VICENTE SANTOS
- BARTŁOMIEJ MARAJ

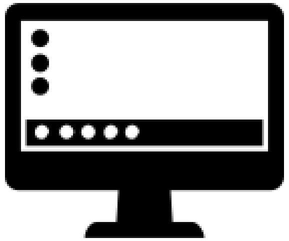
INDEX

1

ANALYSIS

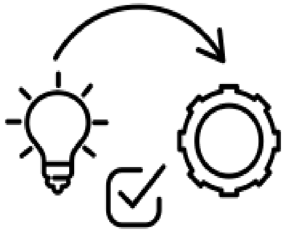
2

DESIGN



3

IMPLEMENTATION



4

DETAILS AND IMPROVEMENTS

1. ANALYSIS

*"A dissection of how the **Radar Station** works and theorically what have we used."*

RADAR STATION DETECTS DIFFERENT FLYING OBJECTS

For this, the program has different Options, in which it does different things for these options to work:

- **User can input information** that wants to be searched.
- **Selection of different characteristics** that the Flying Object can have.
- **Stores all the elements created** in each Session.
- **Creates correctly all the elements from 1000 to 10000 randomly** correctly and showing them in a compact way

1. ANALYSIS

ADT SPECIFICATION

"Showing what the assignment asks and what we implemented"

- **PRACTICE ASKS FOR:**
  - Linear ADT for the elements.
  - Non-linear ADT for the elements.
  - Arbitrary ADT for the Data Storage.

- **IMPLEMENTED:**
  - Linear ADT: **Lists, Stacks, Queues**
  - Non-linear ADT: **Hash Tables**
  - Arbitrary ADT for the Data Storage: **Stacks and Hash Tables**

1. ANALYSIS

IMPORTANT METHODS

"In this part, we talk about some methods that has been very useful and influential in this assignment, focusing on the Hash Tables."

- ***hashFunction():*** *in this method we use the hash table as it is meant to be used. We get the numbers of the plate that are going to be the KEY so we then use it on other ways to get information.*

- ***insert():*** *to put it simply, we add a new Flying Object into the Hash Table.*

- ***countFlyingObjectHash():*** *Goes through all the Hash Table, returning the amount of elements which values are the specified by the user.*

***all these methods are implemented in Hash.cpp and Session.cpp**

2. DESIGN

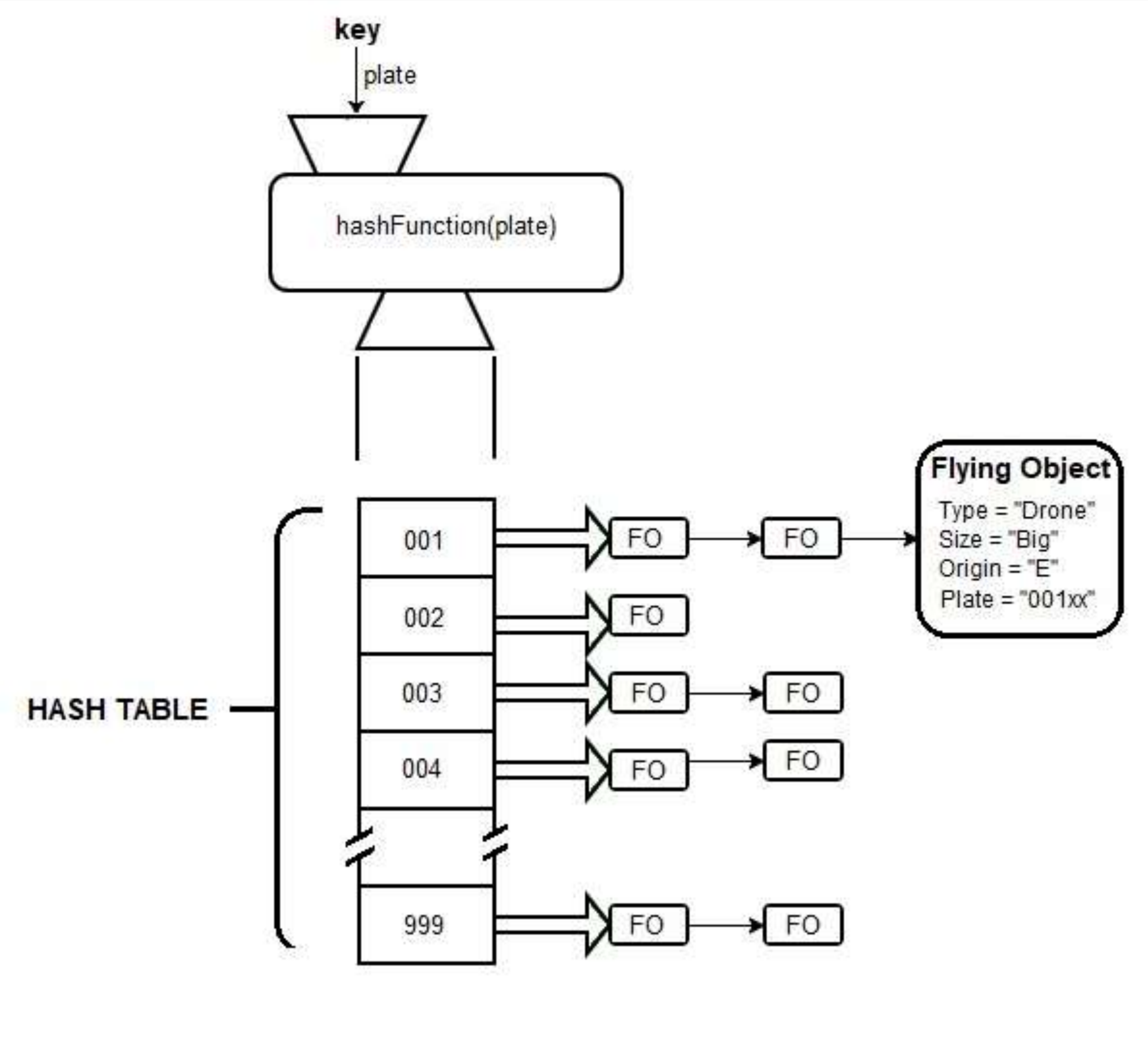
“Here we are going to show, the different diagrams we have made and used, to show how the different structures and program work”

2. DESIGN

RADAR STATION DIAGRAM

*"This is how the **Radar Station** works while having implemented the Hash Tables in it."*

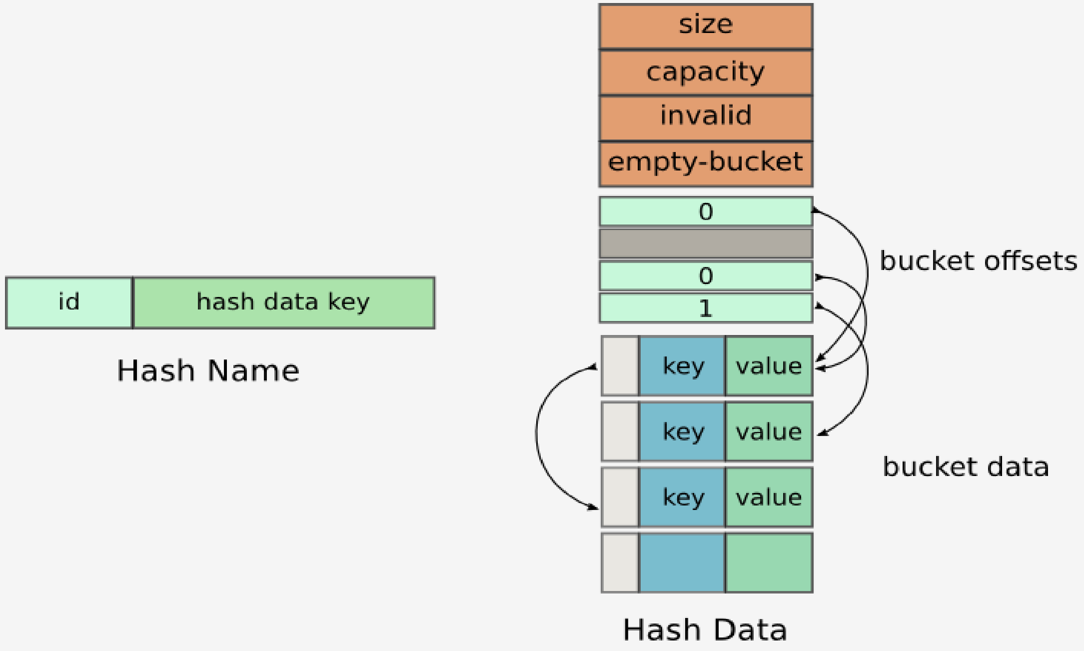
- It goes from 001 to 999, because of the numbers of the plates.

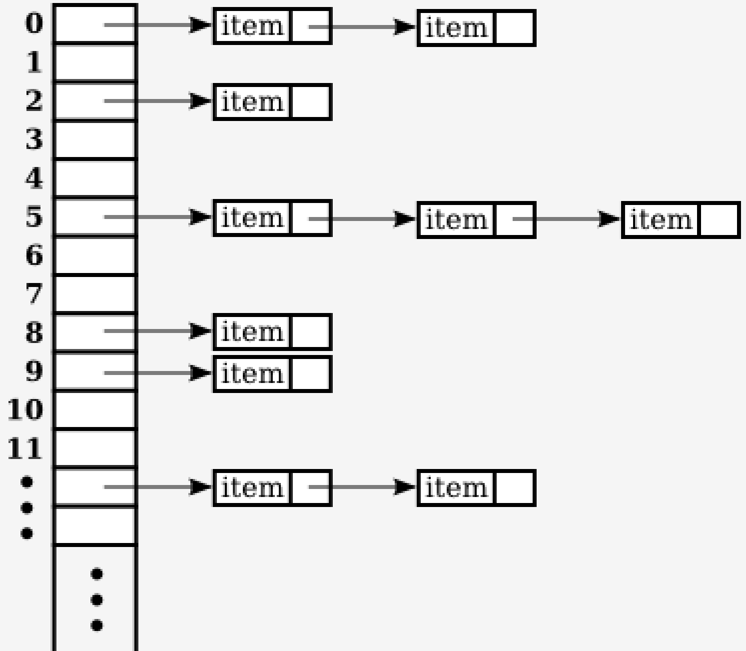


2. DESIGN

HASHTABLE DIAGRAM

"Here we can see different ways to show a Hash Table and how it works; it is used to store information in the program"

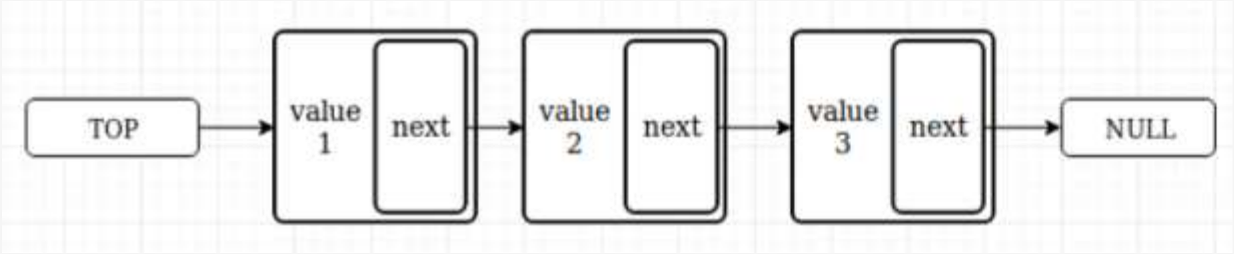


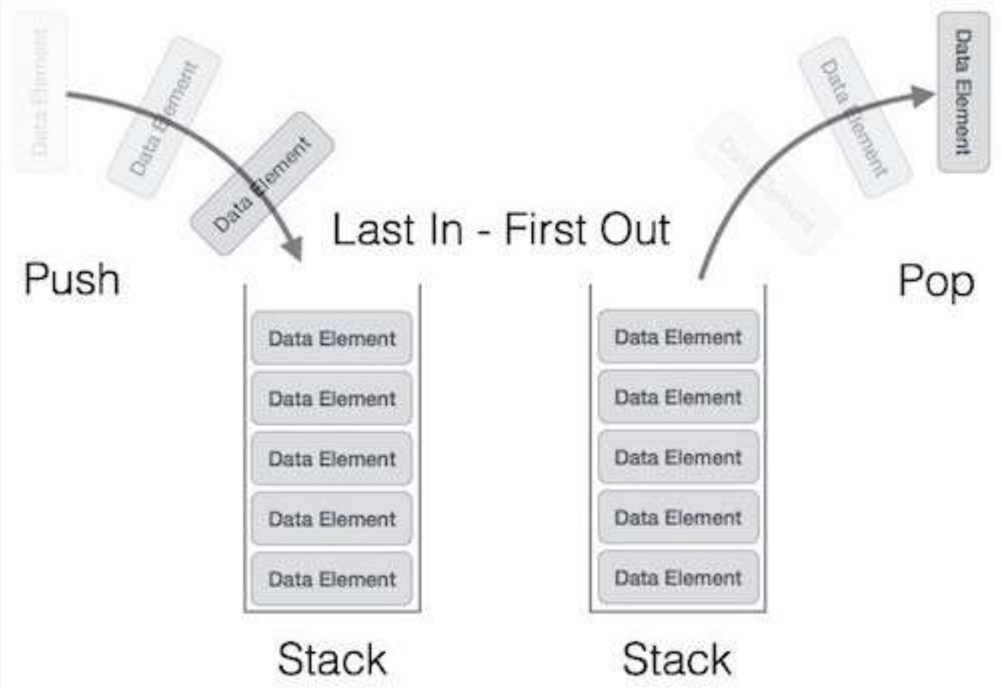


2. DESIGN

STACK BOX DIAGRAM

"Representation of a Stack, which is used to store the different information from the Sessions"

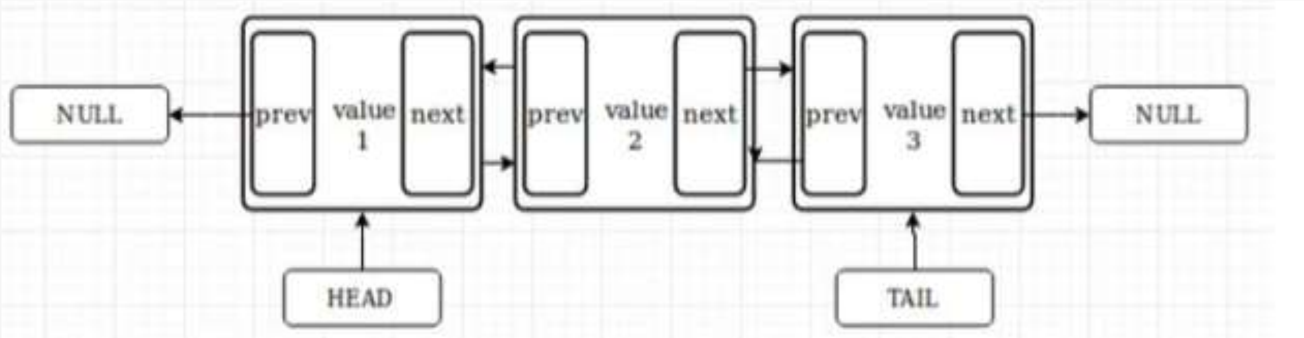


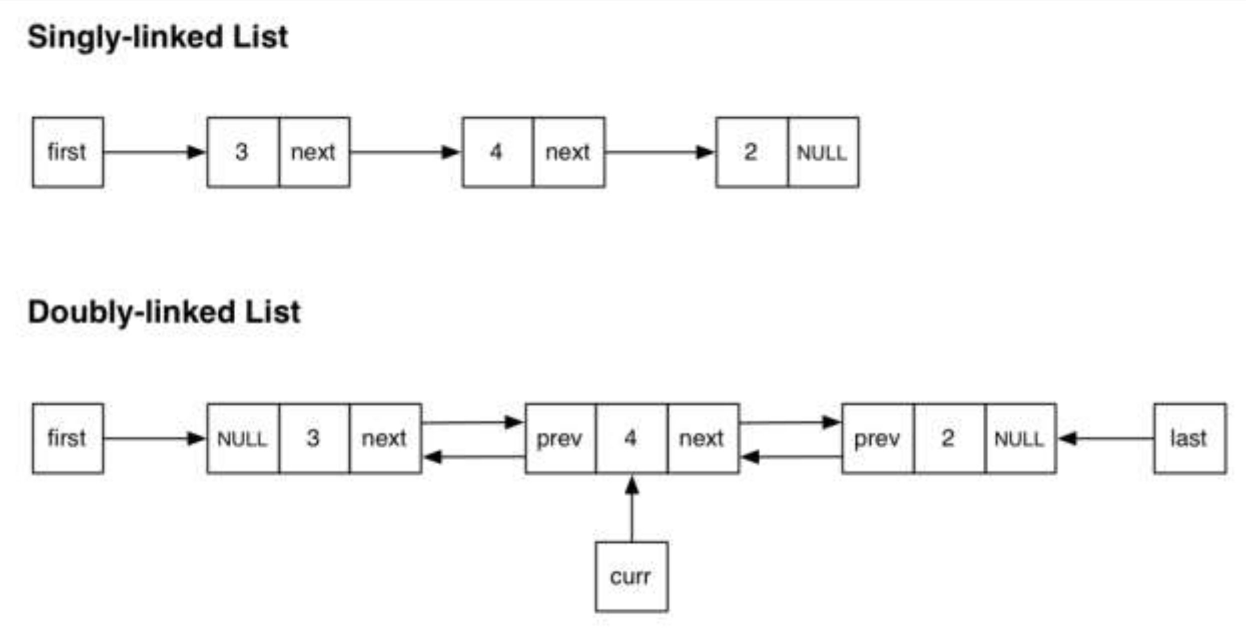


2. DESIGN

LIST BOX DIAGRAM

"Lists are used to implemented most of the functions and is directly related to the Hash Tables"

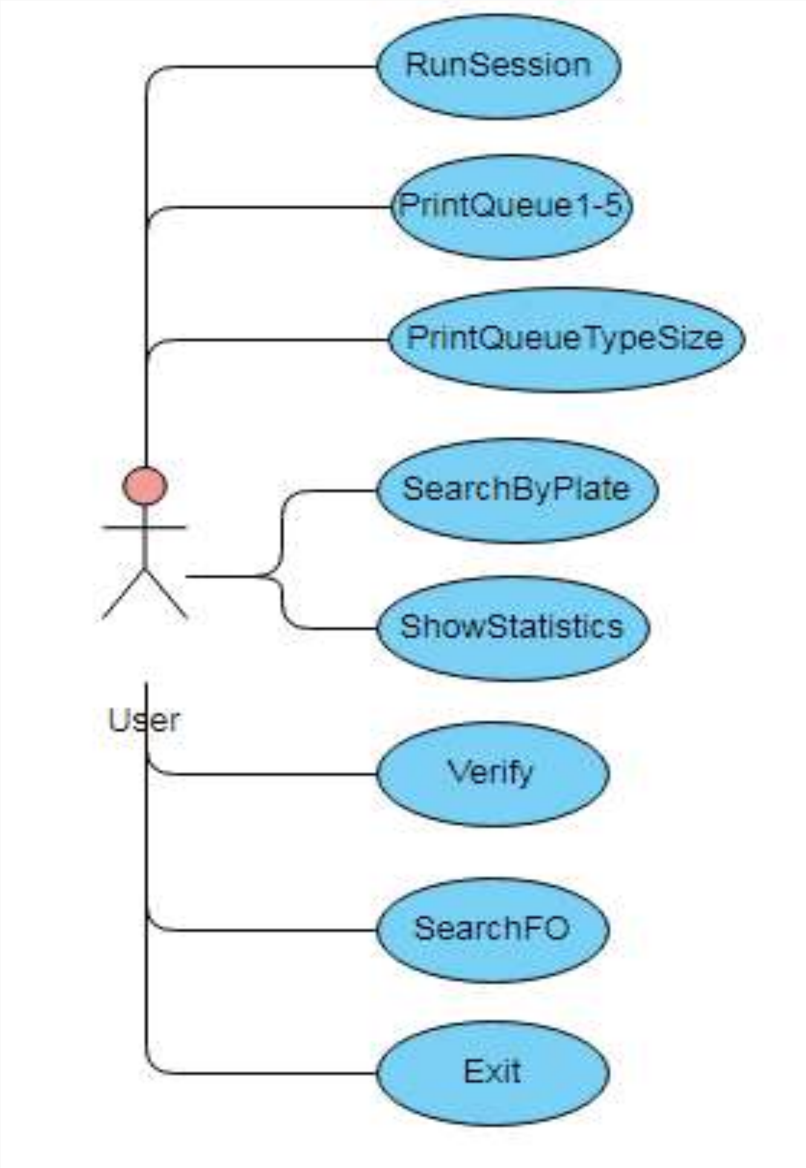




2. DESIGN

UML USE-CASE DIAGRAM

"In this diagram we can see how the User from the outside interacts with which function of the program (directly or indirectly), because in reality he justs plays with the interface and the different options"



2. DESIGN

UML CLASS DIAGRAM

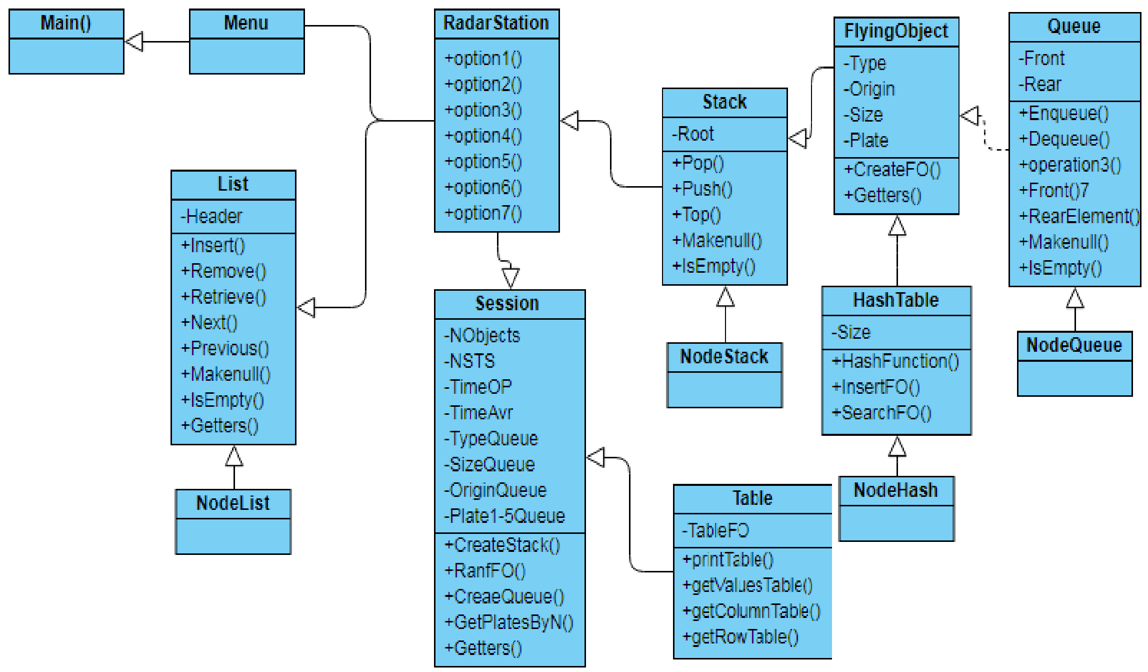
As some of the classes shown it this diagram were explained, we are going to jump into the new ones:

- <u>Node Classes</u>: we implemented the different classes just for not abusing the 'struct'

- *<u>HashTable</u>: implements the DS Hash Table, has the attribute 'size' and uses the methods Hash Function, InsertFO and SearchFO.*

- *<u>Table</u>: prints tables for Statistics*

- *<u>Menu:</u> it is just an interface class to print the options.*



3. IMPLEMENTATION

“The different methods, new and modified that we think should be shown here”

3. IMPLEMENTATION

NODE CLASSES

The creation of different node classes, to evade the use of struct was essential for us.

```
#ifndef NODEHASH_H
#define NODEHASH_H
#include <iostream>
#include "FlyingObject.h"

using namespace std;

class nodeHash
{
public:
        string key;
        FlyingObject object;
        nodeHash* next;
public:
        nodeHash();
        ~nodeHash();
};


#endif
```

```
#ifndef NODELIST_H
#define NODELIST_H
#include <iostream>
#include "Session.h"

class nodeList
{
        public:
                Session sn;
                nodeList* next;
                nodeList* previous;
        public:
                nodeList();
                ~nodeList();
};


#endif
```

```
#ifndef NODESTACK_H
#define NODESTACK_H

#include "FlyingObject.h"

using namespace std;

class nodeStack
{
        public:
                FlyingObject object;
                nodeStack* next;
        public:
                nodeStack();
                ~nodeStack();
};

#endif
```

```
#ifndef NODEQUEUE_H
#define NODEQUEUE_H

#include <iostream>

using namespace std;

class nodeQueue
{
public:
        string element;
        nodeQueue* next;
public:
        nodeQueue();
        ~nodeQueue();

};

#endif // NODEQUEUE_H
```

3. IMPLEMENTATION

HASH.CPP

*"Here we can see a part of the implementation of **hash.cpp**, which sustains mostly the second part of this assigment.*

*We can see **hashFunction** and **insertFlyingObject** which are pretty basic for everything we do"*

```
hashStructure::hashStructure()
{
        for (int i = 0; i < 1000; i++)
        {
                hashTable[i] = new nodeHash();
        }
}

hashStructure::~hashStructure()
{

}

int hashStructure::hashFunction(string plate)
{
        int firstNumber = (int)plate.at(0) - 48;                //ASCII
        int secondNumber = (int)plate.at(1) - 48;               //ASCII
        int thirdNumber = (int)plate.at(2) - 48;                //ASCII

        int index = ( firstNumber * 100 ) + ( secondNumber * 10 ) + thirdNumber;

        return index;
}

void hashStructure::insertFlyingObject(FlyingObject givenObject, string plate)
{
        int index = hashFunction(plate);

        if (hashTable[index]->key == "")
        {
                hashTable[index]->key = plate;
                hashTable[index]->object = givenObject;
        }
        else
        {
                nodeHash *linkedElement = hashTable[index];
                nodeHash *auxPtr = new nodeHash();

                auxPtr->key = plate;
                auxPtr->object = givenObject;
                auxPtr->next = NULL;

                while (linkedElement->next != NULL)
                {
                        linkedElement = linkedElement->next;
                }
                linkedElement->next = auxPtr;
        }
}
```

3. IMPLEMENTATION

OPTIONS 4 TO 7

"Here we can see a nice and clean clode about these options, that are in RadarStation.cpp"

OPTION 5: *Show the statistics, both in Stacks and Hash Tables.*

OPTION 6: *Verifies if the data is there in both structures, shows all of them generated and counts how many Superlopez... has been generated.*

OPTION 7: *This requests the user to put specific values of the FO and it'll show them in both structures.*

```
void RadarStation::option4()
{
        string plateNumbers;

        cout << "Enter the number to be searched ( xxx where x = 0-9 ): ";
        cin >> plateNumbers;
        cout << endl;

        printOption4(true, plateNumbers);
        printOption4(false, plateNumbers);
}

void RadarStation::option5()
{
        table tableFO = *new table();

        tableFO.printTable(true, lastSession);
        tableFO.printTable(false, lastSession);
}

void RadarStation::option6()
{
        printOption6(true);
        printOption6(false);
}

void RadarStation::option7()
{
        string type, size, origin, plate;
        stack searchedFlyingObjectsStack = *new stack();
        FlyingObject printedFlyingObject = *new FlyingObject();

        cout << "Introduce the specific values of the Flying Object: \n";
        cout << "Type(-1 to not specify the type): ";
        cin >> type;
        cout << "Size(-1 to not specify the size): ";
        cin >> size;
        cout << "Origin(-1 to not specify the origin): ";
        cin >> origin;
        cout << "Plate(-1 to not specify the plate): ";
        cin >> plate;

        printOption7(true, type, size, origin, plate);

        printOption7(false, type, size, origin, plate);
}
```

```
void RadarStation::printOption6(bool hash)
{
        unsigned t0, t1, t2, t3, totalTime;

        if (hash)
        {
                cout << "
                cout << "
                cout << "Flying Objects - Hash Table Data\n\n";
                cout << "      Plate          Size          Origin          Type          \n\n";

                t0 = clock();
                lastSession.getHashStructure().printAllFlyingObjects();
                t1 = clock();

                cout << "\nTime to print - Hash Table: " << (double(t1 - t0) / CLOCKS_PER_SEC) << endl;
        }
        else
        {
                cout << "\nFlying Objects - Stack Data\n\n";
                cout << "      Plate          Size          Origin          Type          \n\n";

                t2 = clock();
                lastSession.printAllFlyingObjects();
                t3 = clock();

                cout << "\nTime to print - Stack: " << (double(t3 - t2) / CLOCKS_PER_SEC) << endl;

                cout << "\nNumber of Tiny-Superlopez from Surprise: " << lastSession.getNumberSTS() << endl;
                t0 = clock();
                cout << "Number of Tiny-Superlopez from Surprise - Hash Table: " << lastSession.getHashStructure().countFlyingObject(
                t1 = clock();
                cout << "Number of Tiny-Superlopez from Surprise - Stack: " << lastSession.countFlyingObjects("Superlopez", "Tiny", "
                t2 = clock();

                cout << "\nTime to count Tiny-Superlopez from Surprise elements - Hash Table: " << (double(t1 - t0) / CLOCKS_PER_SEC)
                cout << "Time to count Tiny Superlopez from Surprise element - Stack: " << (double(t2 - t1) / CLOCKS_PER_SEC) << endl

                cout << "\nNumber of elements generated: " << lastSession.getNumberObjects() << endl;
                t0 = clock();
                cout << "Number of elements generated - Hash Table: " << lastSession.getHashStructure().countFlyingObject("-1", "-1",
                t1 = clock();
                cout << "Number of elements generated - Stack: " << lastSession.countFlyingObjects("-1", "-1", "-1") << endl;
                t2 = clock();

                cout << "\nTime to count all elements - Hash Table: " << (double(t1 - t0) / CLOCKS_PER_SEC) << endl;
                cout << "Time to count all elements - Stack: " << (double(t2 - t1) / CLOCKS_PER_SEC) << endl;


                cout << "
                cout << "
```

option 4 has already been explained in the past

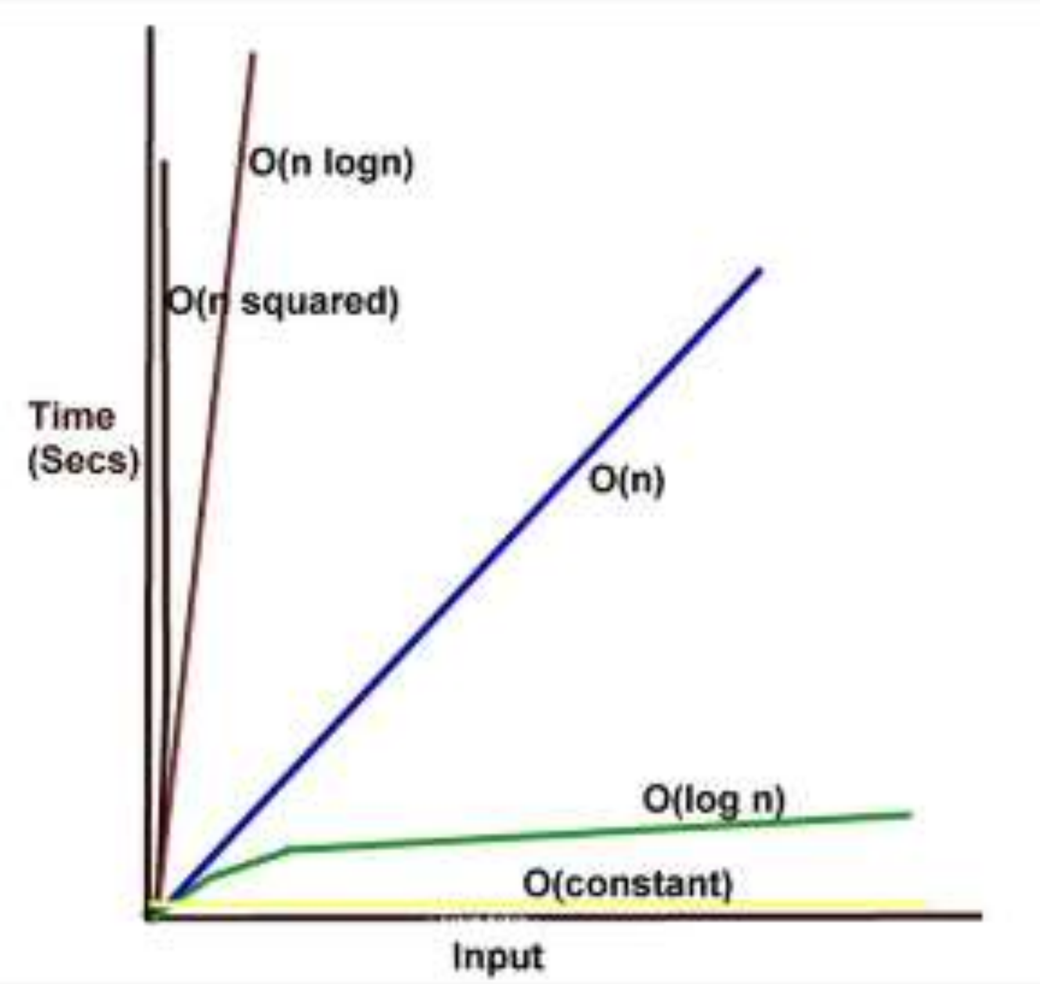
4. DETAILS AND IMPROVEMENTS

"In this part we want to show in an easy format how we have seen the running times, difficulties, what we have used, things that make our program better and improvements."

4. DETAILS AND IMPROVEMENTS

TIMES

"The times comparing to the theorical times with the ones we have when we execute the code, we obtain a close thing to O(1). In other case, when we start the program it could go to 20 seconds until it starts"



4. DETAILS AND IMPROVEMENTS

WHAT WE USED

"For developing all this program, we have used the C++ Programming Language, and within that using the terminal and Verbose for executing and testing the code."

C++

- ***OBJECT ORIENTED***
- ***POINTER OPERATIONS***

UI

- ***TERMINAL BASED USER INTERFACE***
- ***VERBOSE***

4. DETAILS AND IMPROVEMENTS

DIFFICULT SECTIONS & IMPROVEMENTS

"Here we are going to show in which areas we had more problems, and where can we improve the code."

DIFFICULTIES

- *TYPES OF EXECUTION TIME*
- *DATA STRUCTURE ALTERATION*
- DIFFICULTIES DEVELOPING OPTION 4

POSSIBLE ENHANCEMENTS

- *CLEANER CODE*
- *WORKING BETTER WITH C++ AND ITS TOOLS*
- *AN EVEN BETTER UI*

4. DETAILS AND IMPROVEMENTS

WHY OUR PROGRAM IS BETTER THAN YOURS?

BETTER USER INTERFACE

GOOD USE OF TERMINOLOGY AND UNDERSTANDING OF THE CODE

FOLLOWED THE TIPS GIVEN BY THE PROFESSOR

An optimist says:
"the glass is half-full"

A pessimist says:
"the glass is half-empty"

A programmer says:
"the glass is twice as large as necessary"

"After so many videocalls and work, we hope our program is a 10/10 and you liked our presentation"

GROUP PROFESSOR BTW IS OUT

THE END