



DATA STRUCTURES ASSIGNMENT II

BY: GROUP PROFESSOR BTW

- *IVÁN GARCÍA GARCÍA*
- *AARÓN VICENTE SANTOS*
- *BARTŁOMIEJ MARAJ*

RADAR STATION

INDEX

1

ANALYSIS



2

DESIGN



3

IMPLEMENTATION



4



DETAILS AND IMPROVEMENTS

1. ANALYSIS

*"A dissection of how the **Radar Station** works and theoretically 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.*

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

○ **searchFOTypeSize():** *one of the most useful ones; we specifcally look for some characteristics of the Flying Object and it will search for it.*

all these methods are implemented in Hash.cpp

2. *DESIGN*

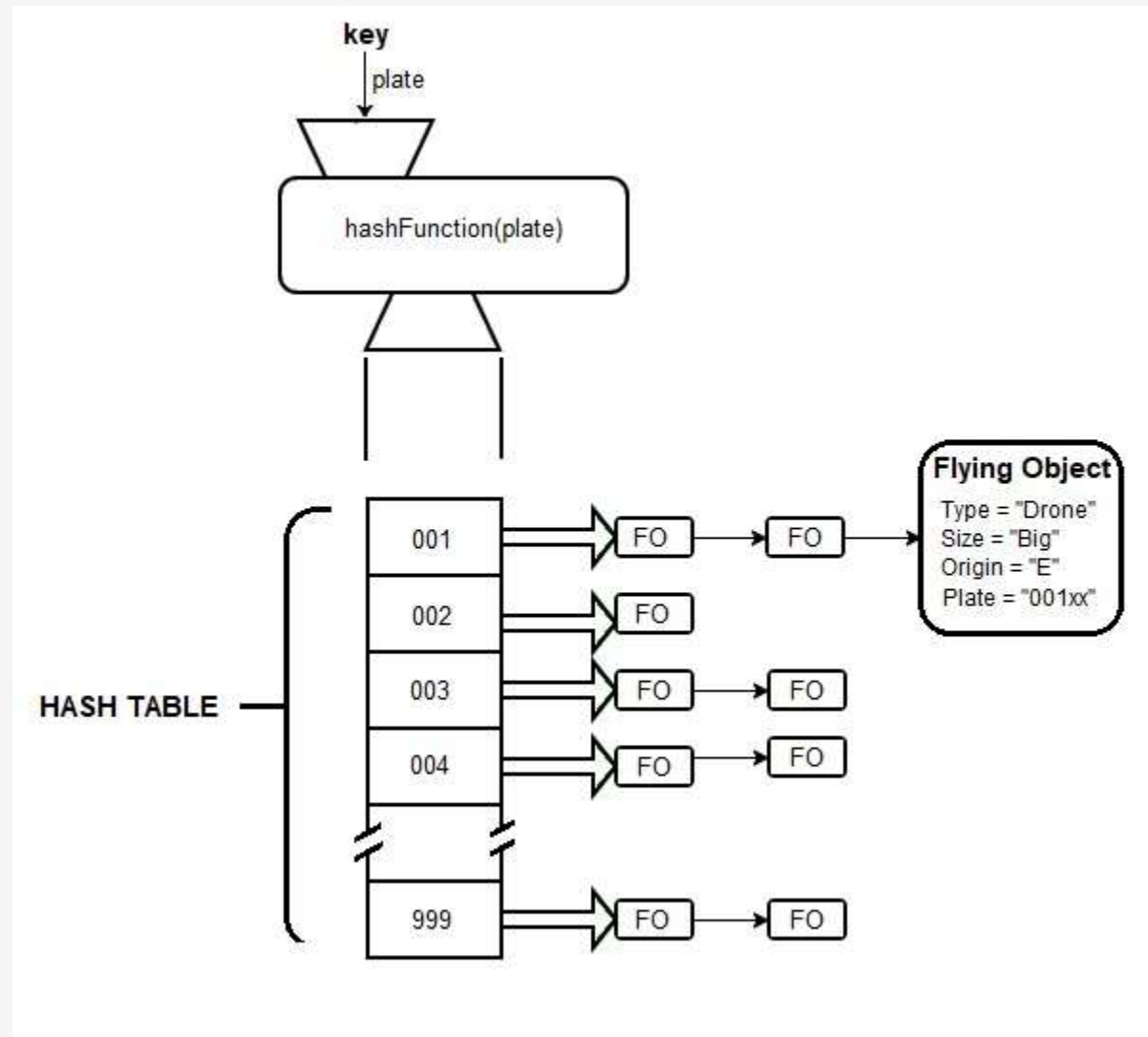
"Here we are going to show, the different diagrams we have made and used, to show how the different structures and program work"

2. DESIGN

RADAR STATION DIAGRAM

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

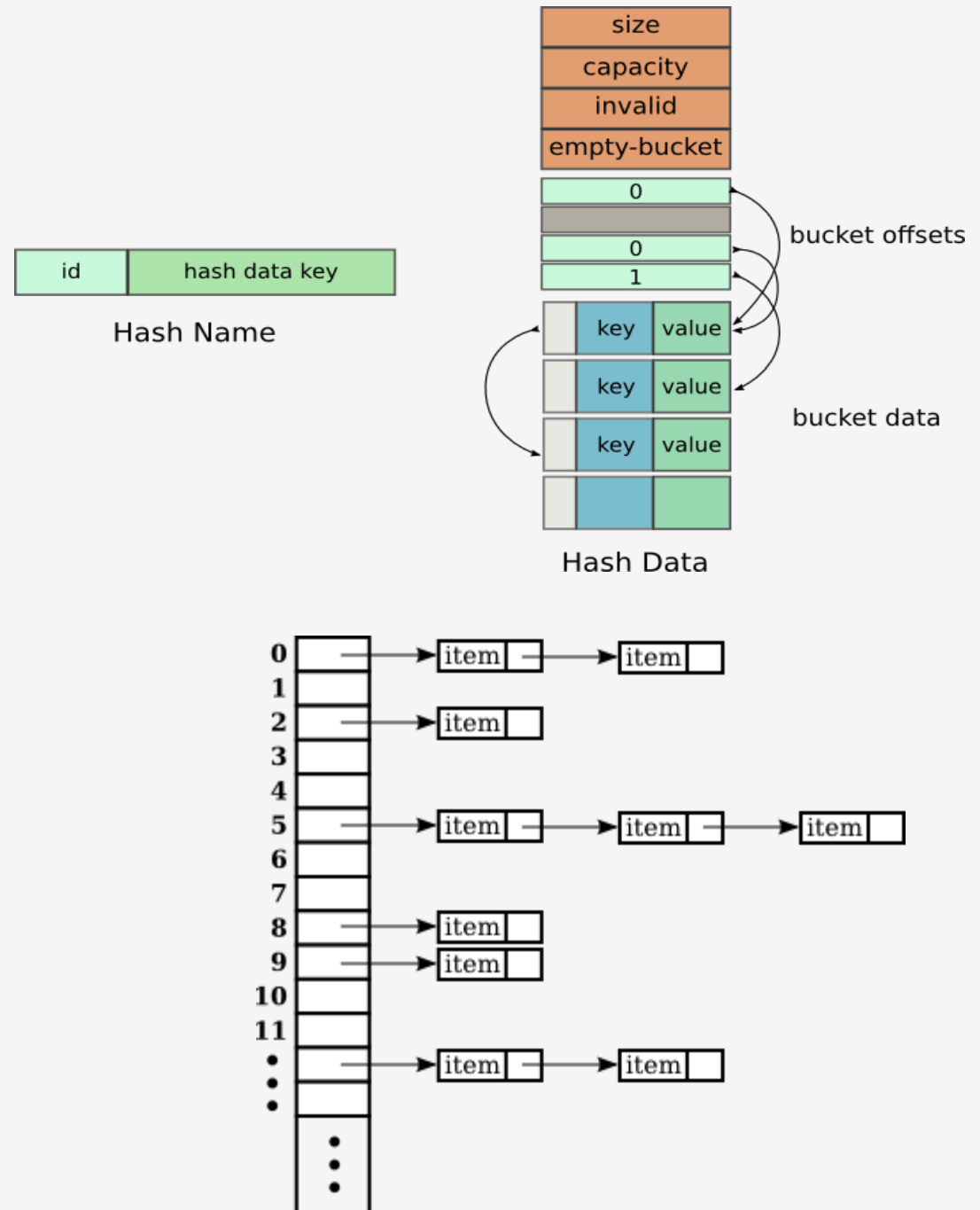
- It goes from 001 to 999, because of the numbers of the plates.



2. DESIGN

HASHTABLE DIAGRAM

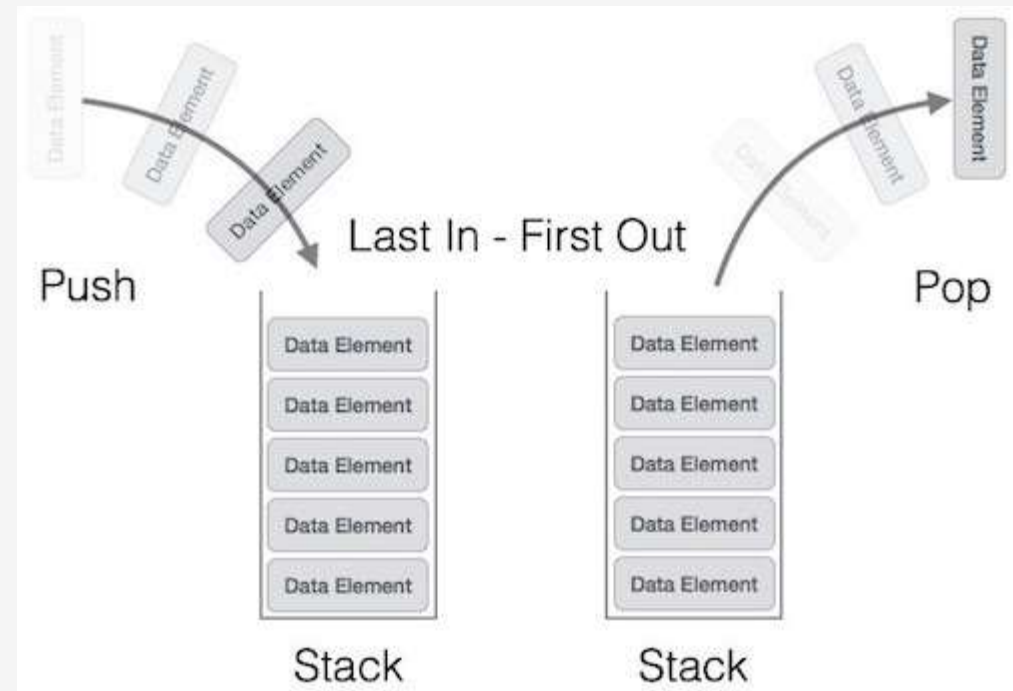
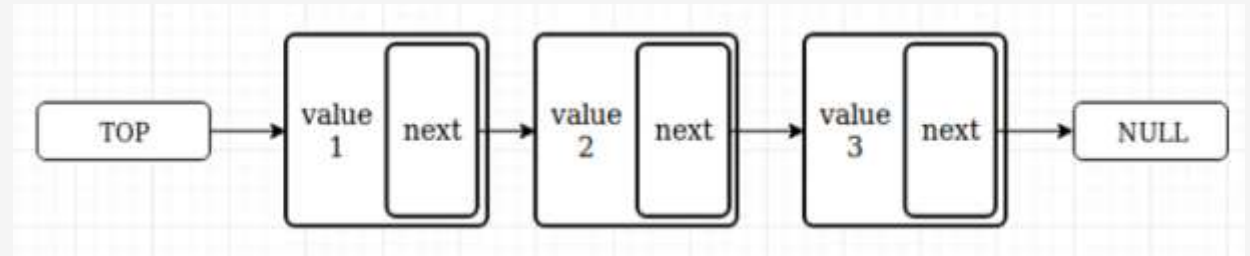
"Here we can see different ways to show a Hash Table and how it works; it is used to store information in the program"



2. DESIGN

STACK BOX DIAGRAM

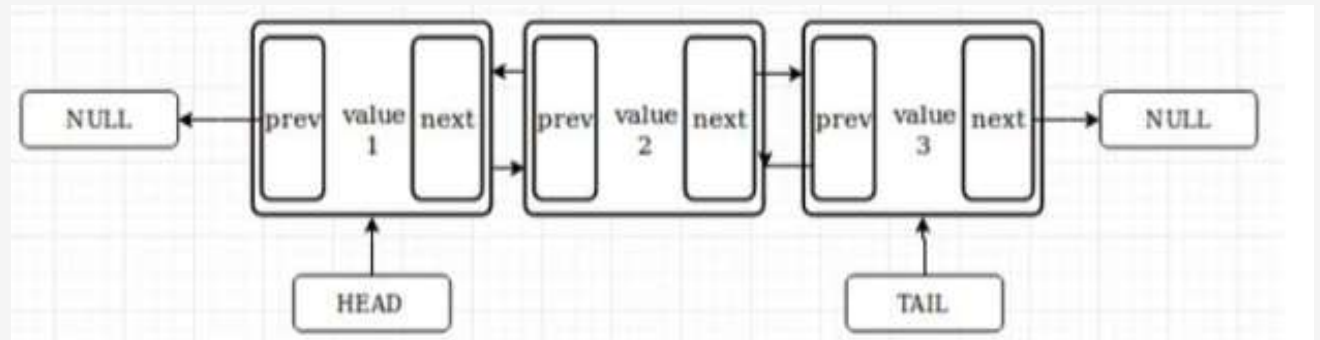
"Representation of a Stack, which is used to store the different information from the Sessions"



2. DESIGN

LIST BOX DIAGRAM

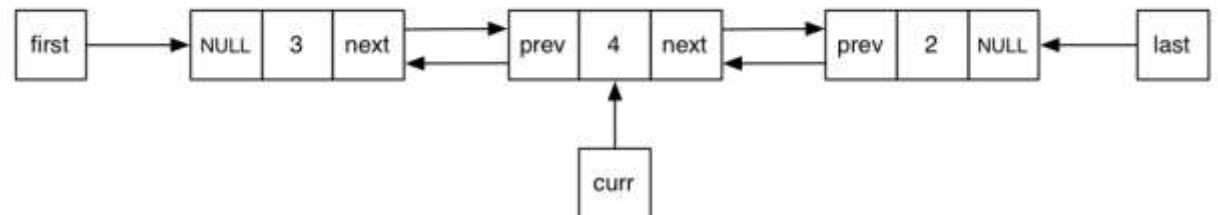
"Lists are used to implemented most of the functions and is directly related to the Hash Tables"



Singly-linked List



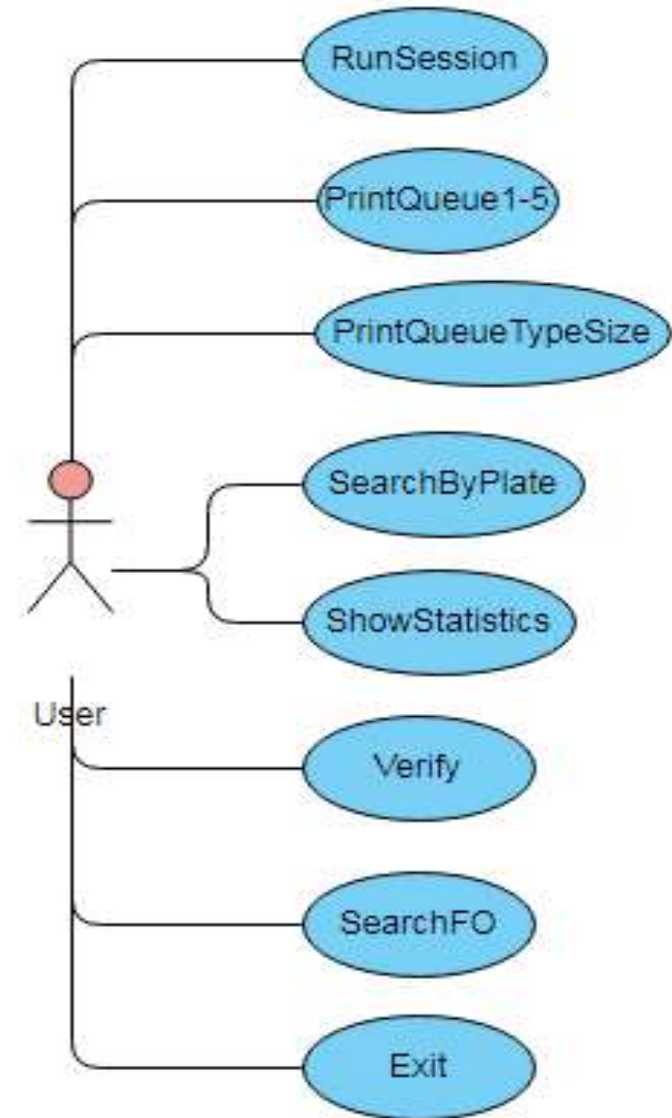
Doubly-linked List



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 just plays with the interface and the different options"

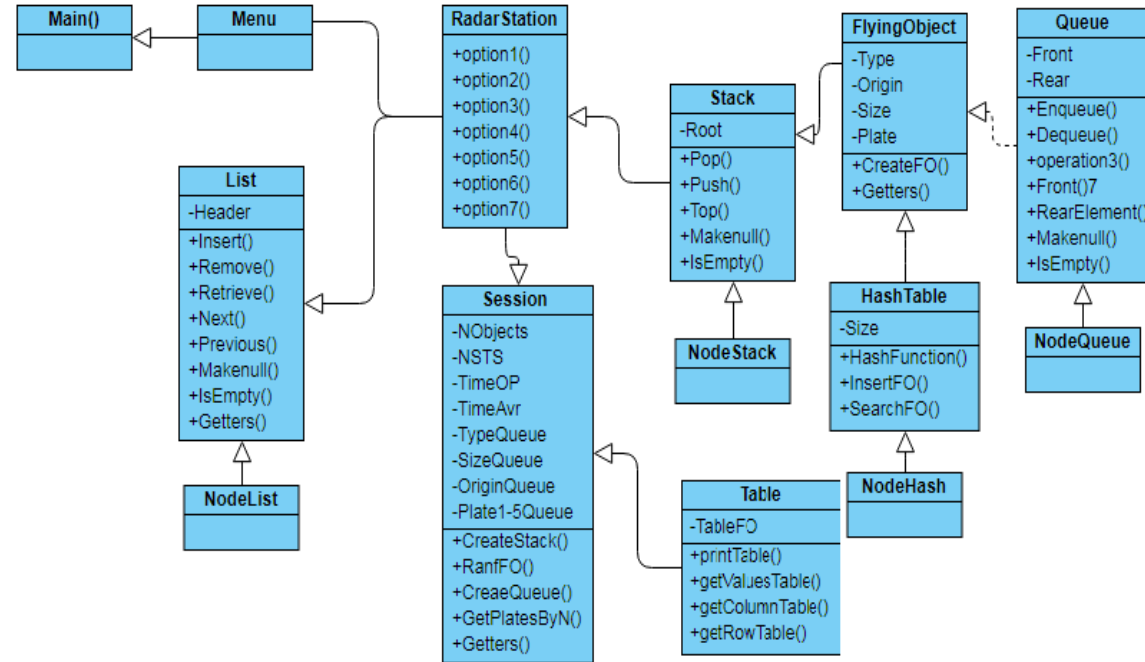


2. DESIGN

UML CLASS DIAGRAM

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

- **Node Classes:** we implemented the different classes just for not abusing the 'struct'
- **HashTable:** implements the DS Hash Table, has the attribute 'size' and uses the methods Hash Function, InsertFO and SearchFO.
- **Table:** prints tables for Statistics
- **Menu:** it is just an interface class to print the options.



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

3. IMPLEMENTATION

3. IMPLEMENTATION

NODE CLASSES

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

```
1  #ifndef NODEHASH_H
2  #define NODEHASH_H
3  #include <iostream>
4  #include "FlyingObject.h"
5
6  using namespace std;
7
8  class nodeHash
9  {
10 public:
11     string key;
12     FlyingObject object;
13     nodeHash* next;
14 public:
15     nodeHash();
16     ~nodeHash();
17 };
18
19
20 #endif
```

```
1  #ifndef NODELIST_H
2  #define NODELIST_H
3  #include <iostream>
4  #include "Session.h"
5
6  class nodeList
7  {
8  public:
9     Session sn;
10     nodeList* next;
11     nodeList* previous;
12 public:
13     nodeList();
14     ~nodeList();
15 };
16
17
18 #endif
```

```
1  #ifndef NODESTACK_H
2  #define NODESTACK_H
3
4  #include "FlyingObject.h"
5
6  using namespace std;
7
8  class nodeStack
9  {
10 public:
11     FlyingObject object;
12     nodeStack* next;
13 public:
14     nodeStack();
15     ~nodeStack();
16 };
17
18 #endif
```

```
1  #ifndef NODEQUEUE_H
2  #define NODEQUEUE_H
3
4  #include <iostream>
5
6  using namespace std;
7
8  class nodeQueue
9  {
10 public:
11     string element;
12     nodeQueue* next;
13 public:
14     nodeQueue();
15     ~nodeQueue();
16 };
17
18
19 #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 assignment.*

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

```
4  hashStructure::hashStructure()
5  {
6      for (int i = 0; i < 1000; i++)
7      {
8          hashTable[i] = new nodeHash();
9      }
10 }
11
12 hashStructure::~hashStructure()
13 {
14 }
15
16
17 int hashStructure::hashFunction(string plate)
18 {
19     int firstNumber = (int)plate.at(0) - 48;           //ASCII
20     int secondNumber = (int)plate.at(1) - 48;          //ASCII
21     int thirdNumber = (int)plate.at(2) - 48;           //ASCII
22
23     int index = ( firstNumber * 100 ) + ( secondNumber * 10 ) + thirdNumber;
24
25     return index;
26 }
27
28 void hashStructure::insertFlyingObject(FlyingObject givenObject, string plate)
29 {
30     int index = hashFunction(plate);
31
32     if (hashTable[index]->key == "")
33     {
34         hashTable[index]->key = plate;
35         hashTable[index]->object = givenObject;
36     }
37     else
38     {
39         nodeHash *linkedElement = hashTable[index];
40         nodeHash *auxPtr = new nodeHash();
41
42         auxPtr->key = plate;
43         auxPtr->object = givenObject;
44         auxPtr->next = NULL;
45
46         while (linkedElement->next != NULL)
47         {
48             linkedElement = linkedElement->next;
49         }
50         linkedElement->next = auxPtr;
51     }
52 }
```


3. IMPLEMENTATION

OPTIONS 4 TO 7

"Here we can see a nice and clean code 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.

```

151 void RadarStation::option4()
152 {
153     string plateNumbers;
154
155     cout << "Enter the number to be searched ( xxx where x = 0-9 ): ";
156     cin >> plateNumbers;
157     cout << endl;
158
159     printOption4(true, plateNumbers);
160     printOption4(false, plateNumbers);
161 }
162
163 void RadarStation::option5()
164 {
165     table tableF0 = *new table();
166
167     tableF0.printTable(true, lastSession);
168     tableF0.printTable(false, lastSession);
169 }
170
171 void RadarStation::option6()
172 {
173     printOption6(true);
174     printOption6(false);
175 }
176
177 void RadarStation::option7()
178 {
179     string type, size, origin, plate;
180     stack searchedFlyingObjectsStack = *new stack();
181     FlyingObject printedFlyingObject = *new FlyingObject();
182
183     cout << "Introduce the specific values of the Flying Object: \n";
184     cout << "Type(-1 to not specify the type): ";
185     cin >> type;
186     cout << "Size(-1 to not specify the size): ";
187     cin >> size;
188     cout << "Origin(-1 to not specify the origin): ";
189     cin >> origin;
190     cout << "Plate(-1 to not specify the plate): ";
191     cin >> plate;
192
193     printOption7(true, type, size, origin, plate);
194
195     printOption7(false, type, size, origin, plate);
196 }
197

```

****option 4 has already been explained in the past****

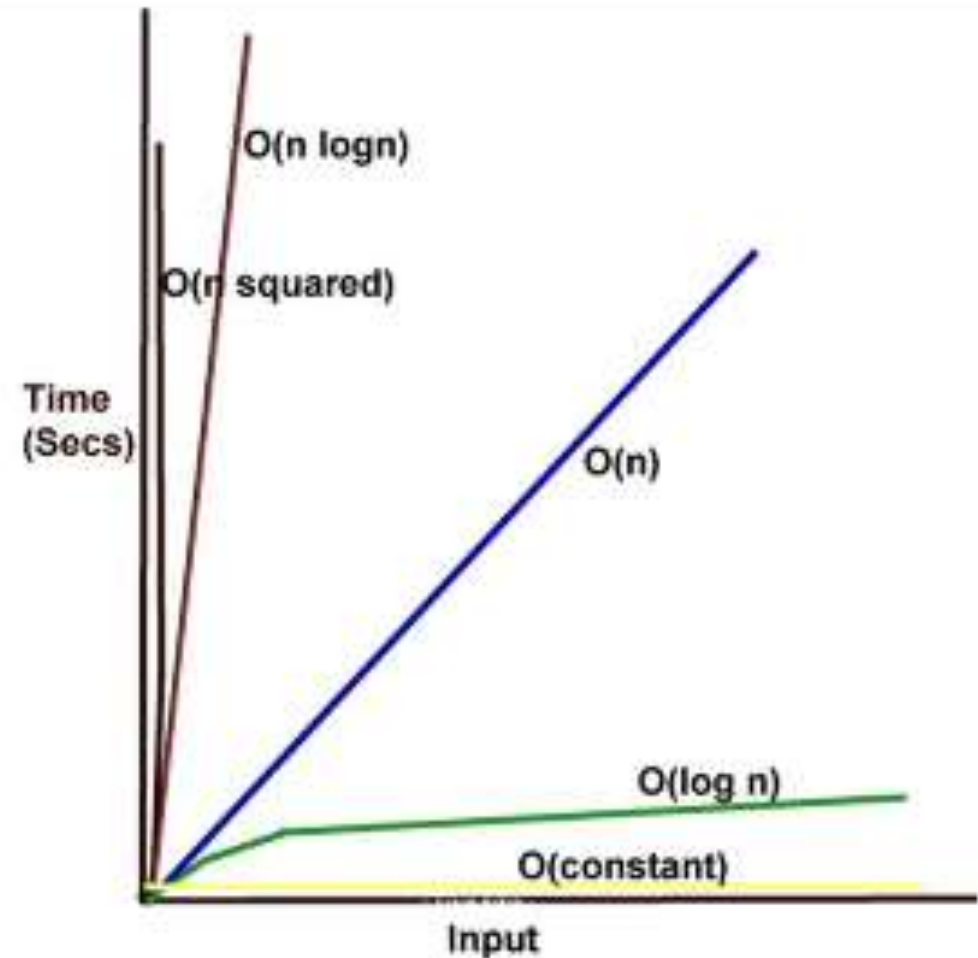
"In this part we want to show in an easy format how we have seen the running times, difficulties, what we have used, things that make our program better and improvements."

4. DETAILS AND IMPROVEMENTS

4. DETAILS AND IMPROVEMENTS

TIMES

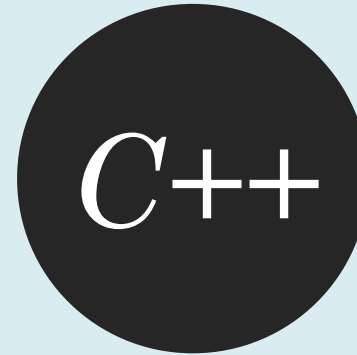
"The times comparing to the theoretical times with the ones we have when we execute the code, we obtain a close thing to $O(1)$. In other case, when we start the program it could go to 20 seconds until it starts"



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."



- OBJECT ORIENTED
- POINTER OPERATIONS



- 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”

me putting a java
command to c++
and it works



*"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