AIRPORT PROJECT

Intro:

During our second semester in engineering, we were asked to complete a fourth project. This one was different from the others because we were asked to make an interface to manage aircraft in an airport. It was initially difficult to understand the subject because there was a lot of information that had to be considered. Nevertheless, we chose to write on a sheet the various constraints and immediately the ideas began to come, and it was easier to work. We also distributed the work better than for the different projects and the fact of making 3 different files allowed us to advance much faster than for the other projects even if the difficulty did not stop increasing during the different projects. Finally, we hope that you will appreciate the work we have done.

Difficulties:

During this project we faced some difficulties, firstly it took us a certain time to fill the linked list because we had never really used them in C with codeblocks, we lost time to understand how to initialize them. Secondly and maybe the biggest problem that we had to face was to good rely the planes in the same time in a waiting list and in a list of company, thinking that a plane could be in 2 lists at the same time was a bit abstract. In fact,s we didn’t have much difficulties to fill the waiting list but when we had to rely all the plane to a company a fill them in the list of plane of this company was really long and hard for us. Then we encountered difficulties to read the event from a file because although we could had done it in the last project, we never saw this in class. And in last as our program was big, sometimes it took us time to find from where the errors were coming.

We had also a problem with the report of the project it’s the 4th that we have to do but we still have difficulties to understand what you are really waiting for. Maybe we explain too many functions, but for us they are all important in the architecture of the program.

Architecture of our program:

Now we will explain how our program work by explaining the most important functions, our program first fills the different lists or makes the user actions, and then each turn its doing all the actions to make the airport working, with slaying the different lists and change the avion of list and make them take off or landing. We will start by explaining the function to fill the list and then the functions to slay the airport.

Function AddAvion:

This function is the basis of our program, we call it all the 5 minutes because it is this functions that permit the user to enter some new planes or to choose to do some operations. This function will ask the user what he wants to do on the airport and in function of the choice we call another function to do the action asked by the user. Thanks to this function that we fill the different list of planes. We will now see the different functions used in function of the choice of the user.

Function user:

When the user wants to enter a new plane whatever the list where he wants to put it, we call the function user, in fact it asks the user for the different feature of the plane. First, he fills a plane with its name and depending on the type of this plane it will ask for the carburant and the consumption or the take-off time, in fact we enter a different variable in this function depending on what the plane is waiting for, a landing or to take off. If the plane is waiting for landing, we put 'B' in the feature heure_decollage to recognize it easily and if it is waiting for take-off we put -9999 the feature carburant for the same reason. When we have fill the basic information of the plane then it asks the company of the plane and it searches in the list of the company if this company exist or not, if it exists then we put the address of the existing company in the company feature of the plane. But if the company doesn't exist then we create a new company "box" and then fill this new company and rely the plane to this company, we initialise the ListeAvion of this company to null to easily fill this list in the functions focusing on company.

Function AddLandingList:

This function is used to fill the landing list, first we create a new cellule_avion and put our plane in this cellule, then we need to fill the feature precedent_company and suivant_company of this cellule_avion. We use the function AvionCompanyArray to do this, we will explain this function just after. But we also must put this cellule_avion at the good place in the landing list. So, we check the time that the plane can stay in the air of all the plane which are already in the landing list and place the plane before the plane which can stay more time in the air. It is done to have a list classified from the plane which can stay the less time in the air to the one which can stay the more. We use the algorithm below to add the new cellule_avion in the linked list that is the landing list.

Function AvionCompanyArray:

This function is used to add a plane to the ListeAvion of a given company, we enter a plane in the function and then it adds the company to the list of company thanks to the function AddCompany, which check if the company always exist or not and if not add it to the and of the list of company. Then we course the whole list of company to find the company of the plane which has been added by the function AddCompany or which already exist. When we have find this company, we add to the end of the ListAvion of this company, thanks to the function addLast, the plane received by the function. And as the company is the same for all the plane of this company, then all the plane->company->ListeAvion of the company will be modified with a new plane at the end. We use this function each time we create a new cellule_avion, without looking to the list where it is.

<u>Function Addlast:</u>

Add to the end of the double linked list of the plane of a company, a new plane. It changes the attribute precedent_compagnie and suivant_compagnie of the last and of the new plane to fill good this list of planes of company. It's thanks to it that we rely all the plane of a given company.

<u>Function AddTakeOffList:</u>

This function is used to add a plane to the take off list. We give a plane to this function then it will create a new cellule_avion, copy the feature of the plane in the plane of this cellule_avion and fill the feature precedent_company and suivant_company thanks to the function AvionCompanyArray described previously. Then it will just add the new cellule_avion in the top of the linked list, that is the take-off list.

<u>Function RemoveName:</u>

The function remove name receive a ListeAvion and use a while loop to run through this list and search a plane by comparing the name of all the plane of the list to the name given by the user that the function receives too. When the while loop finds the good plane, it must change the path of the linked list which is ListeAvion to remove the plane. And it also changes the path of the double linked which is the list of plane in the company of this plane. So, this function searches a plane and cut all the link of the other planes to this plane to remove it from the ListeAvion.

<u>Function AddEmergency:</u>

This function receives the name of a plane and remove it from the landing list using the function RemoveName described above, then using the function AddLastQueue, it creates a new cellule_avion with the plane wanted and add this cellule to the end of the emergency queue.

<u>Function ListeBlackListed:</u>

This function is filling the list of all the blacklisted company using a special structure cellule_blacklist, in fact the cellule_blacklist has as feature companyname which is a string and suivant which is the next blacklisted company. This list of blacklisted company is useful because once it is filled each turn it will run through this list to see if there is any plane of a blacklisted company and if there is do what it has to do with them thanks to the function CheckBlacklist that we will explain later. So ListeBlackList is the function which receive a name of company to black list which add this company in first position of our list of blacklisted company.

<u>Function LorT:</u>

This function run through the ListeAvion of a given company and check the different feature of the planes of the company to count the number of plane at landing or waiting for take, thanks to the 'B' or the -9999 that we put in the function user we can recognize where each plan is. Then if the function finds a plan with -9999 in carburant feature which means it waiting for take off it will check if it is not late by comparing the time to the heure_decollage feature of the plane that we convert into an integer using our strange formula.

<u>Function ReadTestFile:</u>

This function is used to fill the different ListeAvion or Queue using a text file. So, it opens the file and then read all the element of the event one by one, we know that a line is: NAME CODE FEATURE. We are scanning all the separated by a space or a back to the line, one by one. So first we create a plane and read its name, then we read the code, depending to the code we enter the feature of the plane and the plane in the good list, then we reinitialize the code and start a new line with a name and code…To identify that it’s the name of the plane that we are reading we compute its length, for the code we compare to the different code that we know and for the feature using the code we know the form of the element to scan. As we don’t know the company of the plane we ask this user for it. Then we fill the plane using the same syntax as in user once we have recovered all the information, and we enter it in the good list using one of the function described before.

<u>Function Blacklist:</u>

This function receives a name of a blacklisted company. Then, this function will browse the list of companies and compare the names of companies with the name of the blacklisted company received by using a while loop with the conditions that no match have been found and not to be outranged of the list of company. we will put a variable “exist” at 1 if a match has been found. Then, we use the function length which allows us to know the number of planes the company possess by returning “cpt”. After this, we use another while loop with the condition that a variable named “i” equal to 0 doesn’t reach our cpt received from length function. For each plane in the company, we will first remove them from the list company. However, planes are in both list, company and another (either takeoff list or landing list) To know in which they are we use this method:

-If there is a “B” at the Takeoff hour, it means that the plane is about to land, so a blacklist plane in the air must be put in emergency situation so we remove it from list landing by using the function removename, then we put it at the end of the queue of emergencylanding by using the function addlastqueue.

-And if its carburant is equal to -9999, it means that the plain is about to takeoff and a plane from a blacklisted company which is about to takeoff has to be delete. That’s why we use the function removename to delete it from ListeTakeoff.

Finally, we increment i to go to the next plane of the company.

<u>Function CheckEmergencyFirst:</u>

This function checks if a plane in flight needs to be put in an emergency siutation due to a lack of fuel. In fact, if the fuel the plane has over its consumption is equal or inferior to 2, we remove the plane from list landing and we attribute this celulleavion to the variable save by using the function RemoveFirst and we add it in first position of the emergency queue by using AddFirstQueue. Both function will be explained next.

Function RemoveFirst:

For this function it can receives an integer to know what to do:

-If it receives a number different from one, we cut the link of the celulle_avion in the list suivant_attente and we cut the link of the plane with the other in the list of its company.

-If it's 1, we cut only the link of the plane with the other in the list of its company.

In each of these cases, we return the plane, we do the function separate like this because in our function Blacklist, we will use the Function RemoveFirst but we won't need to do what we do when the function receives a number different from one because the same action will be done later in the function Blacklist.

Function AddFirstQueue:

This function is used to add a plane at the first position of a queue. We create a plane called New, then we add it to the list avion of its company by using the function AvionCompagnieArray. Then, we check if the queue is empty or not, if it's empty we say that New is equal to the first and the last of the queue. If the queue is not empty, we attribute to the suivant attente of our New plane the one which was the first at the beginning. And we attribute to the first of the queue our New plane.

Function AvionToHistoric:

This function is used to write in our historic file what happens in our airport. Firstly, we open our historic file so that we can write in it. Then, depending on what happen, and we know this thanks to if condition we manage to write in the historic file with a certain syntax the different events which occur.

Let's take one example, if it's a Takeoff spotted by one of our if conditions, then we will first use our function RemoveQueue to remove the plane which has takeoff from the list takeoff and we will print in Historic the different details you are asking for.

Algorithms:

Here we will explain three algorithm we resume the different way to fill a linked list at all position, remove an element from a linked list at different position.

```
Function.AddLandingList(ListeA : Liste_Avion, A : avion, ListeC : Liste_Compagnie, New : Cellule_Avion) :
Liste_Avion

Changed parameter : ListeC, New

Copied parameters : ListeA, A

Local variable : Rides, Rides2 : integer

                 Pred, Curr : Cellule_Avion

BEGIN

Pred <- NULL

Curr <- ListeA

New.Avion <- A

AvionCompagnieArray(New, ListeC)

If(ListeA = NULL)

        New.suivant_attente <- ListeA

        Return New

End if

Rides = New.Avion.carburant / New.Avion.consommation

Rides2 = Curr.Avion.carburant / curr.Avion.consommation

While(Curr.suivant_attente =/= NULL AND Rides>Rides2)

        Pred <- Curr

        Curr <- Curr.suivant_attente

        Rides2 <- Curr.Avion.carburant / Curr.Avion.consommation

End while

If(Rides2 >= Rides AND Pred = NULL)

        New.suivant_attente <- ListeA

        Return New

End if

Else if(Curr.suivant_attente = NULL AND Rides<Rides2)

        Pred.suivant_attente <- New

        New->suivant_attente <- Curr

End else if

Else

        Curr.suivant_attente <- New

        New.suivant_attente <- NULL

End else
```

Here we can see an algorithm which insert a new avion either in the first, last or at a given place a new Cellule_Avion so this algorithm resume how we insert all our plane in the different linked list.

```
RemoveName(ListeA Liste_Avion, emname[7] : array of character, ListeC : Liste_Compagnie) : Cellule_avion
Changed parameters : ListeA, ListeC
Copied parameter : emname
Local variable : curr, pred, save : Cellule_avion
                 Test : Cellule_compagnie
save = NULL
Test = ListeC
curr = ListeA
pred = NULL
while(curr =/= NULL)
        if(strcmp(emname,curr->Avion.identifiant) == 0)
                if(curr == *ListeA)
                        save <- ListeA
                        ListeA <- ListeA.suivant_attente
                End if
                else
                        save <- pred.suivant_attente
                        pred.suivant_attente <- curr.suivant_attente
                end else
                if(curr.precedent_compagnie = NULL)
                        if(curr.suivant_compagnie =/= NULL)
                                curr.suivant_compagnie.precedent_compagnie <- NULL
                        end if
                        while(strcmp(curr.Avion.compagnie.nom, test.compagnie.nom)=/=0)
                                test <- test.suivant;
                        end while
                        test.compagnie.ListeAvion <- curr.suivant_compagnie
                end while

                else if(curr.suivant_compagnie = NULL)
                        curr.precedent_compagnie.suivant_compagnie <- NULL
                end else if
                else
                        curr.precedent_compagnie.suivant_compagnie <- curr.suivant_compagnie
                        curr.suivant_compagnie.precedent_compagnie <- curr.precedent_compagnie
                end else
        end if
        pred <- curr
        curr <- curr.suivant_attente
end while
return save
```

Here it’s an algorithm which delete a Cellule_Avion either in first, last or at a given position so this algorithm resume how we delete the plane from all the linked list in our program.

Conclusion:

To conclude, this project was not easy to achieve. Indeed, as said before, we had some problems. Nevertheless, we managed to solve them all, or almost. It has been a rich year in programming education and after this fourth project, we can say that these projects are very useful to master what we are learning in class. Despite the hours spent on it, we will remember every feeling felt when the project is over. In all the values that programming teaches us, it is that of perseverance that comes back to us most, never giving up, even though the code does not run or it presents errors, it is by continuing to try that we learn. This is on what we want to conclude our fourth and final project (hopefully) of L1.