Oğuzhan Ertekin
21946113
BBM 203
30/11/2021

1) Introduction:

The Project is creating simple “employee recording system” application. We can add/remove employee, get employee’s info, listing all of employees or we can do things like these operators with this application. Therefore, companies that using this application can hold on all employees’ infos that working at company and they can reach or change employees’ infos or separate employees temporary or permanent.

2) Method:

I generally used ‘for’ and ‘while’ loops.Because we are expected to make a program that will run until the user closes the program in this Project. Therefore, I have ensured the continuity and effective operation of the program. The program menu is displayed on the screen until the program is closed and the user is expected to choose an option. The loops I used made this operation faster and easier.

3) Development:

1-Plan:

First, I decided which classes to create. I planned which variables would be in these classes and which ones would be const. I planned the structure of the linked lists I will create for two different employee types, and whether I will use pointers or not. I have shaped the functions in my mind on how to access,add, change or delete nodes in these linked lists. In oreder to do these, I had to be well-versed in pointers, references, linked lists, classes and inheritance of classes.

That's why I researched these topics thoroughly, studied the sample codes and tried to understand the logic of the codes.

2-Analysis:

Since our program will run in a loop and we weill reach employees with nodes in linked lists instead of array indexes, I decided to use pointers mostly in my codes. In this way, I analyzed that I could Access the information in linked lists more effectively and cleanly. I also analyzed that my code does not occupy too much memory by making use of dynamic memory allocation examples and using them in my code.

3-Design and Implementation:

```
class Date {
public:
    Date(int day,int month,int year);
    const int getDay() const;
    const int getMonth() const;
    const int getYear() const;
    const int operator<(const Date& date2) const;
    friend ostream& operator<<(ostream& os, const Date& dt);

private:
    const int day;
    const int month;
    const int year;

};
```

Diagram 1: Date Class

First, I designed Date class for our program. It has basic class diagram. Its variables (day,month,year) are private and const.

Since the variables are private, I used getter methods. It has getDay (which returns day), getMonth (returns month) and getYear (returns year) methods. Also, I created the '<' opearator to compare 2 dates. Thus I got the information of which date is newer or older. I also created '<<' operator to print the date in text format. I printed the dates with a single operator instead of using the getter methods all the time.

```
class Employee {
public:
    Employee(int id,int type,string name,string surname,string title,double salary,Date birthD,Date appointmentD,int serviceLength);



    const int getId() const;
    const int getType() const;
    const string &getName() const;
    const string &getSurname() const;
    string getTitle();
    void setTitle(string title);
    double getSalary() const;
    void setSalary(double salary);
    const Date &getBirthD() const;
    const Date &getAppointmentD() const;
    const int getServiceLength() const;
    friend ostream& operator<<(ostream& os, const Employee& employee);

private:
    const int id;
    const int type;
    const string name;
    const string surname;
    string title;
    double salary;
    const Date birthD;
    const Date appointmentD;
    const int serviceLength;
```

Diagram 2: Employee Class

Then I created Employee class. It has many variables like ID, type,name, surname etc.(employee's infos). Its variables are private. Since the variables are private, I used getter and setter methods. Some variables are const (id,type,name,surname,birth date,appointment date,service length), beacuse these variables can't be changed, they must be constant but title or salary variable can be changed thus they are not const. I also created '<<' operator to print the employee's info in text format. I printed the employess with a single operator instead of using the getter methods all the time. (For example: cout<<employee.id<<" "<<employee.name<<" "…….)

I created TemporaryEmployee and PermanentEmployee classes to seperate employee types. Both of them inherit from the Employee class. The two have exactly the same struct and similar constructer. They are only used to seperate employee types.

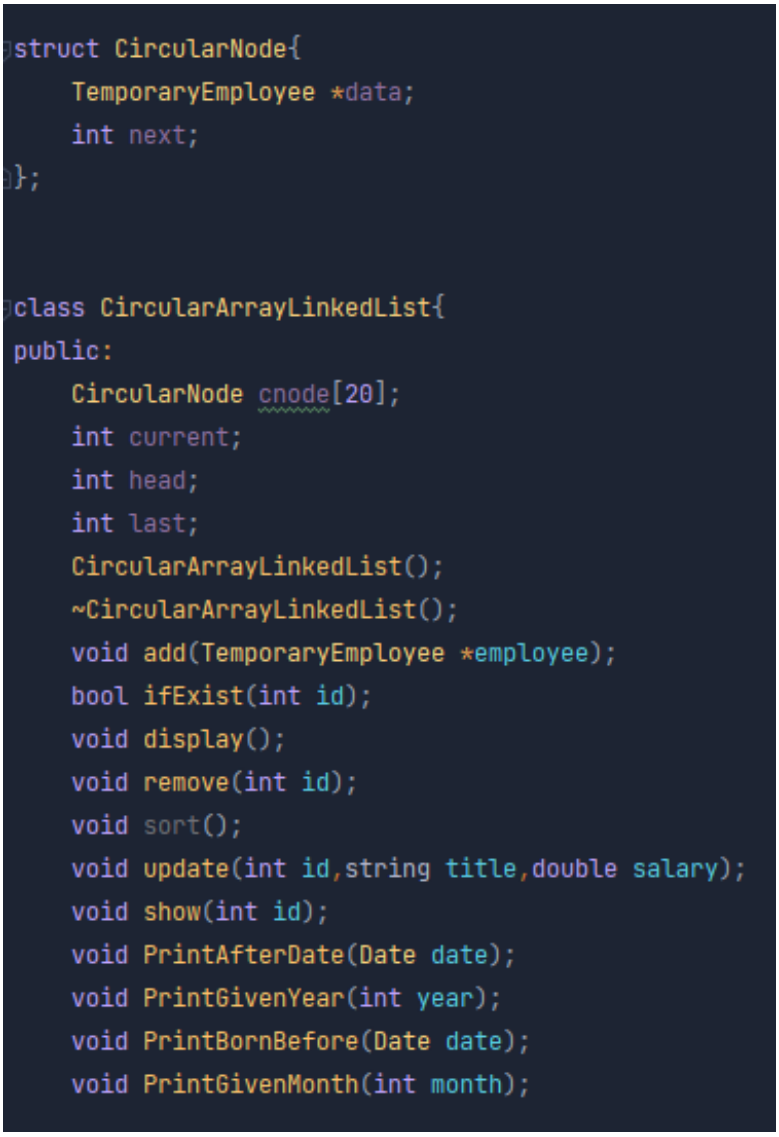
```
struct Node {
public:

    PermanentEmployee data;
    Node * next;
    Node * prev;

    Node(const PermanentEmployee *employee);

};

class DoubleDynamicLinkedList{
public:
    Node* head;
    DoubleDynamicLinkedList();
    ~DoubleDynamicLinkedList();
    Node* ifExist(int id);
    void addNode(Node* node);
    void removeNode(int id);
    void update(int id,string title,double salary);
    void addAfter(int id,Node* node);
    void show();
    PermanentEmployee search(int id);
    int searchID(int id);
    void PrintReverse();
    void PrintAfterDate(Date date);
    void PrintGivenYear(int year);
    void PrintBornBefore(Date date);
    void PrintGivenMonth(int month);
    void PrintGivenTitle(string title);
};
```

Diagram 3: DoubleDynamicLinkedList

I created DoubleDynamicLinkedList to store PermanentEmployee objects. I created Node struct which pointers to itself, thus I easily got data or node's prev/next node.

- 'ifExist' method checks if there is an employee with that id in the DoubleLinkedList with the id parameter it takes.
- 'addNode' method takes Node* parameter that keeps infos of employees and links this node to DoubleLinkedList.
- 'removeNode', takes parameter as id. And it removes the node whose employee's id is equals to id at parameter.
- 'update' method find the employee with id , and updates that employee's salary and title.
- 'show' method displays all of employees that in DoubleLinkedList.
- 'searchID', finds the employee with id.

Most of these methods were created with the logic of navigating between nodes with next and prev pointers.



```
struct CircularNode{
    TemporaryEmployee *data;
    int next;
};


class CircularArrayLinkedList{
public:
    CircularNode cnode[20];
    int current;
    int head;
    int last;
    CircularArrayLinkedList();
    ~CircularArrayLinkedList();
    void add(TemporaryEmployee *employee);
    bool ifExist(int id);
    void display();
    void remove(int id);
    void sort();
    void update(int id,string title,double salary);
    void show(int id);
    void PrintAfterDate(Date date);
    void PrintGivenYear(int year);
    void PrintBornBefore(Date date);
    void PrintGivenMonth(int month);
```

Diagram 4: CircularLinkedList class

I created CircularLinkedList to store TemporaryEmployee objects. I created Node struct named CircularNode which stores TemporaryEmployee pointer and next (which stores next index), thus I easily got data or node's next node. Also I created array with type of CircularNode.

-'ifExist' method checks if there is an employee with that id in the CircularLinkedList with the id parameter it takes.

-'add' method adds TemporaryEmployee into the CircularNode array.

-'display' method displays all of employees that in CircularLinkedList.

- 'remove', takes parameter as id. And it removes the node whose employee's id is equals to id at parameter.
- 'update' method find the employee with id , and updates that employee's salary and title.
- 'show' method shows the employee's infos.

Most of these methods are designed to reach the next value of the node so that we can navigate through the CircularLinkedList.

I think someof these two classes' methods are useful because I tried to write codes to work effectively and cleanly. But I think some of them are useless because I think I wrote some codes too complex.

```
int strToint(string int_x){...}

int* split(string str){...}
void enterInfo(int &type,string &name,string &surname,string &title,double &salary,string &birthD,string &appD){...}
```

Diagram 5: main.cpp methods

There are 3 methods in main.cpp file. 'strToint' method converts string to integer (for Date that taken from input). 'split' method splits the given string into parts according tp a certain specifier. 'enterInfo' method is used to get the information of the employee in the operations of adding.

These methods made the code cleancode and more readable. Also, these methods allowed to avoid unnecessary code redundancy.

4-Programmer Catalog:

I spent 1.5 days for analysis, 3 days for design and implementation, 1 day for testing and 2 hours for reporting.

Other developers can reuse my code's some part. Because some parts of my code are really clean and understandable. So, they can add new features or change features of those codes. But they can't reuse some of my code because some parts of the code are too complicated and time consuming to understand. I suggest other programmers to improve themselves on pointers, node and linked lists. Because in such projects, mastery of these subjects makes the codes more practical and faster to write. They do not have to deal with code complexity.

5-User Catalog:

These code can be used in the program created to seperate those who come to a concert as vip tickets and regular tickets and to store those who come to the concert.

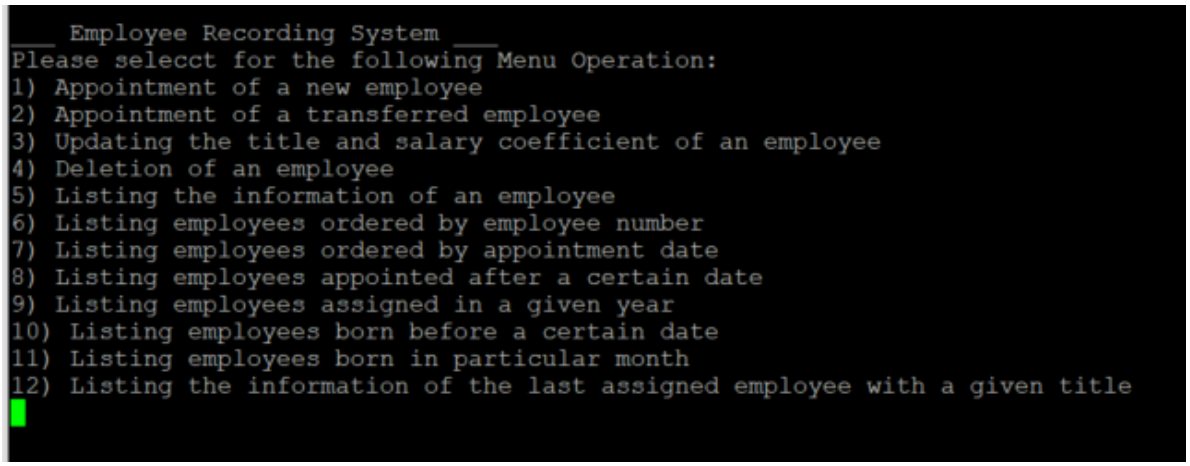


Diagram 6: Menu

- In the menu, there are 12 options. When the user selects an option, the selected option and the submenu of that option appear on the screen. If there is information requested from the user, the program prompts user to type that information. (Diagram 7)

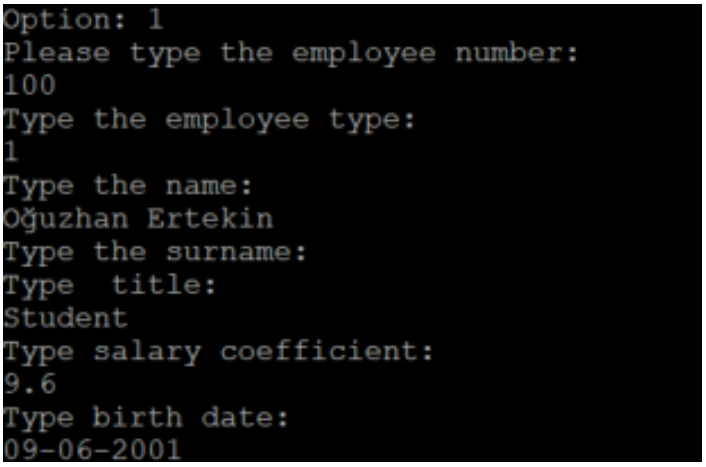


Diagram 7: Taking input information example

- If the user is asked to enter a date, the user should write the day,month and year seperated by “-“
  DAY-MONTH-YEAR → 09-06-2001

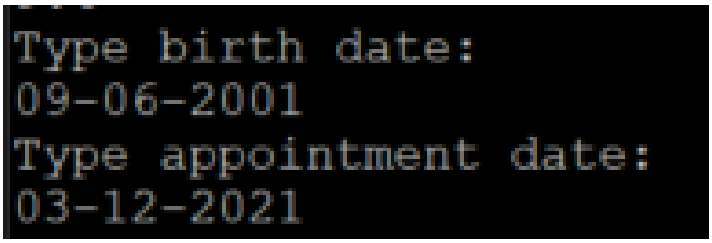
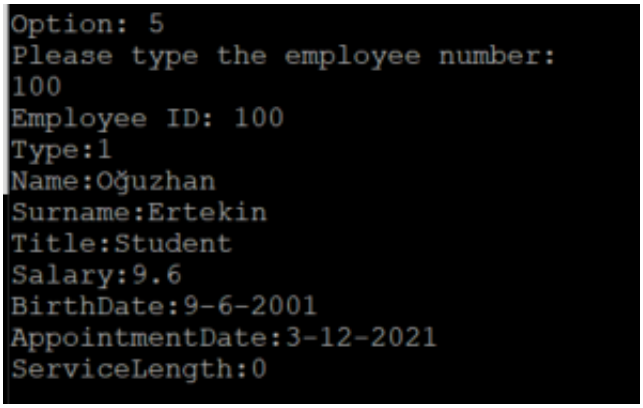
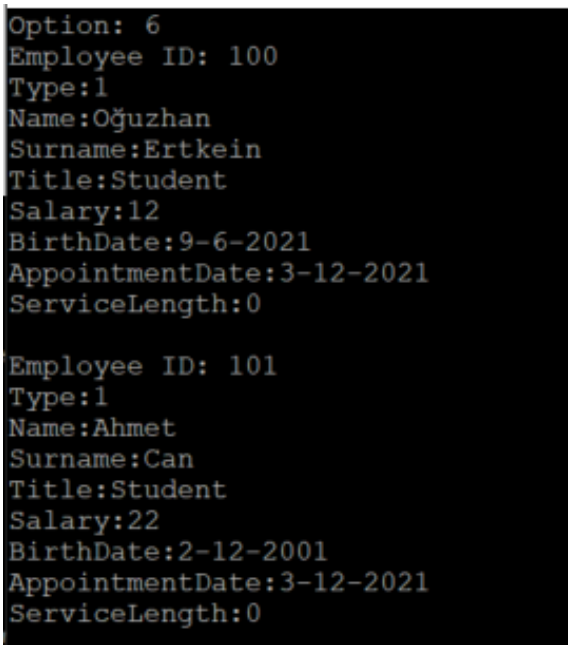


Diagram 8: Typing Date

- If the user wants to print the employee information, the employee information will appear in a row.





4)Results, Discussions and Conclusions:

I faced many errors while coding this program. Some of them are “Memory Error”,”NullPointer Error”,”Reference Error”. To solve these problems, I did a lot of searching and testing. Then I tried to reach the best result I could. I think the DoubleLinkedList codes I wrote properly and completely. Because most of the errors I got were from CircularLinkedList codes. Since I could not fully grasp the logic and sample of codes of the CircularLinkedList, I made mistakes in the codes all the time, and these mistakes challenged me. Therefor, I can’t write ‘sort’ and ‘remove’ methods. Hence, when printing employees in CircularLinkedList, I could not print them in order according to a certain value or remove them from the CircularLinkedList. So , sometimes user can’t see true output. After the Project, I understood the logic of CircularLinkedList but I realized that I was still stuck while writing the CircularLinkedList codes. I have to do more exercises for this. The errors I faced in DoubleLinkedList put me in a better position in terms of nodes, pointers and linkedlists. At least from now on I think I can create a simple LinkedList with self-pointing nodes.
Taking all these into account, my self-rated score fort his assignment is in the 80-85 range.

5)References:

1-BBM 201 Lecture Slides

2-www.geeksforgeeks.org

3-www.cplusplus.com

4-www.stackoverflow.com

5-www.learncpp.com

6- www.w3schools.com

7- www.javatpoint.com

8-www.tutorialspoint.com

9-www.codecademy.com

10-www.programiz.com