Hacettepe University

Computer Engineering Department

BBM203 Software Laboratory I - 2021 Fall

# Assignment 2
# Linked Lists

Date: 03.12.2021

Student Name:
Melike Nur DULKADİR

Student Number:
b21992919

## 1. INTRODUCTION

In this assignment, we were expected to create an "Employee Recording System" for an institution. Each employee had to be kept in the system with different features such as employee number, employee type, name. Then, 12 different types of operations in the console part had to be done with the employees in the system.

## 2. METHOD

I first started by creating the different classes that I will use in this project. Then I created the attributes, getter setters and necessary constructors that I am expected to write in these classes. classes CircularArrayLinkedList, Date,Node, DynamicDoubleLinkedList, Employee, PermanentEmploye, TemporaryEmployee. The PermanentEmploye,TemporaryEmployee classes extend the Employee class.

After creating all this, I started creating a circular single-linked list structure to hold Temporary Employees. I created the node class and set up the node structure that I will create to hold the employee values and pointer. Then, I created isExist, findEmployee, findAvail, insert, deleteTemployee, update function in theCircularArrayLinkedList class in order to perform the operations that may be requested from us in the console section. The structure I created in the CircularArrayLinkedList class is actually a structure formed by keeping the linkedlist structure in the array. Since we were given the number of employees that could be TemporaryEmployee type and it was fixed, I created an array with 20 elements. However, in this array, unlike the normal array structure, I connected the array elements to each other with the next pointer according to the employee number. Since it is in a circular structure, I set the last element to show the first element. So even if my arraysize is 20, if there are two elements in the array and they are in different parts of the array, they can reach each other with the linked list structure.

I created another structure, the Double Linked List structure, with the Dynamic Memory Allocation Implementation Type within the DoubleDynamicLinkedList class. Here, I created the findEmployee, insert, update, remove functions that will do the necessary operations. I stated that the nodes that I will keep in the Double Linked List structure will be created with the struct node structure this time and the data will be PermanenetEmployee, and the prev and next parts will be an int value. Unlike array, it was a great advantage that a new Permanent employee could be added here without any restrictions. And each added employee, if any, is linked to the previous and next ones with the next and prev pointers according to the appointment date.

# 3. DEVELOPMENT

## a) Plan

First of all, I planned to create the employee types that will be kept in the system and then create the data structures where I will store these employee objects. Then I planned to define the relevant functions of this data structure and create additional functional features in the system. Finally, I was going to design the interface part of the system and test it.

## b) Analysis

I analyzed the data structures I created. Since we will create our own linked list structures, I examined the working logic and features of these structures. Then I tried to organize these structures in such a way that they could store my own data types instead of the usual data types. I created two different types of linked lists using two different infrastructures.

## c) Design

The methods I use are isExist, findSuitable, insert, remove, update . The isExist method checks whether the Employee type that will be newly added to our data structure has already been added. FindSuitable is responsible for finding the appropriate place in our data structure for the node to be added. Insert, remove and update functions also perform the classical operations in the same way. Since our isExist and findSuitable methods are pre-prepared, we have provided the necessary controls and requirements as we use them together with the Insert method when adding to the list. Thus, I made the code simplification by turning the operations to be done before the addition into functions.

We kept certain data in certain structures by performing classic linkedlist operations. In the meantime, we used two different infrastructures. The implementation of DoublyLinkedList, which uses dynamic allocation instead of array infrastructure, was easier. Because we are not limited to the array structure and we can keep the next, prev and value data in a single struct node. Thus, it was easier to navigate between the nodes and reach the desired one.

## d) Implementation

First of all, I created 2 different types of linked lists to hold temporaryEmployee and permanentEmployees. Then, the program flow was determined according to the user's choices. There was a number on the interface for each selection, and after these numbers were written to the console, the inputs were read. First of all, employee number is requested in all options and it is checked whether it is in the lists.

Afterwards, if the user selects 1 or 2, a new employee is added to the institution and this insertion process is carried out with the insert method of the data structures themselves. The difference between 1 and 2 is that the employee in 2 is moving from another institution to our own institution.

In option 3, the title and salarycoefficient information of the employee to be updated is taken. Then the update process is completed with the update function.

If it is 4, the relevant employee is removed from its own list and this is done with the remove functions.

In number 5, an employee's information is listed and all information of the employee is suppressed with the cout operator (<<), which we overload with the findEmployee function.

In the option number 6, the employees in the two lists were sorted according to their employee numbers.

In number 7, this sorting was performed according to dateOfAppointment.

In number 8, the employees assigned to the institution after the date the user sent it were listed.

In the option number 9, employees assigned in the year given by the user, are listed.

All employees born before the date posted in transaction number 10 are listed.

In operation number 11, all employees born in the sent month are listed.

In our last operation number 12, the last employee assigned to the institution in the given title is listed.

## e) Programmer Catalog

I spent 1 day for the analysis and it took an extra day to organize a design according to the results I got from the analysis. After these, the implementation part was completed in 3 days. Tests and debugging were done in 4 days after these steps. The report was completed after a 2-day study.

If there are two different employee types among the requirements of other developers, this solution can be considered. In a company where there are two types of employees in two different structures, this project can be used with the development of a generic structure and interface (UI) in addition to this implementation.

Our abstract class itself is the Employee class. TemporaryEmployee and PermanentEmployee classes are inherited from this class. In this way, common methods are not rewritten. You can see the base class implementation in abstract form below.

In addition to the members, there is a method that compares according to the appointment dates and << operator overload that works like the toString method.

```
class Employee {
private:
    int employee_number = 0;
    int employee_type; // 0 -> temp, 1 -> perm
    string name;
    string surname;
    string title;
    float salary_coefficient;
    Date date_of_birth;
    Date date_of_appointment;
    int length_of_service = 0;

public:
    Employee(int employeeNumber, int employeeType, string &name, string &surname,
             string &title, float salaryCoefficient, Date &dateOfBirth, Date &dateOfAppointment)

    Employee(int employeeNumber, int employeeType, string &name, string &surname,
             string &title, float salaryCoefficient, Date &dateOfBirth, Date &dateOfAppointment,
             int lengthOfService)
    Employee() {}
public:
int getEmployeeNumber()
void setEmployeeNumber(int employeeNumber)
int getEmployeeType()
void setEmployeeType(int employeeType)
string getName()
void setName(const string &name)
string getSurname()
void setSurname(const string &surname)
string getTitle()
void setTitle(const string &title)
float getSalaryCoefficient()
void setSalaryCoefficient(float salaryCoefficient)
Date getDateOfBirth()
void setDateOfBirth(const Date &dateOfBirth)
Date getDateOfAppointment()
void setDateOfAppointment(const Date &dateOfAppointment)
int getLengthOfService()
void setLengthOfService(int lengthOfService)
bool compareByAppointment(Employee* emp1, Employee* emp2)
    {
        return emp1->getDateOfAppointment() < emp2->getDateOfAppointment();
    }
};

    friend ostream& operator<<(ostream &output,const Employee& employee);
ostream& operator<<(ostream &output,const Employee& employee) {
           output << "Employee number: " << employee.getEmployeeNumber() <<
endl
           <<"Employee type: " << employee.getEmployeeType() << endl
           <<"Employee name: " << employee.getName() << endl
           <<"Employee surname: " << employee.getSurname() << endl
           <<"Employee title: " << employee.getTitle() << endl
           <<"Employee salary coefficient:
"<<employee.getSalaryCoefficient()<< endl
```

```
           <<"Employee date of birth: " << employee.getDateOfBirth() << endl
           <<"Employee date of appointment: " <<
employee.getDateOfAppointment() << endl
           <<"Employee length of service: " << employee.getLengthOfService()
<< endl;
    return output;
}
```

Child classes TemporaryEmployee and PermanentEmployee override constructors by extending this class. Now let's look at the data structures we created ourselves.

The Node class is created to hold temporaryEmployees. Thus, thanks to the objects to be created from the node class, we can use our employees in our array implementation. We keep a temporaryEmployee object and its next value in a node object. It includes get and set methods.

```
class Node {
private:
    TemporaryEmployee value;
    int next;

public:

    //initializer list constructor
    Node(TemporaryEmployee value, const int& next) : value(value) ,
next(next) {}
    //default
    Node() = default;


    //getters & setters
    TemporaryEmployee getValue(){
        return this->value;
    }

    int getNext(){
        return this->next;
    }

    void setNext(const unsigned& next){
        this->next = next;
    }

};
```

isExist, findAvailable and insert,remove,update operations we mentioned above are included in our linked list classes. We have also summarized the ways of working in the above sections.

Our data structure CircularSinglyLinkedList where we keep the node objects is as follows:

```
const int SIZE = 20;


class CircularArrayLinkedList{

private:
    int avail;
    int head;
    int tail;
    Node list[SIZE];

public:

    CircularArrayLinkedList() {
        this->avail = 0;
        this->head = 0;
        this->tail = 0;
        for(int i = 0; i != SIZE-1; ++i){
            this->list[i].setNext(-1);
        }
    }


    bool isExist(int empNum)

    TemporaryEmployee findEmployee(int empNum)

    int findAvail()

    bool insert(TemporaryEmployee temp)

    int findSuitable(TemporaryEmployee temp)

    void deleteTemployee(TemporaryEmployee temp)

    void update(TemporaryEmployee temp,string title,float salary_coef)
 };
```

The nodes where PermanentEmployees will be found are in struct structure, unlike the previous one. The CircularDoublyLinkedList structure where we keep the PermanentEmployees is as follows:

```
class DynamicDoubleLinkedList {
public:
    DynamicDoubleLinkedList() {}


// A doubly linked list node
    struct Node {
        PermanentEmployee data;
        struct Node *next;
        struct Node *prev;
    };

    struct Node *head = nullptr;
    struct Node* last = nullptr;

    bool isExist(int empNum)

    PermanentEmployee findEmployee(int empNum)

    void insert(PermanentEmployee permEmp)

    struct Node* findSuit(PermanentEmployee permEmp,struct Node* node)

    bool update(PermanentEmployee permEmp,string title,float salary_coef)

    void remove(PermanentEmployee permEmp)

};
```

Our Date class has information to keep the historical attributes in the employee. It is then used with the composition structure in Employee. Also, comparison operators are overloaded in this file. We do parse and split operations in the constructor as can be seen. Its structure was as follows:

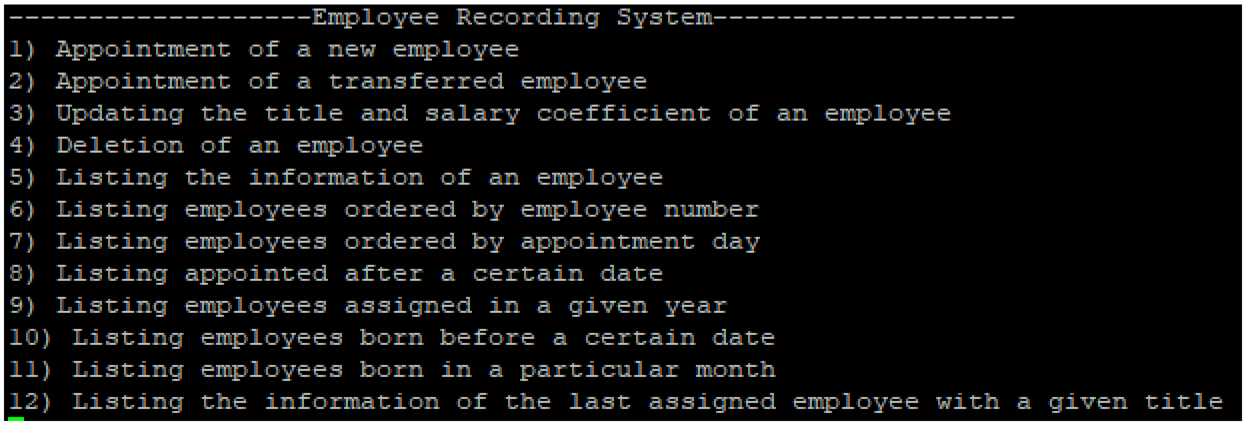
```
class Date {
    public:
        Date(string& date){
    istringstream ss(date);
    string token;
    int count  = 0;
    while(std::getline(ss, token, '-')) {
        if(count==0){
            day = stoi(token);
            count++;
        }else if(count == 1){
            month = stoi(token);
            count++;
        } else year = stoi(token);
    }

}

Date() {}

private:
        int day;
        int month;
        int year;
    public:
          void setDay(int day) {
              day = day;

}

int getDay() const {
    return day;
}

void setMonth(int month) {
    month= month;
}

int getMonth() const {
    return month;
}

void setYear(int year) {
    year  = year;
}
int getYear() const {
    return year;
}


     friend ostream& operator<<(ostream& output, Date date);
};

inline ostream &operator<<( ostream &output, Date date){
    output <<  date.getDay() <<"-"<< date.getMonth() << "-"<< date.getYear();
    return output;
}
inline bool operator>(Date a, Date b)

inline bool operator<(Date a, Date b)              }

inline bool operator<=(Date a, Date b)

inline bool operator>=(Date a, Date b)

inline bool operator==(Date a, Date b)
```

## f) User Catalog

The system provides storage, comparison and listing opportunities for entities of different types or roles. If these entities do not have distinctive features, it is not recommended to use the system.

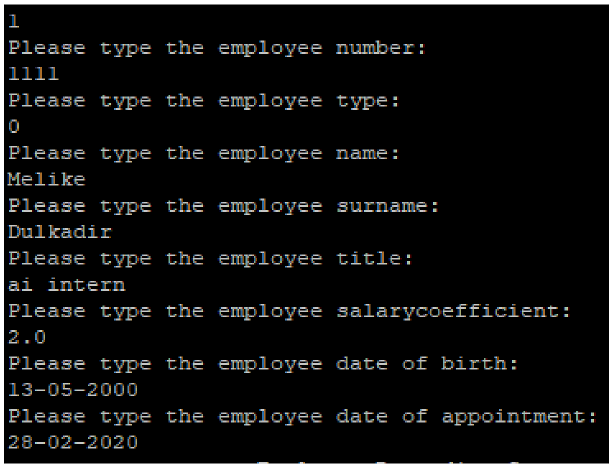
The current operation of the system is as in the images below:

The login screen is like this and the user is expected to make the appropriate selection.



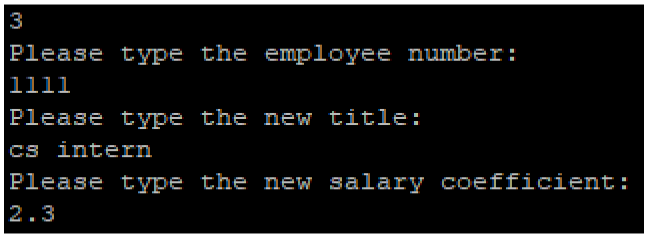
```
-------------------Employee Recording System-------------------
1) Appointment of a new employee
2) Appointment of a transferred employee
3) Updating the title and salary coefficient of an employee
4) Deletion of an employee
5) Listing the information of an employee
6) Listing employees ordered by employee number
7) Listing employees ordered by appointment day
8) Listing appointed after a certain date
9) Listing employees assigned in a given year
10) Listing employees born before a certain date
11) Listing employees born in a particular month
12) Listing the information of the last assigned employee with a given title
```

The steps to add a new employee in option 1 are as follows:



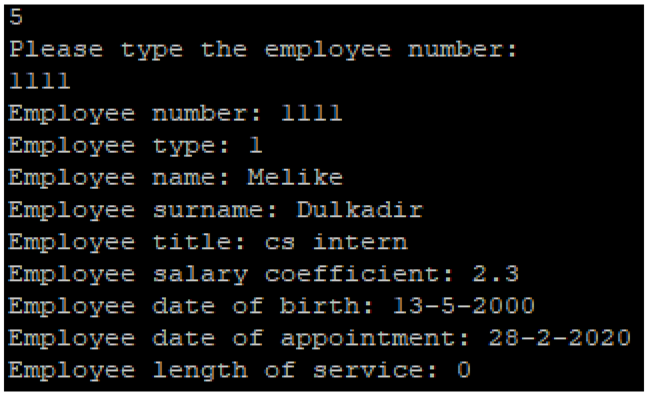
```
1
Please type the employee number:
1111
Please type the employee type:
0
Please type the employee name:
Melike
Please type the employee surname:
Dulkadir
Please type the employee title:
ai intern
Please type the employee salarycoefficient:
2.0
Please type the employee date of birth:
13-05-2000
Please type the employee date of appointment:
28-02-2020
```

The steps to update an employee in option 3 are as follows:



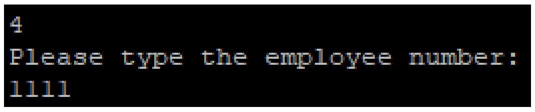
```
3
Please type the employee number:
1111
Please type the new title:
cs intern
Please type the new salary coefficient:
2.3
```

Updated employee information:



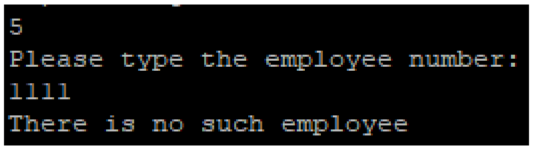
```
5
Please type the employee number:
1111
Employee number: 1111
Employee type: 1
Employee name: Melike
Employee surname: Dulkadir
Employee title: cs intern
Employee salary coefficient: 2.3
Employee date of birth: 13-5-2000
Employee date of appointment: 28-2-2020
Employee length of service: 0
```

The steps to delete an employee in option 4 are as follows:



```
4
Please type the employee number:
1111
```

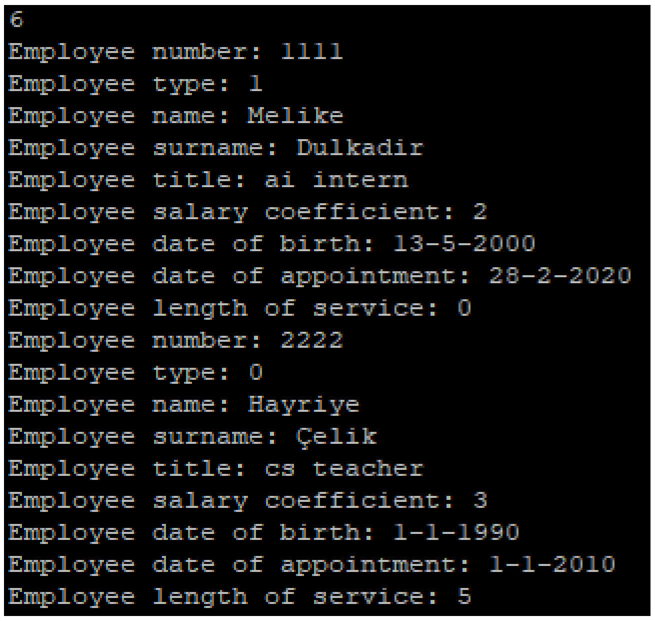
Deleted employee couldn't be found:



```
5
Please type the employee number:
1111
There is no such employee
```

Steps of operation 5 are already shown on top.

The steps to list all employees according to their employeeNumber in option 6 are as follows:



```
6
Employee number: 1111
Employee type: 1
Employee name: Melike
Employee surname: Dulkadir
Employee title: ai intern
Employee salary coefficient: 2
Employee date of birth: 13-5-2000
Employee date of appointment: 28-2-2020
Employee length of service: 0
Employee number: 2222
Employee type: 0
Employee name: Hayriye
Employee surname: Çelik
Employee title: cs teacher
Employee salary coefficient: 3
Employee date of birth: 1-1-1990
Employee date of appointment: 1-1-2010
Employee length of service: 5
```

The other methods also work as the given examples.

## 4. RESULTS

As a result, we have developed a system where we can keep multiple types of employees separately across multiple data structures. In this system, we created linked lists using two different implementations and performed their operations. In the last part, we tested it and saw it working. It has been seen that using dynamic allocation during the development phase is much more convenient and easy. In addition, it was very difficult to run the application on different systems due to the compiler difference. We got to know the basic linked list structure in the end.

## 5. References

https://www.tutorialspoint.com/data_structures_algorithms/circular_linked_list_algorithm.htm

https://www.geeksforgeeks.org/doubly-linked-list/

and the BBM201 Course slides.