Data Structure Final Project

By: Hansel Faren (2501990350)

Kevin Matthew Tanuwijaya (2502036514)

Nathaniel Davide Darmawan (2502087830)

L2BC / Computer Science

Problems Description

As for a startup company or some people that are planning to start a business, they need some preparation to start their business. One of them is when they have some employees or staff, they will need some program to store the data of the employee or staff that is working for their business. As for the solution, we came up with a program where they can store their employees or staff's information.

Choice of Data Structure

The data structure that we use is vector. Why we used vector is because vector is a one-dimensional data structure and easier access to the memory. Even though it consumes more memory, vector can manage the memory and also can extend dynamically.

Alternative Data Structure

Other data structures that can be used in our program would be arrays or linked lists. Array consumes a small amount of memory because its memory cannot extend dynamically where it's not suited for our program. On the other hand, linked lists are the best when it comes to insertion and deletion elements in the middle but it is not the best when you want to access a known index.

Program Manual

This database program uses a command line interface. A menu will appear when you start the program, and it will prompt you with 5 actions.

```
============================EMPLOYEE DATABASE============================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action:
```

1.) Here the program prints every value in the database in an set order and you will be able to view your database.

```
class Database{
    //all variables
    int id;
    string name;
    string address;
    string BirthDate;
    string PhoneNumber;
    string position;

    int HoursWorked;
    int HoursOvertime;

    int RateNormal;
    int RateOvertime;
    int salary;

    //making vectors
    public:
    vector<int> EmpID;

    vector<string> EmpName;
    vector<string> EmpAddress;
    vector<string> EmpBirthDate;
    vector<string> EmpPhoneNumber;
    vector<string> EmpPosition;

    vector<int> EmpHoursWorked;
    vector<int> EmpHoursOvertime;

    vector<int> EmpRateNormal;
    vector<int> EmpRateOvertime;
    vector<int> EmpSalary;
```

2.) This is the setters of the value

```
    //all setters
    void set_id(int x){
        id=x;
    }

    void set_name(string x){
        name=x;
    }

    void set_address(string x){
        address=x;
    }

    void set_BDate(string x){
        BirthDate=x;
    }

    void set_number(string x){
        PhoneNumber=x;
    }

    void set_position(string x){
        position=x;
    }
```

```
    void set_Hours(int x){
        HoursWorked=x;
    }

    void set_overtime(int x){
        HoursOvertime=x;
    }

    void set_rate(int x){
        RateNormal=x;
    }

    void set_rateOver(int x){
        RateOvertime=x;
    }

    void set_salary(int x){
        salary=x;
    }
```

3.) This class to run some function that is featured in our program

```
    //all the getters made will be used to input the info there (exclude hours and salary)
    //note: we can force them to make an if statement calling all setters
    void employee_info_store(int id, string name, string address, string BirthDate, string PhoneNumber, string position){
        EmpID.push_back(id);
        EmpName.push_back(name);
        EmpAddress.push_back(address);
        EmpBirthDate.push_back(BirthDate);
        EmpPhoneNumber.push_back(PhoneNumber);
        EmpPosition.push_back(position);
        EmpSalary.push_back(0);
    }

    //check their total hours worked
    void Total_hours(int HRS, int HRSO){
        EmpHoursWorked.push_back(HRS);
        EmpHoursOvertime.push_back(HRSO);
    }

    //calculate their salary, maybe an if statement if we have time
    void Calculate_Salary(int x){
        x-=1;
        int NormalHr=EmpHoursWorked[x];
        int OvertimeHr=EmpHoursOvertime[x];
        int total= NormalHr*RateNormal+OvertimeHr*NormalHr;
        EmpSalary[x]=total;

    }
```

4.) This class is for showing that the database is updated and for removing value

```
    void updatedMessage(){
        cout << "=========================Database successfully updated!=========================";
    }

    //a way to remove employees by their index (not sorting proof)
    void remove_employee(int x){
        x-=1;
        EmpName.erase(EmpName.begin()-1+x);
        EmpAddress.erase(EmpAddress.begin()-1+x);
        EmpBirthDate.erase(EmpBirthDate.begin()-1+x);
        EmpPhoneNumber.erase(EmpPosition.begin()-1+x);
        EmpPosition.erase(EmpPosition.begin()-1+x);

        EmpHoursWorked.erase(EmpHoursWorked.begin()-1+x);
        EmpHoursOvertime.erase(EmpHoursOvertime.begin()-1+x);

        EmpRateNormal.erase(EmpRateNormal.begin()-1+x);
        EmpRateOvertime.erase(EmpRateOvertime.begin()-1+x);
        EmpSalary.erase(EmpSalary.begin()-1+x);
    }

};
```

5.) Starts the driver file with declaring variable

```cpp
/*
Driver for the Command Line Interface
*/
#include <iostream>
#include <vector>
#include <./Database.hpp>

int main()
{
    int id;
    string name;
    string address;
    string BirthDate;
    string PhoneNumber;
    string position;

    int HoursWorked;
    int HoursOvertime;

    int RateNormal;
    int RateOvertime;
    int salary;

    Database EMPLOYEE;
    int userInput = 0;
    int userInput2 = 0;
    int i = -1;
```

6.) While function to make sure the program runs endlessly except choose the terminate function

```cpp
    while (userInput != 5){
        cout<<"\n=========================EMPLOYEE DATABASE=========================";
        cout<<"\n1. View Database\n2. Insert new employee\n3. Edit employee data\n4. Delete employee data\n5. Close";
        cout<<"\n\nInput an action: ";
        cin >> userInput;

        switch (userInput){
            case 1:
                cout << "\n INDEX//ID//NAME//ADDRESS//BIRTH DATE//PHONE NUMBER//POSITION//HOURS WORKED//HOURS WORKED OVERTIME//NORMAL RATE//OVERTIME RATE\n";
                for(int x = 0; x < EMPLOYEE.EmpID.size(); x++){
                    cout<<x << ". // "<<EMPLOYEE.EmpID[x] << "//" <<EMPLOYEE.EmpName[x] <<"//"<<EMPLOYEE.EmpAddress[x]<< "//" <<EMPLOYEE.EmpBirthDate[x]<<"//"<< EMPLOYEE.EmpPh
                }
                break;

            case 2:


                cout << "\nInput ID: ";
                cin >> id;
                    while (id<0){
                        cout << "Value must be a positive integer: ";
                        cin >> id;
                    }
                EMPLOYEE.EmpID.push_back(id);

                cout << "Input Name: ";
                cin >> name;
                EMPLOYEE.EmpName.push_back(name);

                cout << "Input Address: ";
                cin >> address;
                EMPLOYEE.EmpAddress.push_back(address);
```

```
                cout << "Input Birth Date: ";
                cin >> BirthDate;
                EMPLOYEE.EmpBirthDate.push_back(BirthDate);

                cout << "Input Phone Number: ";
                cin >> PhoneNumber;
                EMPLOYEE.EmpPhoneNumber.push_back(PhoneNumber);

                cout << "Input Position: ";
                cin >> position;
                EMPLOYEE.EmpPosition.push_back(position);

                cout << "Input Hours Worked: ";
                cin >> HoursWorked;
                while (HoursWorked<0){
                    cout << "Value must be a positive integer: ";
                    cin >> HoursWorked;
                }
                EMPLOYEE.EmpHoursWorked.push_back(HoursWorked);

                cout << "Input Hours Worked Overtime: ";
                cin >> HoursOvertime;
                while (HoursOvertime<0){
                    cout << "Value must be a positive integer: ";
                    cin >> HoursOvertime;
                }
                EMPLOYEE.EmpHoursOvertime.push_back(HoursOvertime);
```

```
                cout << "Input Normal Rate: ";
                cin >> RateNormal;
                while (RateNormal<0){
                    cout << "Value must be a positive integer: ";
                    cin >> RateNormal;
                }
                EMPLOYEE.EmpRateNormal.push_back(RateNormal);

                cout << "Input Overtime Rate: ";
                cin >> RateOvertime;
                while (RateOvertime<0){
                    cout << "Value must be a positive integer: ";
                    cin >> RateOvertime;
                }
                EMPLOYEE.EmpRateOvertime.push_back(RateOvertime);

                EMPLOYEE.updatedMessage();
                break;
```

```
            case 3:
                if(EMPLOYEE.EmpID.size() == 0){
                    cout << "The database is still empty!";
                    break;
                }

                i = -1;
                userInput2 = 1;
                cout << "Enter the employee index of the employee you want to edit: ";
                cin >> i;
                if (i<0 || i>EMPLOYEE.EmpID.size()){
                    while(i<0 || i>EMPLOYEE.EmpID.size()){
                        cout << "Index must be between 0 and the number of employees: ";
                        cin >> i;
                    }
                }

                while  (userInput2 != 0){
                    cout << "\n Which element would you like to change?";
                    cout << "\n0. Back\n1. ID\n2.Name\n3. Address\n4. DOB\n5. Phone Number\n6. Position\n7.Hours Worked\n8. Overtime Hours\n9. Normal Rate\n10. Overtime Rate";
                    cout << "\nEnter number: ";
                    cin >> userInput2;
```

```
                case 8:
                    cout << "Enter new hours worked overtime: ";
                    cin >> HoursOvertime;
                    while (HoursOvertime<0){
                        cout << "Value must be a positive integer: ";
                        cin >> HoursOvertime;
                    }
                    EMPLOYEE.EmpHoursOvertime[i]=HoursOvertime;
                    EMPLOYEE.updatedMessage();

                    break;

                case 9:
                    cout << "Enter new normal rate: ";
                    cin >> RateNormal;
                    while (RateNormal<0){
                        cout << "Value must be a positive integer: ";
                        cin >> RateNormal;
                    }
                    EMPLOYEE.EmpRateNormal[i]=RateNormal;
                    EMPLOYEE.updatedMessage();

                    break;
```

```

            switch (userInput2){
                case 0:
                    //cancel edit
                    cout << "---Cancelling edit---";
                    break;

                case 1:
                    cout << "Enter new ID: ";
                    cin >> id;
                    while (id<0){
                        cout << "Value must be a positive integer: ";
                        cin >> id;
                    }
                    EMPLOYEE.EmpID[i] = id;
                    EMPLOYEE.updatedMessage();

                    break;

                case 2:
                    cout << "Enter new name: ";
                    cin >> name;
                    EMPLOYEE.EmpName[i] = name;
                    EMPLOYEE.updatedMessage();

                    break;
```

```
                case 3:
                    cout << "Enter new address: ";
                    cin >> address;
                    EMPLOYEE.EmpAddress[i] = address;
                    EMPLOYEE.updatedMessage();

                    break;

                case 4:
                    cout << "Enter new DOB: ";
                    cin >> BirthDate;
                    EMPLOYEE.EmpBirthDate[i] = BirthDate;
                    EMPLOYEE.updatedMessage();

                    break;

                case 5:
                    cout << "Enter new phone number: ";
                    cin >> PhoneNumber;
                    EMPLOYEE.EmpPhoneNumber[i] = PhoneNumber;
                    EMPLOYEE.updatedMessage();

                    break;
```

```
                    case 6:
                        cout << "Enter new position: ";
                        cin >> position;
                        EMPLOYEE.EmpPosition[i] = position;
                        EMPLOYEE.updatedMessage();

                        break;

                    case 7:
                        cout << "Enter new hours worked: ";
                        cin >> HoursWorked;
                        while (HoursWorked<0){
                            cout << "Value must be a positive integer: ";
                            cin >> HoursWorked;
                        }
                        EMPLOYEE.EmpHoursWorked[i]=HoursWorked;
                        EMPLOYEE.updatedMessage();

                        break;
```

```
                    case 10:
                        cout << "Enter new overtime rate: ";
                        cin >> RateOvertime;
                        while (RateOvertime<0){
                            cout << "Value must be a positive integer: ";
                            cin >> RateOvertime;
                        }
                        EMPLOYEE.EmpRateOvertime[i]=RateOvertime;
                        EMPLOYEE.updatedMessage();

                        break;

                    }
                }
                break;
```

```
            case 4:
                if(EMPLOYEE.EmpID.size() == 0){
                    cout << "The database is still empty!";
                    break;
                }

                int idtemp = -1;
                userInput2 = 0;
                cout << "Enter the employee index of the employee you want deleted: ";
                cin >> idtemp;

                int checkID=0;
                bool Checked=false;
                int empIndex;
                //making sure the id input is there
                for(i=0;i<EMPLOYEE.EmpID.size();i++){
                    checkID=EMPLOYEE.EmpID[i];
                    if(idtemp==checkID){
                        Checked=true;
                        empIndex=i;
                        break;
                    }else{
                        continue;
                    }
                }
```

```
                if (Checked==true){

                    while (userInput2 != (1 || 2)){
                        cout << "Are you sure you want to delete Index " << i;
                        cout << "\n1. Yes\n2. No" << endl;
                        cout << "Choice: ";
                        cin >> userInput2;
                    }
                    switch (userInput2){
                        case 1:
                            EMPLOYEE.EmpID.erase(EMPLOYEE.EmpID.begin()+empIndex);
                            EMPLOYEE.EmpName.erase(EMPLOYEE.EmpName.begin()+empIndex);
                            EMPLOYEE.EmpAddress.erase(EMPLOYEE.EmpAddress.begin()+empIndex);
                            EMPLOYEE.EmpBirthDate.erase(EMPLOYEE.EmpBirthDate.begin()+empIndex);
                            EMPLOYEE.EmpPhoneNumber.erase(EMPLOYEE.EmpPhoneNumber.begin()+empIndex);
                            EMPLOYEE.EmpPosition.erase(EMPLOYEE.EmpPosition.begin()+empIndex);

                            EMPLOYEE.EmpHoursWorked.erase(EMPLOYEE.EmpHoursWorked.begin()+empIndex);
                            EMPLOYEE.EmpHoursOvertime.erase(EMPLOYEE.EmpHoursOvertime.begin()+empIndex);

                            EMPLOYEE.EmpRateNormal.erase(EMPLOYEE.EmpRateNormal.begin()+empIndex);
                            EMPLOYEE.EmpRateOvertime.erase(EMPLOYEE.EmpRateOvertime.begin()+empIndex);
                            break;

                        case 2:
                            break;
                    }
```

```
                } else if (Checked==false){
                    cout << "employee does not exist"<<endl;
                }


                break;


            case 5:
            //breaks switch case; closes program
                break;
            }
        }


    return 0;

}
```

Results

```
C:\Users\Asus\Documents\L2BC\Data Structures\with Ms Nurul\driver.exe

==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 2

Input ID: 21
Input Name: hansel
Input Address: binus
Input Birth Date: 12
Input Phone Number: 0812
Input Position: manager
Input Hours Worked: 12
Input Hours Worked Overtime: 2
Input Normal Rate: 10
Input Overtime Rate: 2
========================Database successfully updated!==========================
==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 1

 ID//NAME//ADDRESS//BIRTH DATE//PHONE NUMBER//POSITION//HOURS WORKED//HOURS WORKED OVERTIME//NORMAL RATE//OVERTIME RATE
21//hansel//binus//12//0812//manager//12//2//10//2

==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 4
Enter the employee index of the employee you want deleted: 0
Are you sure you want to delete Index 0
1. Yes
2. No
Choice: 1
```

```
C:\Users\Asus\Documents\L2BC\Data Structures\with Ms Nurul\driver.exe
Input Normal Rate: 10
Input Overtime Rate: 2
========================Database successfully updated!==========================
==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 1

 ID//NAME//ADDRESS//BIRTH DATE//PHONE NUMBER//POSITION//HOURS WORKED//HOURS WORKED OVERTIME//NORMAL RATE//OVERTIME RATE
21//hansel//binus//12//0812//manager//12//2//10//2

==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 4
Enter the employee index of the employee you want deleted: 0
Are you sure you want to delete Index 0
1. Yes
2. No
Choice: 1

==========================EMPLOYEE DATABASE==========================
1. View Database
2. Insert new employee
3. Edit employee data
4. Delete employee data
5. Close

Input an action: 5

--------------------------------
Process exited after 35.32 seconds with return value 0
Press any key to continue . . .
```

Answer for question from the day of presentation

1. Delete by ID

   Added to the file

2. If a lot of employees are deleted by half, what happens to the memory ?

   The memory of the vector cannot be decreased but the memory space that has been used, will be reused for the future elements so that it can be inserted efficiently