PES UNIVERSITY

SUMMER COURSE

B.TECH. (CSE)

IV SEMESTER

Object Oriented Programming through C++

PROJECT REPORT

ON

Automated Performance Appraisal System

SUBMITTED BY :

NAME SRN

SPOORTHI SHIVAPRASAD PES2UG21CS536

SMRITI S KASHYAP PES2UG21CS529

JULY (summer) - 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ELECTRONIC CITY CAMPUS,

BENGALURU – 560100, KARNATAKA, INDIA

TABLE OF CONTENTS

Introduction

The Automated Performance Appraisal System is a software application developed using object-oriented programming concepts in C++. It aims to streamline and automate the process of evaluating and assessing employee performance within an organization.
The system provides functionalities such as adding employees, displaying the employee list, and enabling search operations. It also incorporates the role of managers, who are responsible for assigning ratings to employees based on their performance. The ratings provided by managers cannot be modified by other employees, ensuring data integrity and accuracy.

Methodology:
The system utilizes key concepts of object-oriented programming, including classes, inheritance, constructors, access specifiers, vectors, pointers, member functions, range-based for loops, and function overriding. It employs two main classes: the Employee class as the base class and the Manager class as the derived class. The Employee class encapsulates employee details such as name, ID, and rating. The Manager class extends the functionality of the Employee class, introducing additional features specific to managers, such as the ability to add employees, display employee details, and search for employees. By employing these concepts, the Automated Performance Appraisal System simplifies and automates the performance appraisal process, improving efficiency and accuracy within an organization.

Source Code

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
class Employee {
protected:
string name;
int id;
double rating;
public:
Employee(const string &name, int id, double rating)
: name(name), id(id), rating(rating) {} int getID()
const { return id; }
virtual void displayDetails() const {
cout << "Name: " << name << "\n"
<< "ID: " << id << "\n"
<< "Rating: " << rating << "\n";
}
virtual ~Employee()
{
cout<<"c++ projected executed!!"<<endl;
}
};
class Manager : public Employee {
private:
vector<Employee *>
employess;
public:
Manager(const string &name, int id, double rating)
: Employee(name, id, rating) {}
void addEmployee(Employee *emp) {
employess.push_back(emp);
cout << "Succesfully added employee!\n"; }
void displayEmployees() const {
if (employess.empty()) // checking whether the vector is empty or not
{
cout << "No employess to show"
<< "\n";
return;
} else {
cout << "====================" << endl; for
(const auto &emp : employess) {
emp->displayDetails();
cout << "------"
<< "\n";
}
}
}
virtual ~Manager()
{
for(const auto &emp:employess)
{
delete emp;
cout<<"Manager object deleted"<<endl; }
}
void searchEmployee(int empID) const {
if (employess.empty()) {
cout << "No employess "
<< "\n";
return;
} else {
bool check = false;
for (const auto &emp : employess) { if
(emp->getID() == empID) {
cout << "====================" << endl; cout
<< "Employee found!"
<< "\n";
emp->displayDetails();
check = true;
break;
}
}
if (!check) {
cout << "Employee not found\n";
}
}
}
};
Manager manager("shaun",0,5);
void addEmployee(){
string name;
int id;
double rating;
cout << "Enter employee name: ";
cin.ignore();
```

```
getline(cin, name);
cout << "Enter employee ID: ";
cin >> id;
cout << "Enter employee rating: ";
cin >> rating;
Employee *emp = new Employee(name, id, rating);
manager.addEmployee(emp);
}
void searchEmployee() {
int empId;
cout << "Enter employee ID to search:";
cin >> empId;
manager.searchEmployee(empId);
}
int main() {
int usertype;
cout << "Enter user type 1->Employee 2->Manager\n";
cin >> usertype;
if(usertype==2)
{
int choice;
do {
cout << "=====================" << endl; cout <<
"Employee Performance Review" << endl; cout <<
"=====================" << endl;
cout << "1. Add Employee" << endl;
cout << "2. Display Employees"<< endl;
cout << "3. Search Employee" << endl;
cout << "4. Exit" << endl;
cout<<"5.Change mode"<<endl;
cout << "Enter your choice: ";
cin >> choice;
switch (choice) {
case 1:
addEmployee();
break;
case 2:
manager.displayEmployees();
break;
case 3:
searchEmployee();
break;
case 4:
cout << "Exiting program..." << endl;
break;
case 5:
cout<<"Changed mode"<<endl;
usertype=1;
break;
default:
cout << "Invalid choice. Please try again." <<
endl;
break;
}
cout << endl;
}while(choice!=4 && usertype==2);
}
if(usertype==1)
{
int choice;
do {
cout << "------"
<< "\n";
cout << "Employee Performance Review!" <<
"\n";
cout << "Choose: "
<< "\n";
cout << "1.Display Employee Details: " << endl;
cout << "2.Exit" << endl;
cout << "Enter your choice: ";
cin>>choice;
switch (choice) {
case 1:
searchEmployee();
break;
case 2:
cout << "Exiting"
<< "\n";
break;
default:
cout << "Invalid choice.Please try again" << endl;
break;
}
cout << endl;
} while (choice != 2);
}
return 0;
};
```

Features of Project

The above code contains the following features from c++ :

- Classes + objects along with few member functions and data members • Inheritence / derived classes
- Vectors
- Loops
- Conditional checks
- Virtual functions

Classes and respective member variables and functions involved in this code are :

1) `Employee` [base class] - has the ability to accept basic employee details and display them as an object.

- `name` [protected / data member] - stores the name of the employee as a string.
- id [protected / data member] - stores the ID of the employee as int. • `rating` [protected / data member] - stores the rating provided by the manager as double.
- `Employee()` [public/ constructor function] - the details of the employee object is taken from the user and assigned through a initializer list. • `getID()` [public/ member function] - returns the protected data member - id , to prevent attempts of editing the data but only access to view it. • `displayDetails()` [public / virtual member function] - displays all the details of an object.
- `~Employee()` [public / virtual destructor function] - deallocates the memory of all objects belonging to this class. Since it is virtual it also allows for the deallocation of any derived classes if they exist.

2) `Manager` [derived class] - class is derived from Employee Class and has added features like : storing multiple employee objects, searching for an employee and displaying their details. All the member functions of Employee class are already inherited here this is achieved by setting the access modifier as public.

- `employess` [private / data member] - vector that stores array of pointers to objects of Employee class which is used to display, add and search for a particular employee using their ID .We made the use of vector rather than an array as we need a dynamic array to store details of a given employee. We have deleted the vector at the end of program in order to clear any memory allocated to it.

- `Manager()` [public / constructor function] - that takes name, id and rating of the employee and stores it in an Employee class.
- `addEmployee()` [public/ member function] - public function which add the object created of employee class into the vector employess. • `displayEmployees()` [public/ member function] - check whether the employess vector is empty and displays details of all employees using a for range to call on `displayDetails()` from the base class in case of the contrary.
- `searchEmployee()` [public / member function] - check whether the employess vector is empty. If not, compares the Employee id using the for range, accessing the Employee Id using the `getID()`
- `Manager manager("shaun",4,5)` [global object] - an initial user is created of class Manager
- `~Manager()` -[public / virtual destructor function] - deallocates the memory of all objects belonging to this class.

3) `main()` [main function] - allows for user to interact and use all the classes and function defined in the whole program.

- `usertype` [int variable] - stores the type of user : Employee or Manager ; which both have different permissions, the value of usertype is set 1 to enter as Employee and is set to 2 to enter as Manager.
- `choice` [int variable] - stores the choice of action any particular user is allowed used as a parameter in switch cases.
- `do-while` [loop] - asks the user for the next action to be performed until the user wishes to exit
- `switch case` [conditional statement] - used to execute particular choice of the user and call the according function, Each type of user type have been given different options of action such as :

1) Manager
- Can add Employees
- Can display Employee Details
- Can Search for an Employee
- Can change to employee mode
- Exit

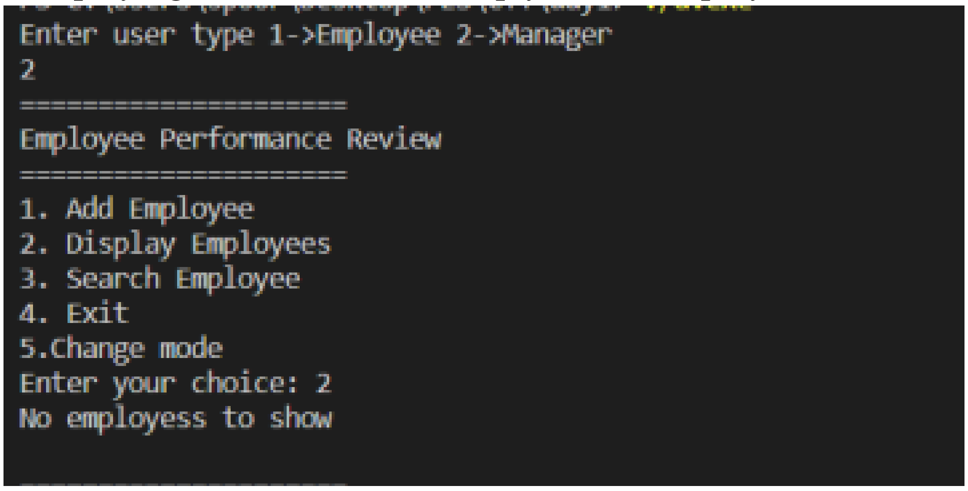
2) Employee

- Can display the rating given by the manager (cannot make any changes regarding this)
- Can exit

Outputs

1) As Manager:

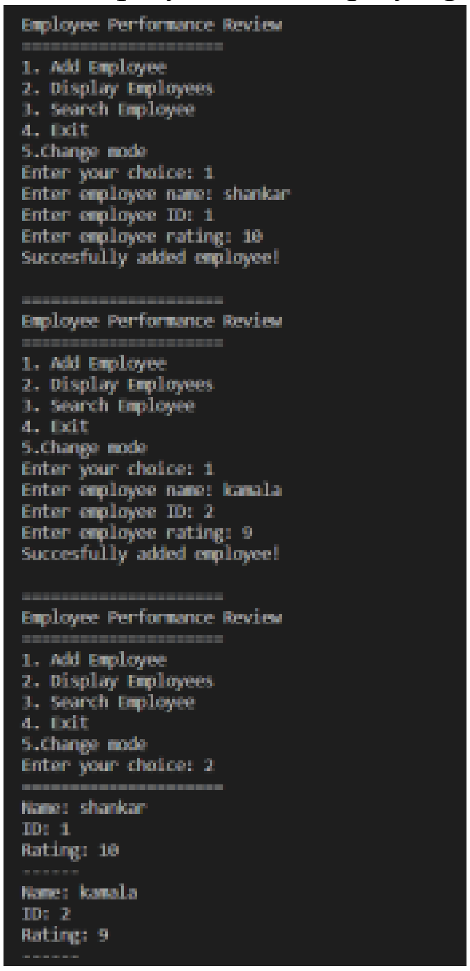
- Displaying the contents of a empty vector employees



```
Enter user type 1->Employee 2->Manager
2
======================
Employee Performance Review
======================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 2
No employess to show
```

•

Add an employees and displaying the employee details

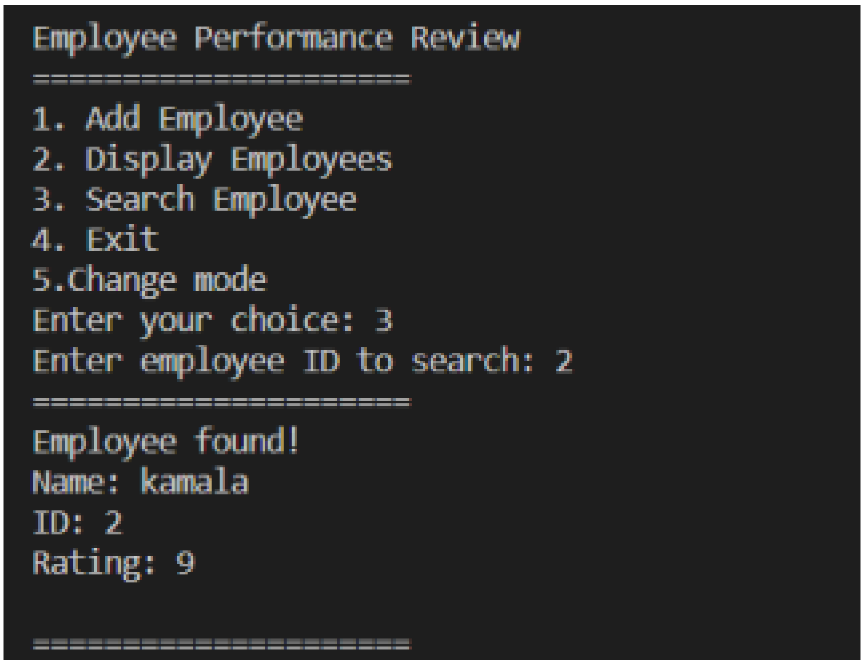


```
Employee Performance Review
====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 1
Enter employee name: shankar
Enter employee ID: 1
Enter employee rating: 10
Succesfully added employee!

====================
Employee Performance Review
====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 1
Enter employee name: kamala
Enter employee ID: 2
Enter employee rating: 9
Succesfully added employee!

====================
Employee Performance Review
====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 2
====================
Name: shankar
ID: 1
Rating: 10
------
Name: kamala
ID: 2
Rating: 9
------
```

• Searching for an employee

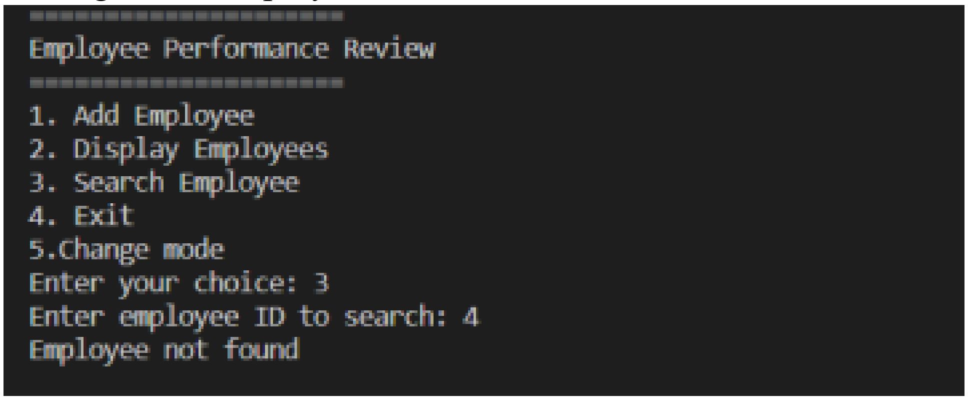


```
Employee Performance Review
=====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 3
Enter employee ID to search: 2
=====================
Employee found!
Name: kamala
ID: 2
Rating: 9

=====================
```

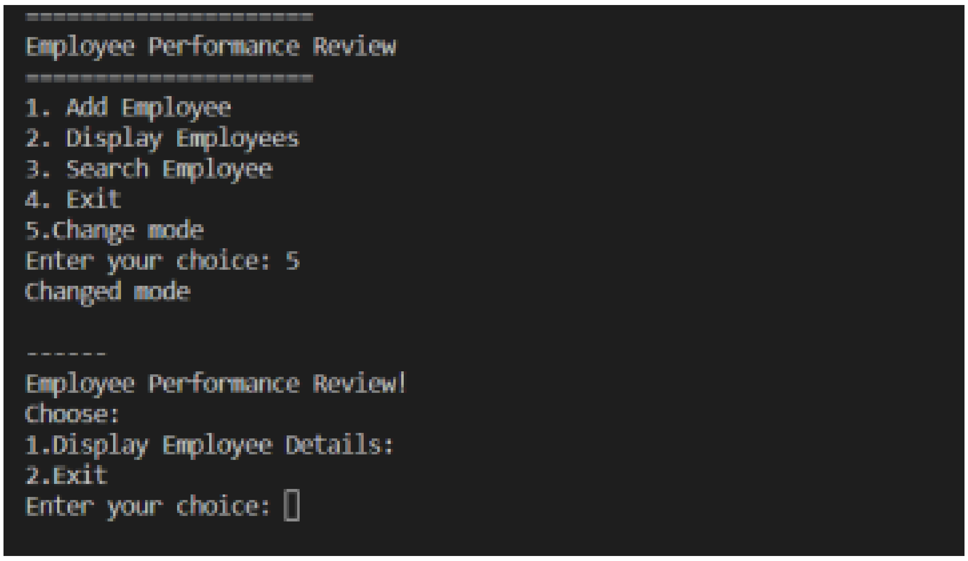
•

Searching for an employee which hasn't been added into the vector



```
=====================
Employee Performance Review
=====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 3
Enter employee ID to search: 4
Employee not found
```

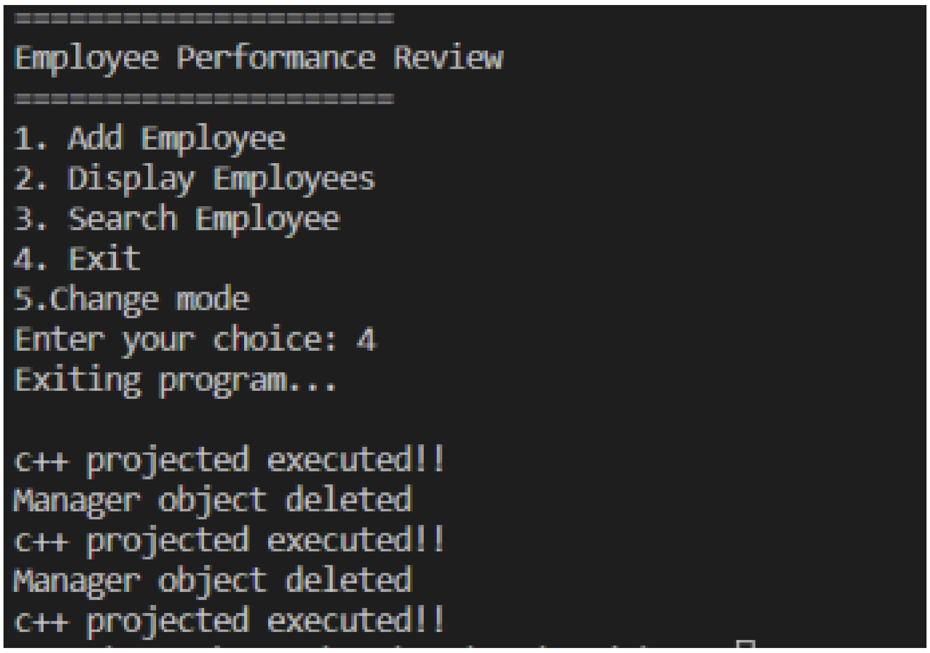
• Changing into Employee Mode



```
=====================
Employee Performance Review
=====================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 5
Changed mode

------
Employee Performance Review!
Choose:
1.Display Employee Details:
2.Exit
Enter your choice: 
```

• Exiting:

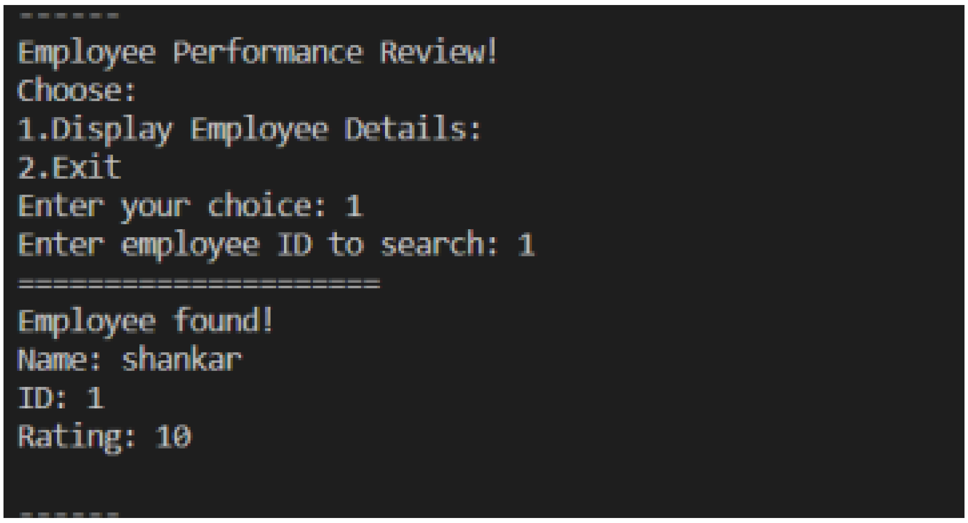


```
======================
Employee Performance Review
======================
1. Add Employee
2. Display Employees
3. Search Employee
4. Exit
5.Change mode
Enter your choice: 4
Exiting program...

c++ projected executed!!
Manager object deleted
c++ projected executed!!
Manager object deleted
c++ projected executed!!
```

2) As Employee

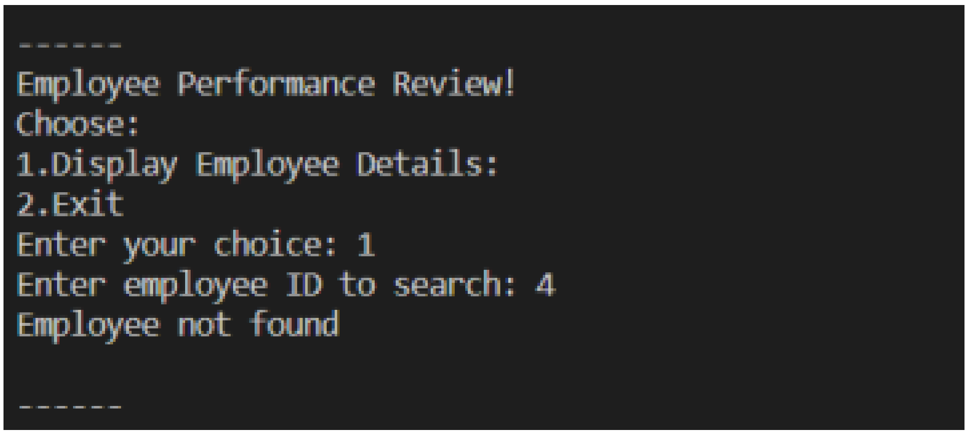
• Displaying the contents of a employee by taking ID as an input whose review has been entered:



```
------
Employee Performance Review!
Choose:
1.Display Employee Details:
2.Exit
Enter your choice: 1
Enter employee ID to search: 1
=====================
Employee found!
Name: shankar
ID: 1
Rating: 10

------
```

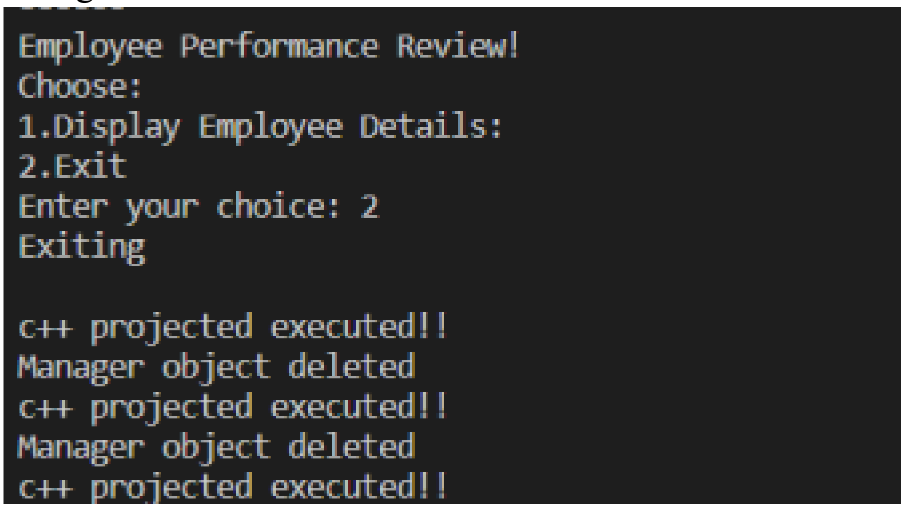
• Displaying the contents of a employee by taking ID as an input whose review hasn't been entered:



```
------
Employee Performance Review!
Choose:
1.Display Employee Details:
2.Exit
Enter your choice: 1
Enter employee ID to search: 4
Employee not found

------
```

• Exiting:



```
Employee Performance Review!
Choose:
1.Display Employee Details:
2.Exit
Enter your choice: 2
Exiting

c++ projected executed!!
Manager object deleted
c++ projected executed!!
Manager object deleted
c++ projected executed!!
```

Here we see that two object of class Employee was added into the vector as we are deleting the vector the object stored are deleted as a result the destructor function of the class Manager has been called and the base class Employee destructor function has been called twice! we did have a global object of class Employee which was deleted as a result there is another destructor function of class Employee called as a result we have three destructor function of class Employee called and two destructor function of class Manager called.

Future Enhancements

Adding GUI(like dashboard)
Using firebase as database to store details of employees and their review
Check for validity of inputs