[Date]

FUNDAMENTAL OF COMPUTER PROGRAMMING

**LAB\_12**

**HANAN MAJEED**

**CMS: 519166**

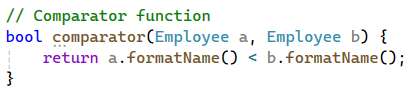
**Task:**

Write a modular program to manage a small company’s employee data. Each employee will have the following details:

* ID (integer)
* First Name (string)
* Last Name (string)
* Position (string)
* Department (Enum: HR, Finance, IT, Sales, Research, Production)
* Salary (float)

**Requirements:**

1. **Structure Definition:**
   * Define a structure “Employee” that holds all the employee details.
   * Include the following functions within the structure:
     + **formatName()**: A function that formats the employee's name by combining first and last name. Also, capitalize the first letter of each name. use function toupper() to change the string to upper case.
     + **calculateTax()**: A function that calculates income tax based on the salary:
       - If salary < 50,000, tax = 5% of salary.
       - If salary >= 50,000 and < 100,000, tax = 10% of salary.
       - If salary >= 100,000, tax = 15% of salary.
     + **displayDetails()**: A function that displays the employee details in a formatted manner.
2. **Main Program Tasks:** 
   * Create an array of employees that maintain the list of employees in the company
   * The program should allow a user to:
     + Add new employees
     + Delete employee based on his/her ID.
       - When an employee is deleted, the position they occupied in the array will become vacant. To maintain a continuous list of employee records without gaps, shift all the records that come after the deleted employee's position one step back in the array. This ensures that all employee records are stored consecutively.
     + Edit data of existing employees
     + Show the list of all employee names in ascending order (use sort() function with custom comparator)



* + - Show full details of the selected employee

**INPUT:**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string>

using namespace std;

// Enum for Departments

enum Department {

HR, Finance, IT, Sales, Research, Production

};

// Function to convert Department enum to string

string departmentToString(Department dept) {

switch (dept) {

case HR:

return "HR";

case Finance:

return "Finance";

case IT:

return "IT";

case Sales:

return "Sales";

case Research:

return "Research";

case Production:

return "Production";

default:

return "Unknown";

}

}

// Structure for employee data

struct Employee {

int id;

string firstName, lastName, position;

Department department;

float salary;

// Format the employee's name

string formatName() const {

string fullName = firstName + " " + lastName;

for (size\_t i = 0; i < fullName.length(); i++) {

if (i == 0 || fullName[i - 1] == ' ')

fullName[i] = (fullName[i] >= 'a' && fullName[i] <= 'z') ? fullName[i] - 32 : fullName[i];

else

fullName[i] = (fullName[i] >= 'A' && fullName[i] <= 'Z') ? fullName[i] + 32 : fullName[i];

}

return fullName;

}

// Tax Calculation

float calculateTax() const {

if (salary < 50000)

return salary \* 0.05;

if (salary < 100000)

return salary \* 0.10;

return salary \* 0.15;

}

// Display employee details

void displayDetails() const {

cout << "ID: " << id << endl;

cout << "Name: " << formatName() << endl;

cout << "Position: " << position << endl;

cout << "Department: " << departmentToString(department) << endl;

cout << "Salary: $" << fixed << setprecision(2) << salary << endl;

cout << "Tax: $" << fixed << setprecision(2) << calculateTax() << endl;

}

};

// Comparator function to compare Employees based on formatted name

bool comparator(const Employee& a, const Employee& b) {

return a.formatName() < b.formatName(); // Compare formatted names

}

// Function to add new employee

void addEmployee(Employee employees[], int& count) {

Employee emp;

cout << "Enter Employee ID: ";

cin >> emp.id;

cout << "Enter First Name: ";

cin >> emp.firstName;

cout << "Enter Last Name: ";

cin >> emp.lastName;

cout << "Enter Position: ";

cin.ignore();

getline(cin, emp.position);

cout << "Enter Department (0: HR, 1: Finance, 2: IT, 3: Sales, 4: Research, 5: Production): ";

int dept;

cin >> dept;

emp.department = static\_cast<Department>(dept);

cout << "Enter Salary: ";

cin >> emp.salary;

employees[count++] = emp;

}

// Function to delete employee by using ID No

void deleteEmployee(Employee employees[], int& count, int id) {

int index = -1;

for (int i = 0; i < count; i++) {

if (employees[i].id == id) {

index = i;

break;

}

}

if (index != -1) {

for (int i = index; i < count - 1; i++) {

employees[i] = employees[i + 1];

}

count--;

cout << "Employee with ID " << id << " has been deleted.\n";

}

else {

cout << "Employee with ID " << id << " not found.\n";

}

}

// Function to edit existing employee

void editEmployee(Employee employees[], int count, int id) {

int index = -1;

for (int i = 0; i < count; i++) {

if (employees[i].id == id) {

index = i;

break;

}

}

if (index != -1) {

cout << "Editing Employee with ID " << id << ".\n";

cout << "Enter New First Name: ";

cin >> employees[index].firstName;

cout << "Enter New Last Name: ";

cin >> employees[index].lastName;

cout << "Enter New Position: ";

cin.ignore();

getline(cin, employees[index].position);

cout << "Enter New Department (0: HR, 1: Finance, 2: IT, 3: Sales, 4: Research, 5: Production): ";

int dept;

cin >> dept;

employees[index].department = static\_cast<Department>(dept);

cout << "Enter New Salary: ";

cin >> employees[index].salary;

}

else {

cout << "Employee with ID " << id << " not found.\n";

}

}

// Function to show all employee names in ascending order

void showAllEmployeeNames(Employee employees[], int count) {

// Sort employees array using the comparator function

sort(employees, employees + count, comparator);

cout << "\nEmployee Names in Ascending Order:\n";

for (int i = 0; i < count; i++) {

cout << employees[i].formatName() << endl;

}

}

// Function to display details of selected employee

void showEmployeeDetails(Employee employees[], int count, int id) {

int index = -1;

for (int i = 0; i < count; i++) {

if (employees[i].id == id) {

index = i;

break;

}

}

if (index != -1) {

employees[index].displayDetails();

}

else {

cout << "Employee with ID " << id << " not found.\n";

}

}

int main() {

Employee employees[100];

int count = 0;

int choice;

while (true) {

cout << "\nEmployee Management System:\n";

cout << "1. Add New Employee\n";

cout << "2. Delete Employee\n";

cout << "3. Edit Employee\n";

cout << "4. Show All Employee Names (Ascending Order)\n";

cout << "5. Show Employee Details\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

if (choice == 6) {

cout << "Exiting Program..." << endl;

break;

}

switch (choice) {

case 1:

addEmployee(employees, count);

cout << "EMPLOYEE ADDED SUCCESSFULLY" << endl;

break;

case 2:

{

int id;

cout << "Enter Employee ID to delete: ";

cin >> id;

deleteEmployee(employees, count, id);

break;

}

case 3:

{

int id;

cout << "Enter Employee ID to edit: ";

cin >> id;

editEmployee(employees, count, id);

break;

}

case 4:

showAllEmployeeNames(employees, count);

break;

case 5:

{

int id;

cout << "Enter Employee ID to show details: ";

cin >> id;

showEmployeeDetails(employees, count, id);

break;

}

default:

cout << "Invalid choice. Please try again.\n";

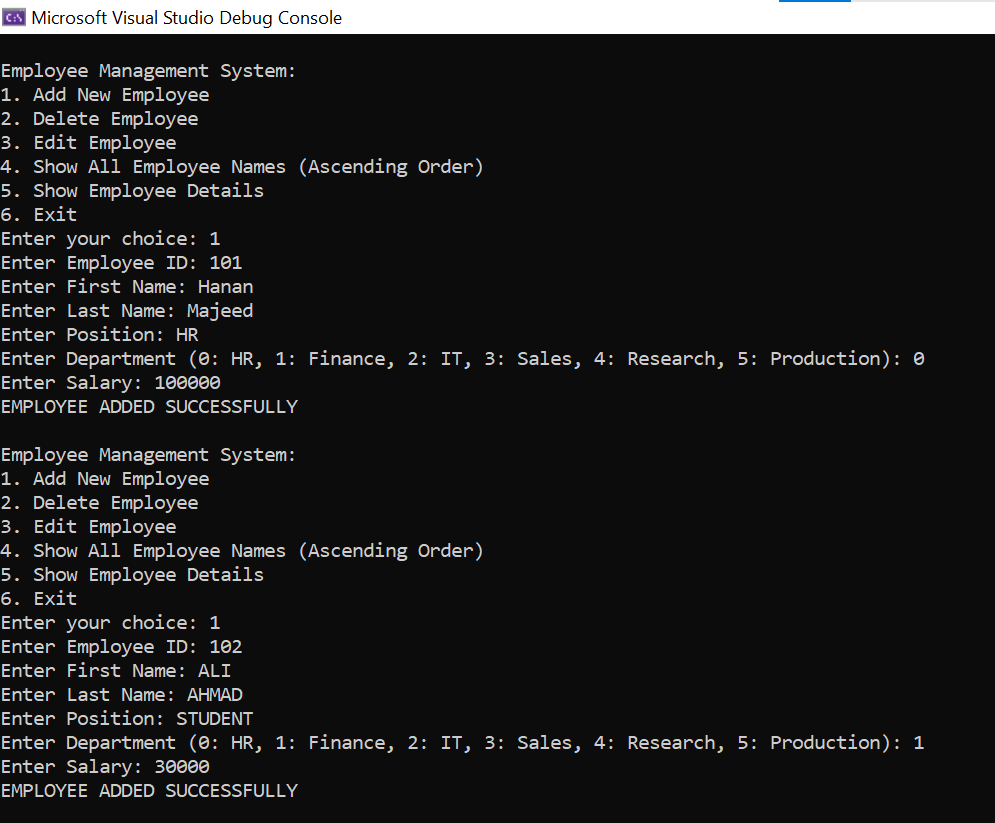
}

}

return 0;

}

**OUTPUT:**



A screenshot of a computer program

Description automatically generated

A computer screen with white text

Description automatically generated