DAA project (by SQUAD)

SOURCE CODE(University)

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class University {
public:
    int ID;
    string univ_code;
    string univ_name;
    string univ_address;
    string univ_mail;
    string univ_website;

public:
    University(int ID, const string &univ_code, const string
&univ_name,
               const string &univ_address, const string &univ_mail, const
string &univ_website)
        : ID(ID), univ_code(univ_code), univ_name(univ_name),
univ_address(univ_address), univ_mail(univ_mail),
univ_website(univ_website) {}

    int getID() const { return ID; }
    void setID(int ID) { this->ID = ID; }

    string getUnivCode() const { return univ_code; }
    void setUnivCode(const string &univ_code) { this->univ_code =
univ_code; }

    string getUnivName() const { return univ_name; }
```

```
    void setUnivName(const string &univ_name) { this->univ_name =
univ_name; }

    string getUnivAddress() const { return univ_address; }
    void setUnivAddress(const string &univ_address) {
this->univ_address = univ_address; }

    string getUnivMail() const { return univ_mail; }
    void setUnivMail(const string &univ_mail) { this->univ_mail =
univ_mail; }

    string getUnivWebsite() const { return univ_website; }
    void setUnivWebsite(const string &univ_website) {
this->univ_website = univ_website; }

    void displayInfo() const {
        cout << "University ID: " << ID << endl;
        cout << "University Code: " << univ_code << endl;
        cout << "University Name: " << univ_name << endl;
        cout << "University Address: " << univ_address << endl;
        cout << "University Email: " << univ_mail << endl;
        cout << "University Website: " << univ_website << endl;
    }
};

vector<pair<University, string>> sort_by_code;
vector<pair<University, string>> sort_by_name;
vector<pair<University, string>> sort_by_address;
vector<pair<University, string>> sort_by_email;
vector<pair<University, string>> sort_by_web;

// Function to perform insertion sort on a vector of strings
void squad_university_insertionSort(vector<pair<University,
string>> &arr) {
    for (int i = 1; i < arr.size(); i++) {
```

```
        string key2 = arr[i].second;
        University key1 = arr[i].first;
        int j = i - 1;

        while (j >= 0 && arr[j].second > key2) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = {key1, key2};
    }
}

int squad_university_binarySearch(const vector<pair<University,
string>> &sortedArr, const string &key) {
    int left = 0, right = sortedArr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (sortedArr[mid].second == key) return mid;
        else if (sortedArr[mid].second < key) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}

void updateSortedVectors(const vector<University> &universities) {
    sort_by_code.clear();
    sort_by_name.clear();
    sort_by_address.clear();
    sort_by_email.clear();
    sort_by_web.clear();

    for (const auto &univ : universities) {
        sort_by_code.push_back({univ, univ.getUnivCode()});
        sort_by_name.push_back({univ, univ.getUnivName()});
        sort_by_address.push_back({univ, univ.getUnivAddress()});
```

```
        sort_by_email.push_back({univ, univ.getUnivMail()});
        sort_by_web.push_back({univ, univ.getUnivWebsite()});
    }

    squad_university_insertionSort(sort_by_code);
    squad_university_insertionSort(sort_by_name);
    squad_university_insertionSort(sort_by_address);
    squad_university_insertionSort(sort_by_email);
    squad_university_insertionSort(sort_by_web);
}

void squad_University_create(vector<University> &universities) {
    int ID;
    string univ_code, univ_name, univ_address, univ_mail,
univ_website;
    cout << "Enter University ID: ";
    cin >> ID;
    cout << "Enter University Code: ";
    cin >> univ_code;
    cout << "Enter University Name: ";
    cin.ignore();
    getline(cin, univ_name);
    cout << "Enter University Address: ";
    getline(cin, univ_address);
    cout << "Enter University Email: ";
    getline(cin, univ_mail);
    cout << "Enter University Website: ";
    getline(cin, univ_website);

    universities.emplace_back(ID, univ_code, univ_name,
univ_address, univ_mail, univ_website);

    updateSortedVectors(universities);
    cout << "University added successfully!" << endl;
}
```

```
void squad_University_update(vector<University> &universities, int
ID) {
    for (auto &univ : universities) {
        if (univ.getID() == ID) {
            string univ_code, univ_name, univ_address, univ_mail,
univ_website;
            cout << "Enter new University Code: ";
            cin >> univ_code;
            cout << "Enter new University Name: ";
            cin.ignore();
            getline(cin, univ_name);
            cout << "Enter new University Address: ";
            getline(cin, univ_address);
            cout << "Enter new University Email: ";
            getline(cin, univ_mail);
            cout << "Enter new University Website: ";
            getline(cin, univ_website);

            univ.setUnivCode(univ_code);
            univ.setUnivName(univ_name);
            univ.setUnivAddress(univ_address);
            univ.setUnivMail(univ_mail);
            univ.setUnivWebsite(univ_website);

            updateSortedVectors(universities);
            cout << "University updated successfully!" << endl;
            return;
        }
    }
    cout << "University with ID " << ID << " not found." << endl;
}

void squad_University_delete(vector<University> &universities, int
ID) {
```

```
    for (auto it = universities.begin(); it != universities.end(); ++it) {
        if (it->getID() == ID) {
            universities.erase(it);
            updateSortedVectors(universities);
            cout << "University deleted successfully!" << endl;
            return;
        }
    }
    cout << "University with ID " << ID << " not found." << endl;
}

void searchUniversity(const vector<University> &universities) {
    int searchChoice;
    cout << "\n--- Search Options ---" << endl;
    cout << "1. Search by ID" << endl;
    cout << "2. Search by Code" << endl;
    cout << "3. Search by Name" << endl;
    cout << "4. Search by Address" << endl;
    cout << "5. Search by Email" << endl;
    cout << "6. Search by Website" << endl;
    cout << "Enter your choice: ";
    cin >> searchChoice;

    bool found = false;
    int index;
    switch (searchChoice) {
        case 1: {
            int ID;
            cout << "Enter University ID: ";
            cin >> ID;
            for (const auto &univ : universities) {
                if (univ.getID() == ID) {
                    univ.displayInfo();
                    found = true;
                }
```

```
            }
            break;
        }
        case 2: {
            string code;
            cout << "Enter University Code: ";
            cin >> code;
            index = squad_university_binarySearch(sort_by_code,
code);
            if (index != -1) {
                cout << "Match found" << endl;
                sort_by_code[index].first.displayInfo();
                found = true;
            }
            break;
        }
        case 3: {
            string name;
            cout << "Enter University Name: ";
            cin.ignore();
            getline(cin, name);
            index = squad_university_binarySearch(sort_by_name,
name);
            if (index != -1) {
                cout << "Match found" << endl;
                sort_by_name[index].first.displayInfo();
                found = true;
            }
            break;
        }
        case 4: {
            string address;
            cout << "Enter University Address: ";
            cin.ignore();
            getline(cin, address);
```

```
            index = squad_university_binarySearch(sort_by_address,
address);
            if (index != -1) {
                cout << "Match found" << endl;
                sort_by_address[index].first.displayInfo();
                found = true;
            }
            break;
        }
        case 5: {
            string mail;
            cout << "Enter University Email: ";
            cin >> mail;
            index = squad_university_binarySearch(sort_by_email,
mail);
            if (index != -1) {
                cout << "Match found" << endl;
                sort_by_email[index].first.displayInfo();
                found = true;
            }
            break;
        }
        case 6: {
            string website;
            cout << "Enter University Website: ";
            cin >> website;
            index = squad_university_binarySearch(sort_by_web,
website);
            if (index != -1) {
                cout << "Match found" << endl;
                sort_by_web[index].first.displayInfo();
                found = true;
            }
            break;
        }
```

```
        default:
            cout << "Invalid search option." << endl;
            return;
    }

    if (!found) {
        cout << "No matching University found." << endl;
    }
}

int main() {
    vector<University> universities;
    int choice, ID;

    do {
        cout << "\n--- University Management System ---" << endl;
        cout << "1. Create University" << endl;
        cout << "2. Update University" << endl;
        cout << "3. Retrieve University" << endl;
        cout << "4. Delete University" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                squad_University_create(universities);
                break;
            case 2:
                cout << "Enter University ID to update: ";
                cin >> ID;
                squad_University_update(universities, ID);
                break;
            case 3:
                searchUniversity(universities);
```

```
            break;
        case 4:
            cout << "Enter University ID to delete: ";
            cin >> ID;
            squad_University_delete(universities, ID);
            break;
        case 5:
            cout << "Exiting program..." << endl;
            break;

        default:
            cout << "Invalid choice. Please try again." << endl;
      }
   } while (choice != 5);

   return 0;
}
```