A report for the CS204:Design and Analysis of Algorithm project

OBE IMPLEMENTATION: UNIVERSITY SETTINGS

by

AP23110011022 | NeeliHarshitha

AP23110011021 | V Himaja

AP23110011012 | K Mohitha

AP23110011071 I K Jaswanth

AP23110011054 | S Harshavardha[RegNo]

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRM UNIVERSITY AP::AMARAVATI

INDEX

Introduction:

The University Management System is a C++ application that allows users to manage university records, including creating, updating, retrieving, and deleting university information such as: University ID, code, name, address, email, and website. Key Features: 1. Create new university records. 2. Update existing university details using ID. 3. Search universities by various attributes. 4. Delete a university by ID. The system uses insertion sort for sorting universities and binary search for efficient retrieval. The program utilizes object-oriented programming to manage university data in a structured way.

Is a software solution designed to manage information about universities efficiently. This system enables the creation, updating, deletion, and retrieval of university records, providing a structured and user-friendly interface for managing university details. The system leverages object-oriented programming principles to model university entities, making it modular and scalable. It offers a menu-driven interface for users to interact with the system and perform various operations on university data.

In this system, each university is represented as an object of the `University` class. This class encapsulates essential attributes of a university, including its ID, code, name, address, email, and website. The system provides functionalities to perform several operations on these university objects, such as adding new universities, modifying existing ones, deleting records, and searching for specific universities based on different attributes.

The system is designed to store and manage university data in a vector of `University` objects. To enhance search efficiency, the universities are sorted by various attributes—such as university code, name, address, email, and website—using a customized sorting mechanism. The sorting is accomplished using the **Insertion Sort** algorithm, and the **Binary Search** algorithm is employed to quickly locate universities based on a specified search criterion.

1. **Create University**: This feature allows the user to add new universities to the system by providing detailed information like university ID, code, name, address, email, and website.

2. **Update University**: Users can modify the details of existing universities by providing a unique university ID. This functionality ensures that users can keep the data up-to-date.

3. **Delete University**: The system allows users to delete a university record based on its unique ID. Once a university is deleted, the system updates the sorted vectors to maintain proper order.

4. **Search University**: Users can search for universities based on various attributes like university ID, code, name, address, email, or website. The system performs the search using **Binary Search** on pre-sorted vectors, ensuring a fast search process.

5. **Sorting**: Universities are stored in sorted order according to multiple attributes (code, name, address, email, and website). This sorting facilitates quick searching and easy retrieval of records.

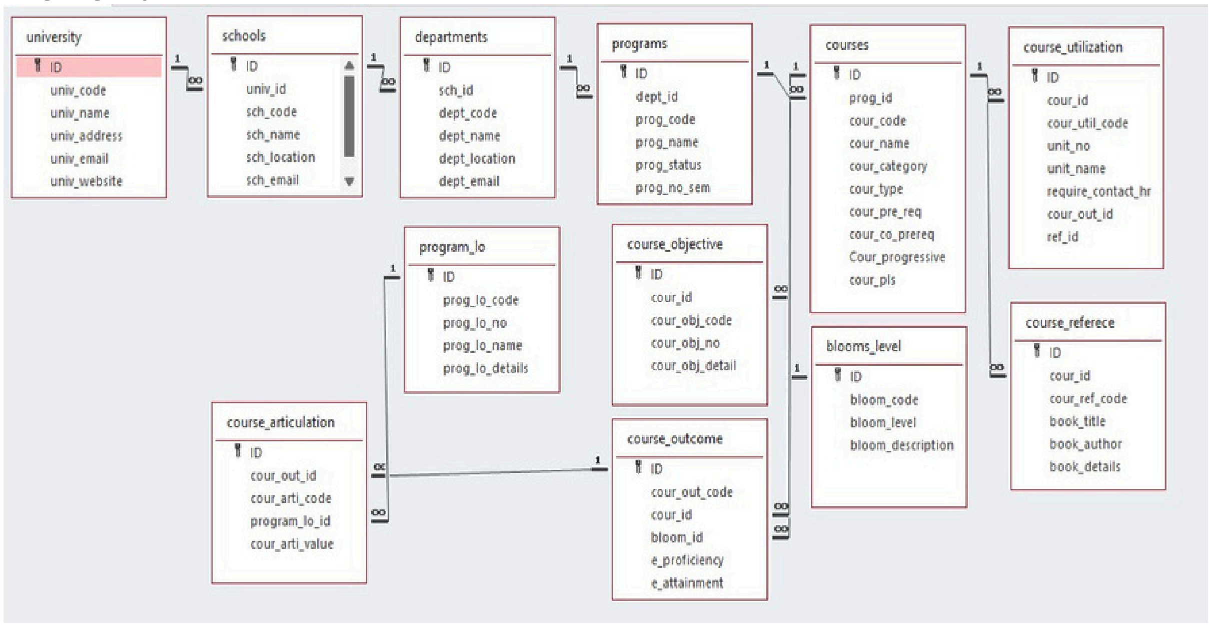
Project Modules:

Various Modules available in the project are

1.Blooms Level setting
2.Program Level Objective Setting 3.University 4.Schools
5.Department 6.Programs
7.Courses 8.Course objective setting
9.Course Outcome Setting
10.Course Articulation matrix Setting 11.Course Utilization Setting 12.Course Reference Setting.

Architecture Diagram

**highlight your module*



Module Description

Module Name: UNIVERSITY SETTINGS
Module Description:

The **University Management System** is organized into several distinct modules, each responsible for a specific functionality within the system. These modules, implemented as individual functions and classes in the program, provide clear separation of concerns, making the code more maintainable, scalable, and easy to understand. Below is a detailed description of each module and its functionality.

1. **University Class Module**
 - **Purpose**: The `University` class represents the core data structure used to model a university. It encapsulates the attributes of a university and provides methods for accessing and modifying those attributes.
 - **Attributes**:
   - `int ID`: The unique identifier for each university.
   - `string univ_code`: The university's code (e.g., a short abbreviation or unique alphanumeric identifier).
   - `string univ_name`: The full name of the university.
   - `string univ_address`: The address of the university.
   - `string univ_mail`: The university's contact email address.
   - `string univ_website`: The university's official website URL.
 - **Methods**:
   - **Getters and Setters**: For each attribute (`getID()`, `setID()`, `getUnivCode()`, `setUnivCode()`, etc.), allowing access and modification of university attributes.
   - **`displayInfo()`**: Displays all the university information in a structured format.

 - **Functionality**: This class is fundamental to storing university data. Every university created, updated, or deleted in the system is represented as an instance of this class.

---

2. **Sorting and Search Module**
 - **Purpose**: This module handles sorting and searching operations on the list of universities. It ensures that the universities are stored in sorted order and allows for efficient searching based on different attributes like university code, name, address, email, and website.
 - **Functions**:
   - **`squad_university_insertionSort()`**:
     - **Purpose**: Performs an **insertion sort** on the universities based on the specific attribute (code, name, address, email, or website).
     - **Operation**: The function iterates over the vector of universities and sorts them in ascending order by comparing values of the sorting criterion (represented as a string).

- **`squad_university_binarySearch()`**:
- **Purpose**: This function performs a **binary search** on a sorted vector of university records.
- **Operation**: After sorting universities based on a chosen attribute (e.g., code, name), it searches for a specific value efficiently by repeatedly dividing the search range in half.
- **Return Value**: If a match is found, it returns the index of the university; otherwise, it returns -1 (indicating no match).
---
3. **University CRUD Operations Module**
- **Purpose**: This module handles the basic **CRUD operations** (Create, Read, Update, Delete) for managing university records.

- **Functions**:
- **`squad_University_create()`**:
- **Purpose**: Allows users to create and add a new university to the system.
- **Operation**: The function prompts the user for all the necessary information (ID, code, name, address, email, and website) and then adds the university to the `universities` vector. Afterward, it updates the sorted vectors for the respective attributes.

- **`squad_University_update()`**:
- **Purpose**: Allows users to update the details of an existing university.
- **Operation**: The function first searches for the university by its ID, then allows the user to modify the university's attributes. Once the update is complete, it re-sorts the university records.

- **`squad_University_delete()`**:
- **Purpose**: Allows users to delete a university by its unique ID.
- **Operation**: The function searches for the university using its ID and, if found, removes the university from the list. It then updates the sorted lists of universities to maintain consistency.

- **`searchUniversity()`**:
- **Purpose**: Provides a search interface for users to find a university by its attributes.
- **Operation**: This function prompts the user to choose a search criterion (ID, code, name, address, email, or website) and performs a search using the **binary search** method on the corresponding sorted vector. If a match is found, it displays the university’s details.

Programming Details naming conventions to be used:

- File name:Squad_university
- Function/method name
    - Create: squad_university_create
    - Update: squad_university_update
    - Retrieve: squad_University_retrive
    - Delete: squad_University_delete
    - Sorting: squad_University_insertion
    - Searching: squad_University_binary

  Comparison(bothsearchingandSorting)
        - For Searching-squad_university_binary search
        - For Sorting-squad_university_insertion sort

    - Time Complexity(both searching and Sorting):
        - For Searching-squad_university_binary search (O(log n))
        - For Sorting-squad_university_insertion sort (O(n^2)

Field/table details:(eg university)[you consider you module]

Field Name	Datatype
id	integer
univ_code	String
univ_name	String
univ_address	String
univ_email	String
univ_website	String

Algorithm Details:

(i)Sorting

- Sorting Algorithm Name: insertion sort

Algorithm:

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step2 - Pick the next element, and store it separately in a key.

Step3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

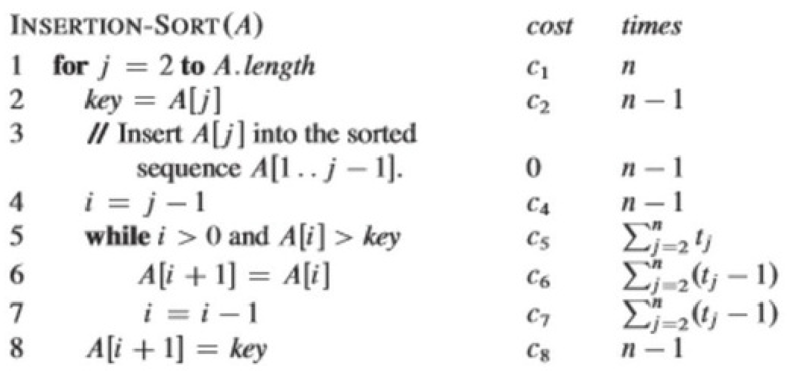
Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted

(ii)Searching

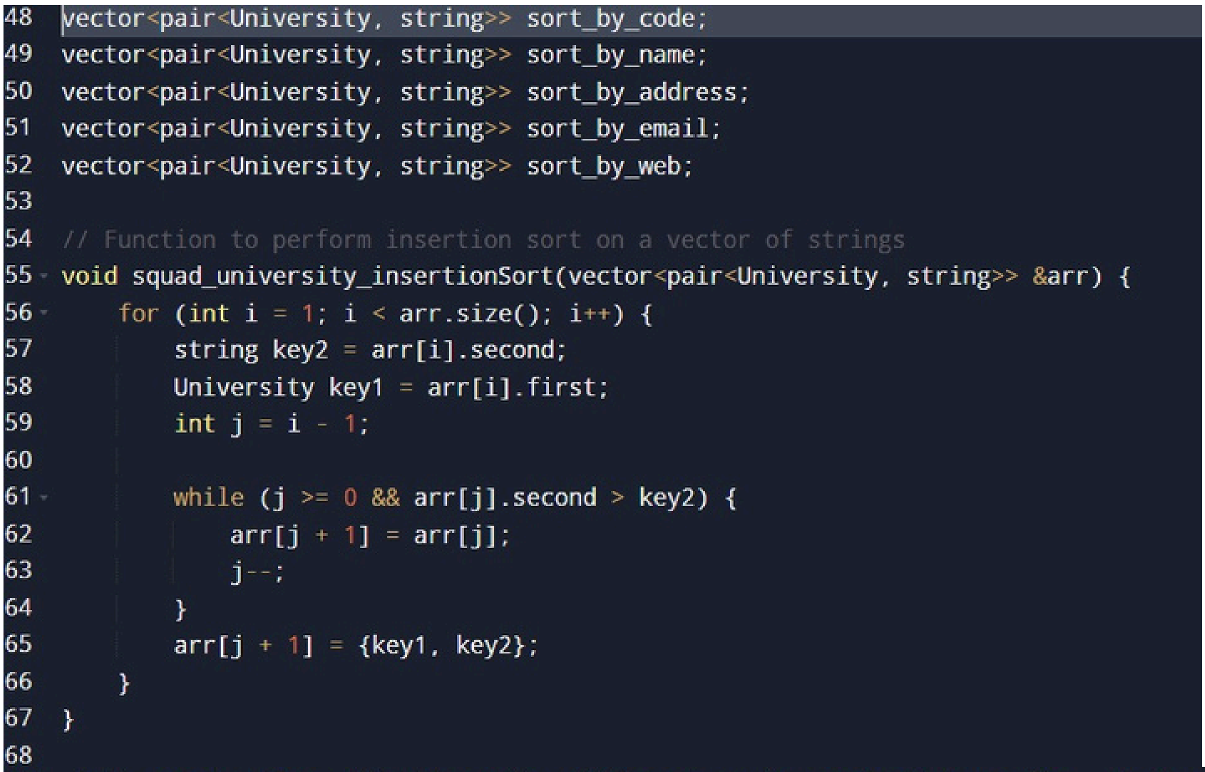
Sorting Algorithm Name: insertion sort

pseudocde:

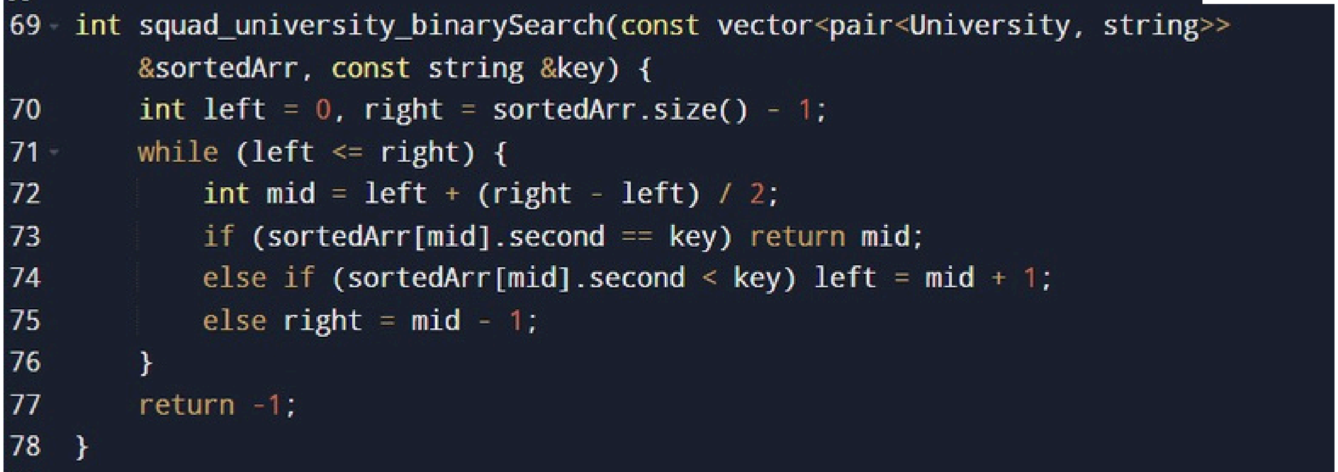


```
INSERTION-SORT(A)                              cost    times
1  for $j = 2$ to $A.length$                   $c_1$   $n$
2      $key = A[j]$                            $c_2$   $n-1$
3      // Insert $A[j]$ into the sorted
           sequence $A[1..j-1]$.               0       $n-1$
4      $i = j - 1$                             $c_4$   $n-1$
5      while $i > 0$ and $A[i] > key$          $c_5$   $\sum_{j=2}^{n} t_j$
6          $A[i+1] = A[i]$                     $c_6$   $\sum_{j=2}^{n} (t_j - 1)$
7          $i = i - 1$                         $c_7$   $\sum_{j=2}^{n} (t_j - 1)$
8      $A[i+1] = key$                          $c_8$   $n-1$
```

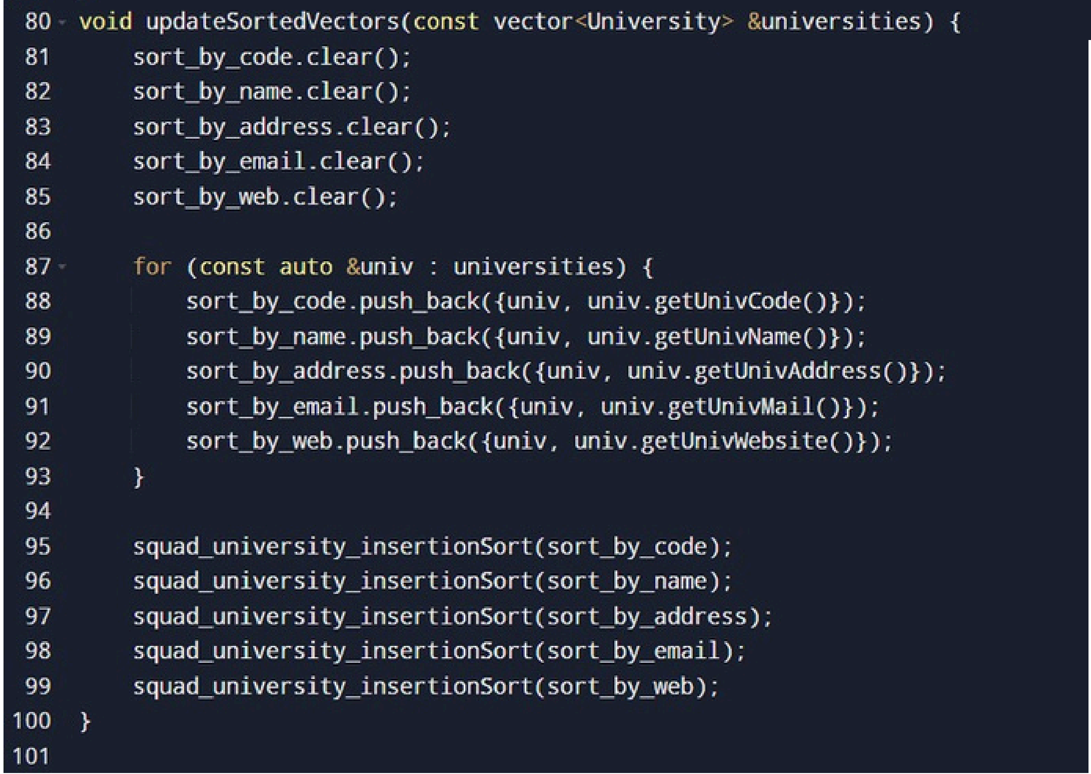
Source Code



```cpp
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};
    }
}

```

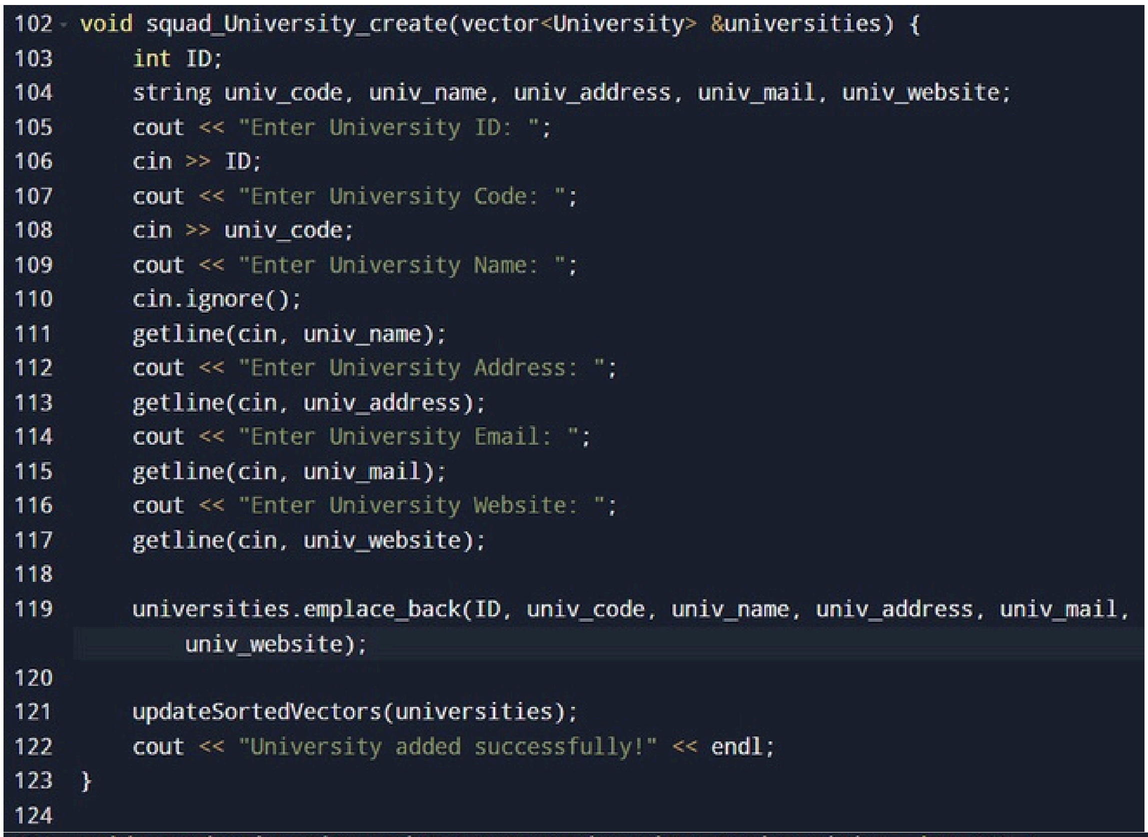


```cpp
int squad_university_binarySearch(const vector<pair<University, string>>
    &sortedArr, const string &key) {
    int left = 0, right = sortedArr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (sortedArr[mid].second == key) return mid;
        else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

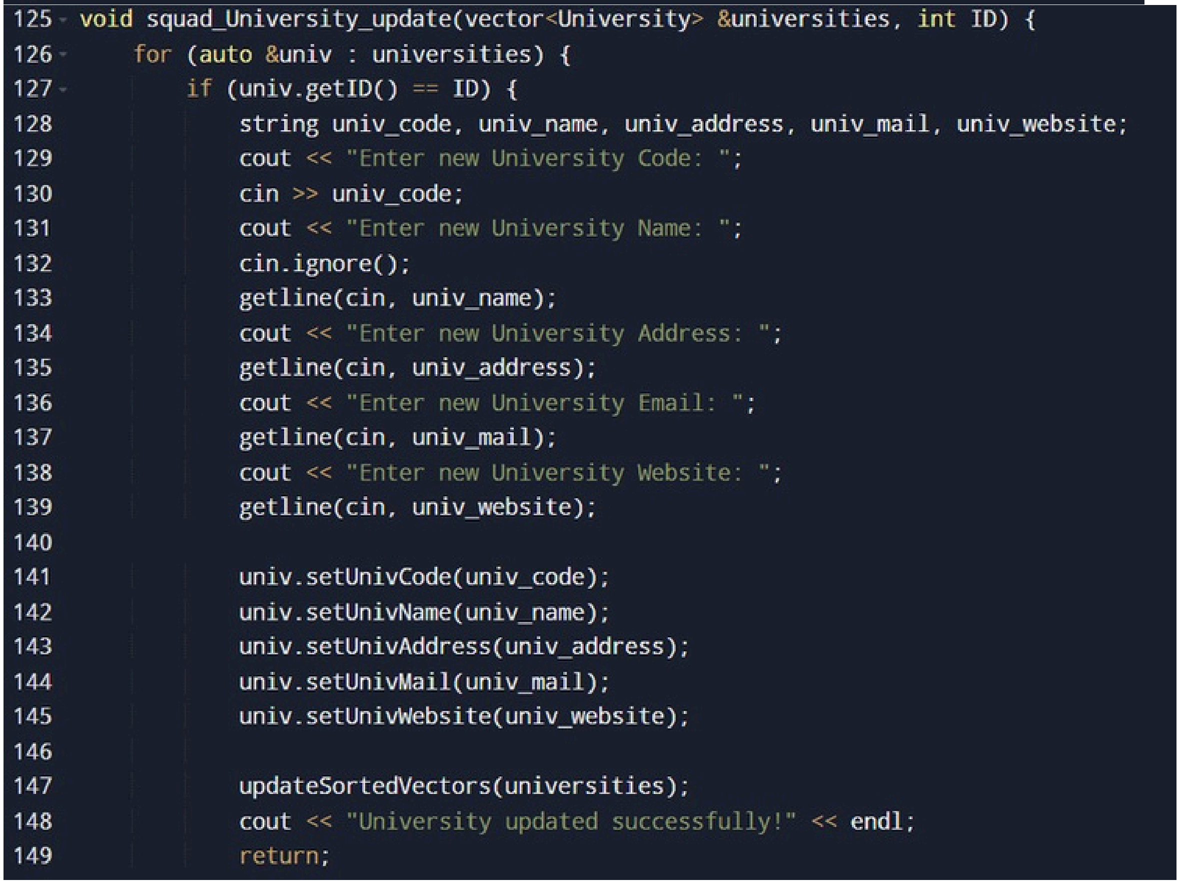


```cpp
void updateSortedVectors(const vector<University> &universities) {
    sort_by_code.clear();
    sort_by_name.clear();
    sort_by_address.clear();
    sort_by_email.clear();
    sort_by_web.clear();

    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);
}

```

Source Code

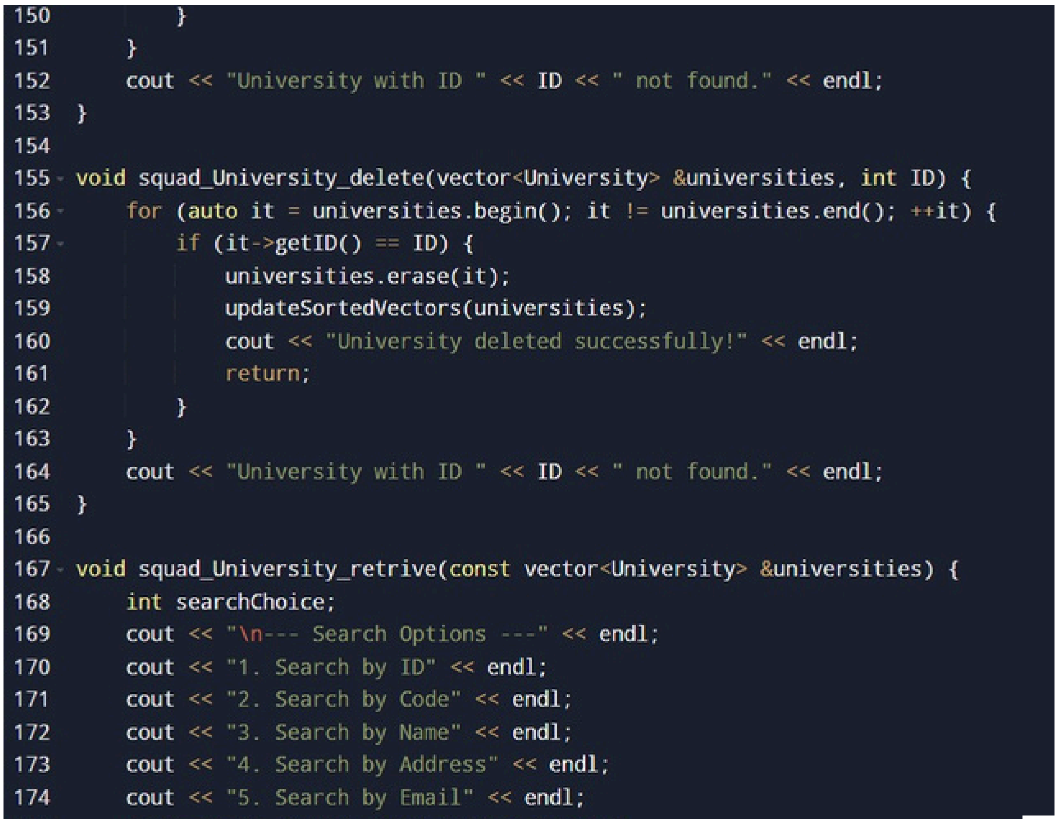


```cpp
void squad_University_create(vector<University> &universities) {
    int ID;
    string univ_code, univ_name, univ_address, univ_mail, univ_website;
    cout << "Enter University ID: ";
    cin >> ID;
    cout << "Enter University Code: ";
    cin >> univ_code;
    cout << "Enter University Name: ";
    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;
}

```

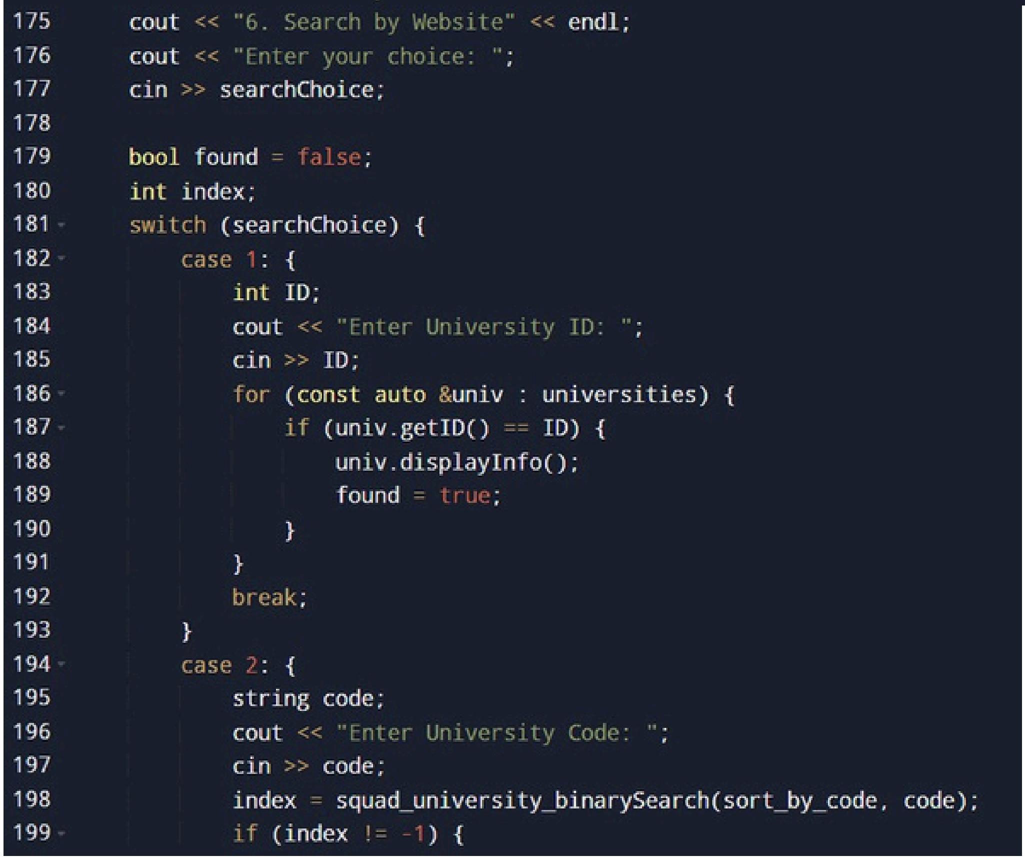


```cpp
void squad_University_update(vector<University> &universities, int ID) {
    for (auto &univ : universities) {
        if (univ.getID() == ID) {
            string univ_code, univ_name, univ_address, univ_mail, univ_website;
            cout << "Enter new University Code: ";
            cin >>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

Source Code

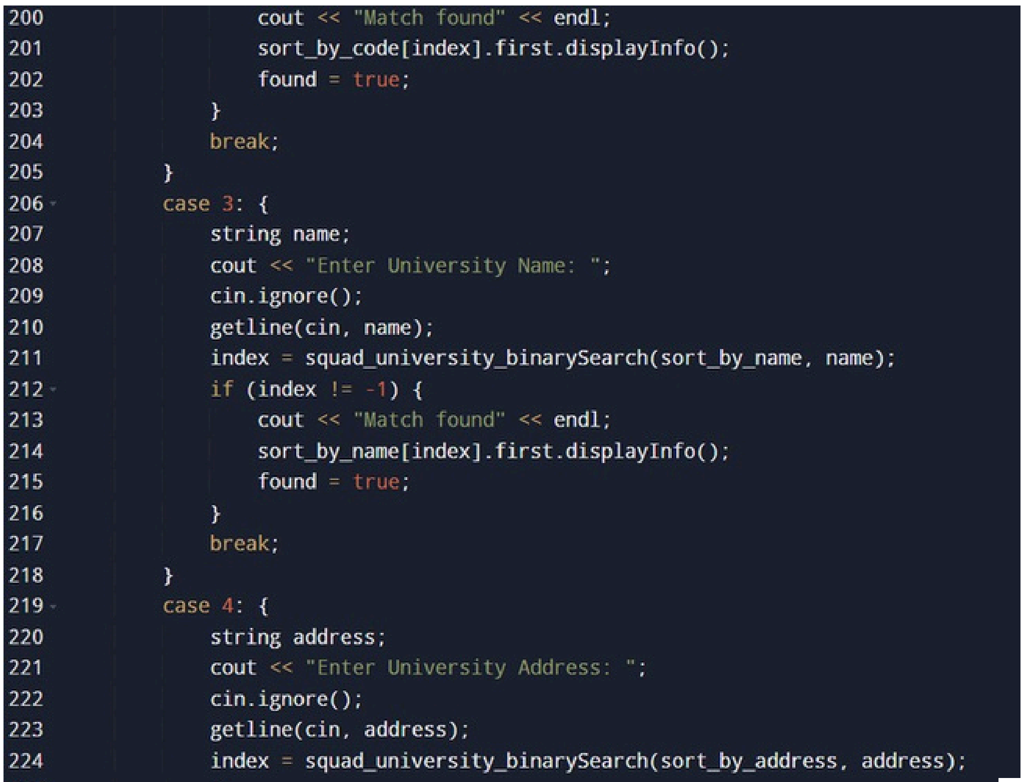


```
        }
    }
    cout << "University with ID " << ID << " not found." << endl;
}

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 squad_University_retrive(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;
```

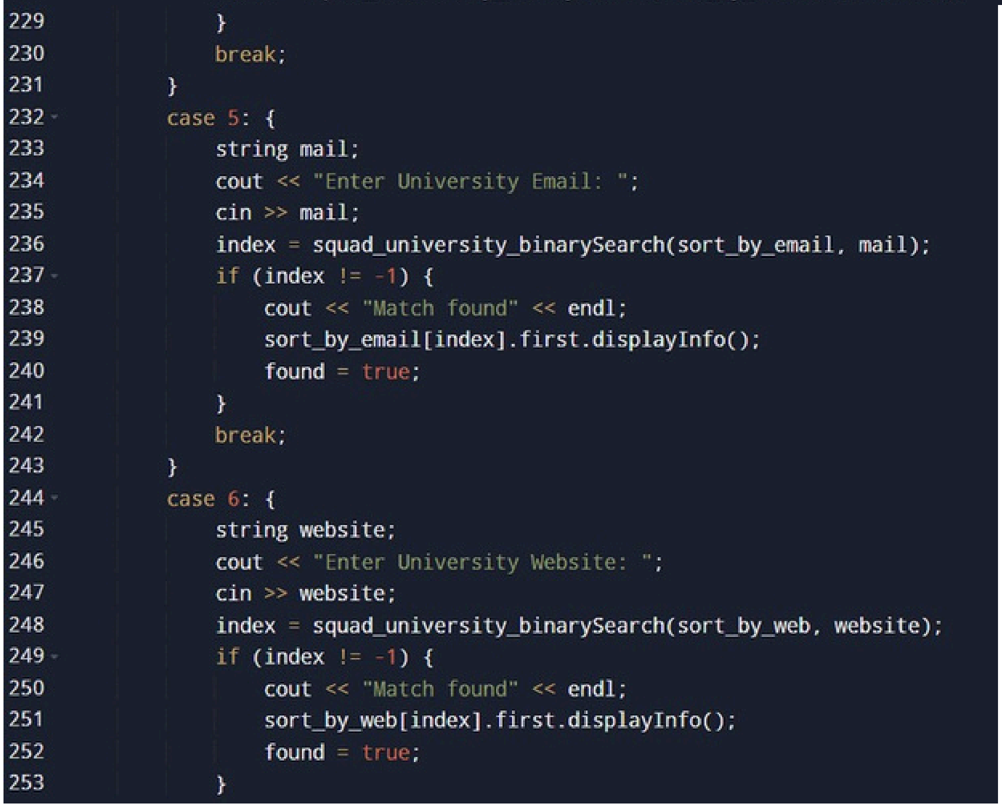


```
    cout << "6. Search by Website" << endl;
    cout << "Enter your choice: ";
    cin >> searchChoice;

    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) {
```

Source Code

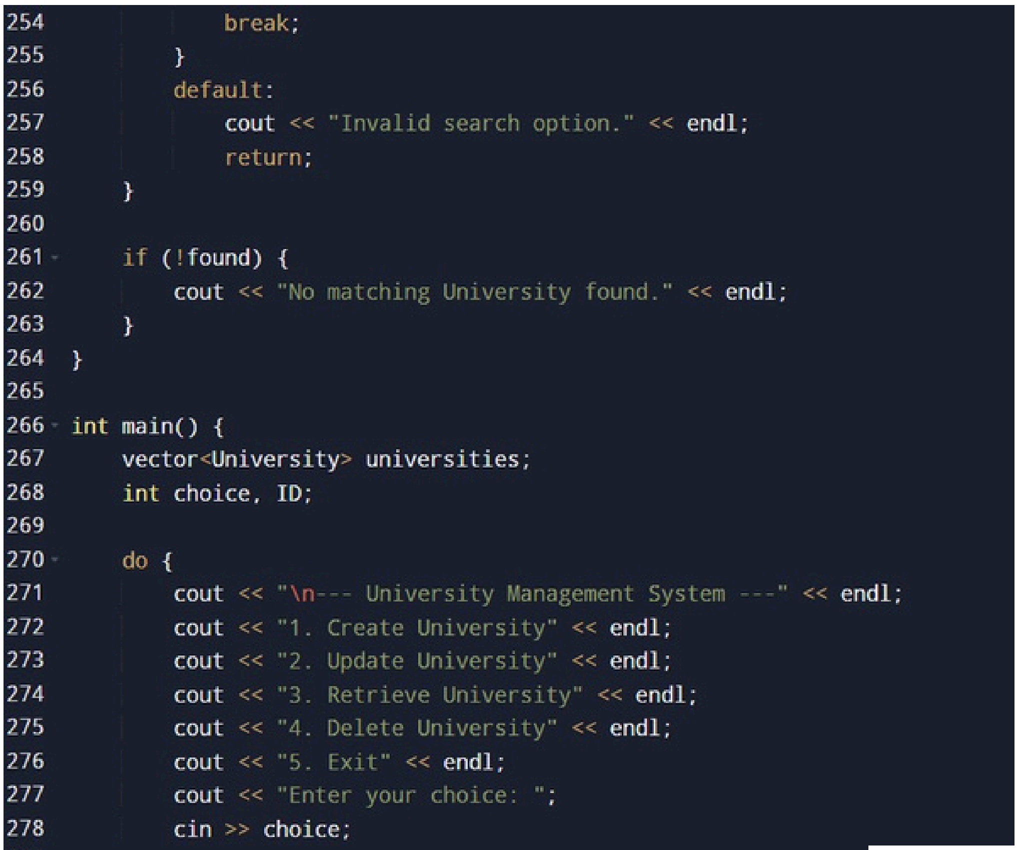


```
                cout << "Match found" << endl;
                sort_by_code[index].first.displayInfo();
                found = true;
            }
            break;
        }
        case 3: {
            string name;
            cout << "Enter University Name: ";
            cin.ignore();
            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);
```

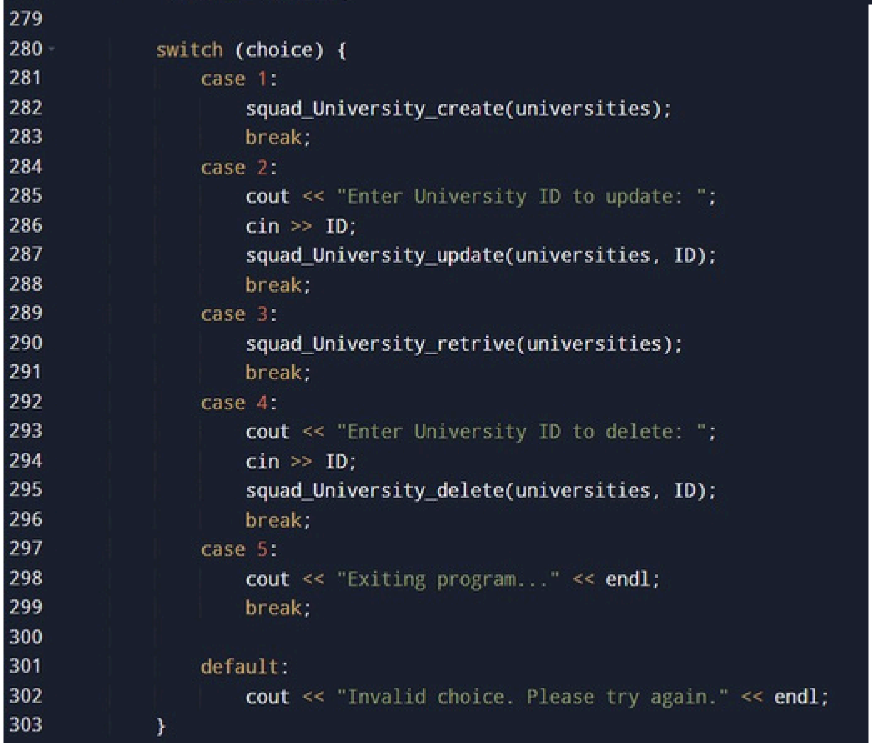


```
            }
            break;
        }
        case 5: {
            string mail;
            cout << "Enter University Email: ";
            cin >> mail;
            index = squad_university_binarySearch(sort_by_email, mail);
            if (index != -1) {
                cout << "Match found" << 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;
            }
```

Source Code

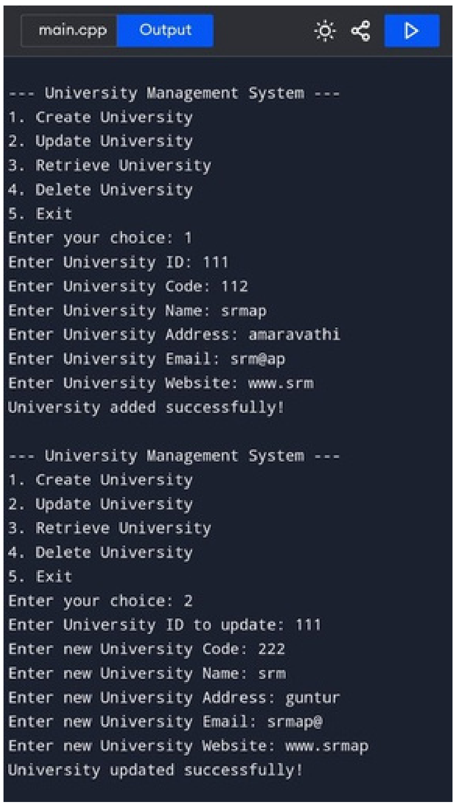


```cpp
            break;
        }
        default:
            cout << "Invalid search option." << endl;
            return;
    }

    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;
```

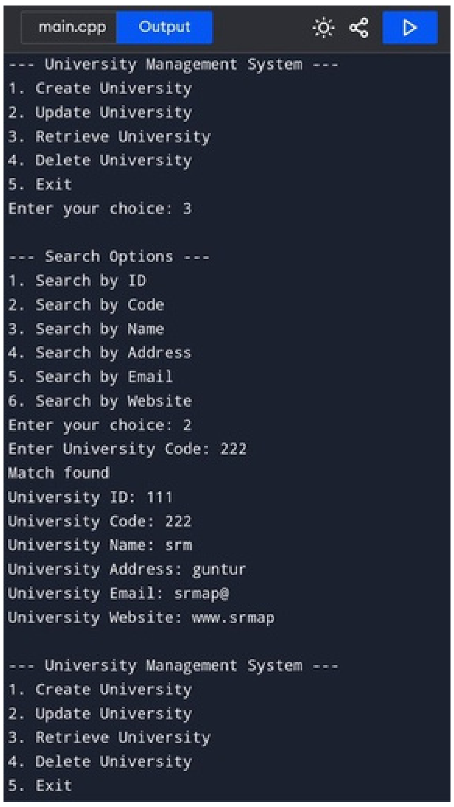


```cpp

        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:
                squad_University_retrive(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;
        }
```

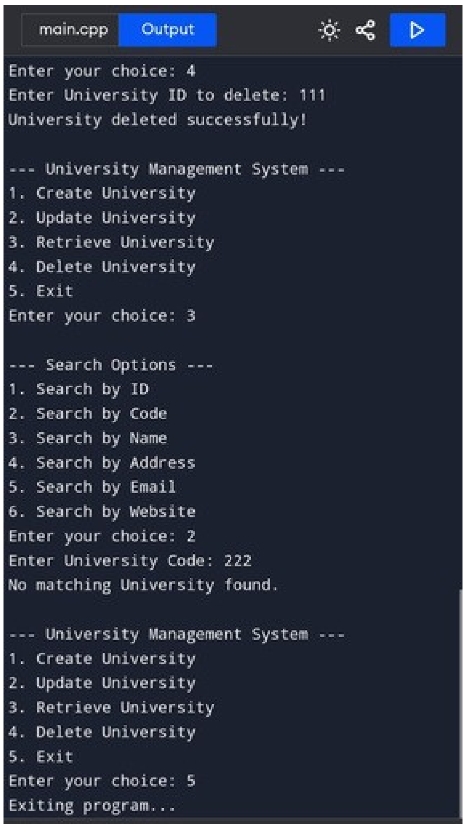
Screen Shots



```
--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
Enter your choice: 1
Enter University ID: 111
Enter University Code: 112
Enter University Name: srmap
Enter University Address: amaravathi
Enter University Email: srm@ap
Enter University Website: www.srm
University added successfully!

--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
Enter your choice: 2
Enter University ID to update: 111
Enter new University Code: 222
Enter new University Name: srm
Enter new University Address: guntur
Enter new University Email: srmap@
Enter new University Website: www.srmap
University updated successfully!
```



```
--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
Enter your choice: 3

--- Search Options ---
1. Search by ID
2. Search by Code
3. Search by Name
4. Search by Address
5. Search by Email
6. Search by Website
Enter your choice: 2
Enter University Code: 222
Match found
University ID: 111
University Code: 222
University Name: srm
University Address: guntur
University Email: srmap@
University Website: www.srmap

--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
```



```
Enter your choice: 4
Enter University ID to delete: 111
University deleted successfully!

--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
Enter your choice: 3

--- Search Options ---
1. Search by ID
2. Search by Code
3. Search by Name
4. Search by Address
5. Search by Email
6. Search by Website
Enter your choice: 2
Enter University Code: 222
No matching University found.

--- University Management System ---
1. Create University
2. Update University
3. Retrieve University
4. Delete University
5. Exit
Enter your choice: 5
Exiting program...
```

Conclusion

This C++ program serves as a simple university management system, allowing users to create, update, retrieve, and delete university records. It uses sorting and binary search to manage and efficiently retrieve data by various attributes (like ID, name, code, etc.), providing a menu-driven interface for easy user interaction. This program demonstrates basic CRUD operations and efficient data handling in C++