OBE Implementation

Module-1: UNIVERSITY SETTINGS

Submitted By:

AP23110011022 | NeeliHarshitha
AP23110011021 | V Himaja
AP23110011012 | K Mohitha
AP23110011071 I K Jaswanth
AP23110011054 | S Harshavardhan

Btech cse(O)
semester: 3

Introduction to Project

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:

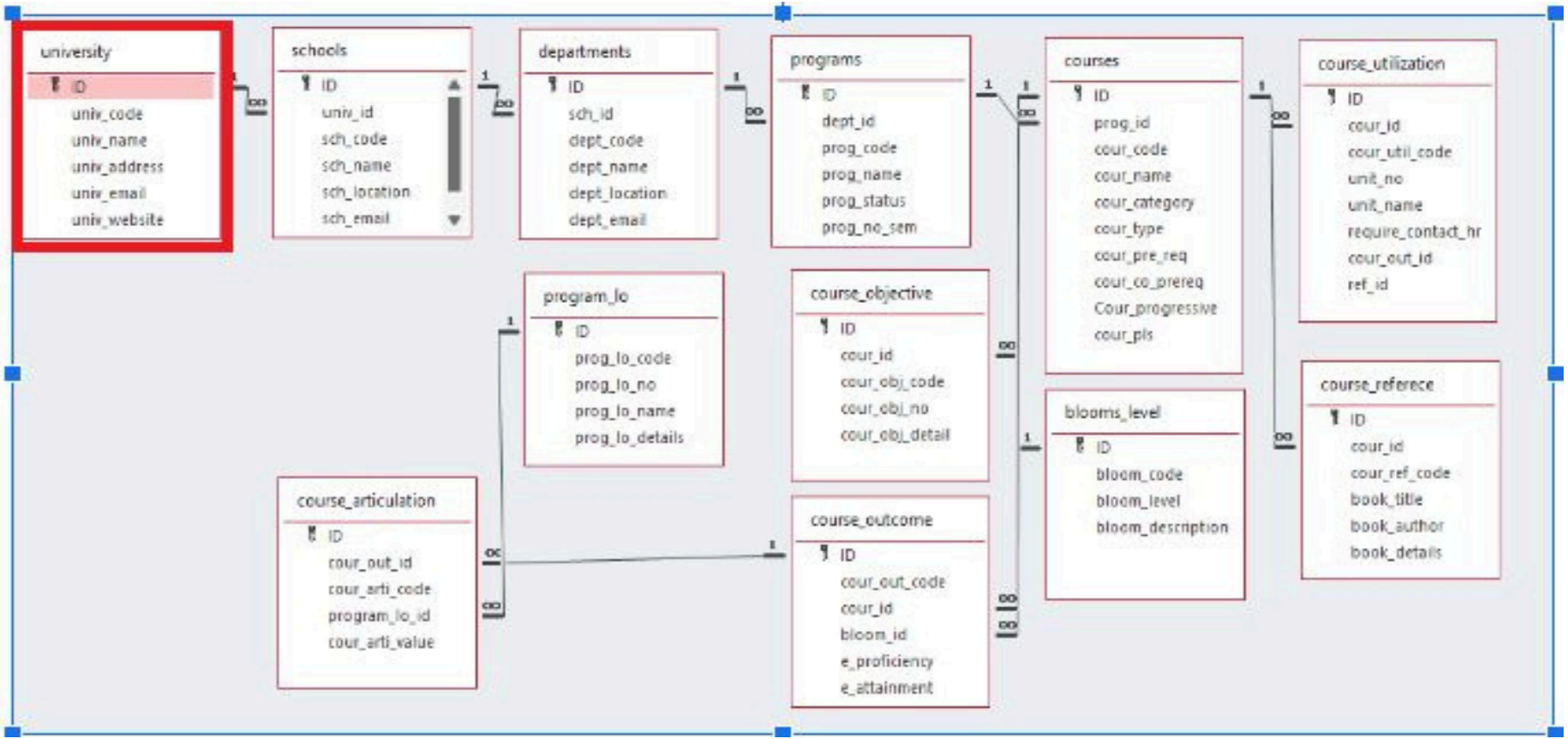
- University ID, code, name, address, email, and website.

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.

Architecture Diagram[*highlight your module as shown]



Module Description : University Setting

- This module is used to create,Update,Retrieve,Delete(hereafter known as CURD) details of the module and storing the details in the text file.you have to provide option for searching and sorting of fields mentioned below according to algorithms given for you

University Setting:Field/table details

Field Name	Data type
id	integer
univ_code	String
univ_name	String
univ_address	String
univ_email	String
univ_website	String

University Setting:Programming Details

- 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
  -

University Setting:Programming Details

- Comparison(both searching and Sorting)
  - 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))

University : Sorting Algorithm used

- 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.

University : Sorting Algorithm used

- 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	$n-1$

University : Time Complexity of Sorting Algorithm

Sl.No	sort Algorithm Name	Time complexity
1	insertion sort	O(n^2)
2	insertion sort	O(log n)

University : Searching Algorithm used

- Algorithm Name: BINARY SEARCH

Binary Search Algorithm

```
Binary-Search(A,x,l,r)   //intial call parameters are Binary-Search
                                (A,1,n,x)
1.  if l > r then
2.      return –1; //Not found
3.  end if
4.  m := [(l + r )/2];
5.  if A[m] = x then
6.      return m
7.  else if x < A[m] then
8.      return Binary-Search(A, x, l ,m – 1))
9.  else
10.     return Binary-Search (A, x,m + 1, r ))
11. end if
```

University : Searching Algorithm used

- Algorithm Name: LINEAR SEARCH

Algorithm of Linear Search:

Linear Search (Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit

University : Time Complexity of Searching Algorithm

Sl.No	search Algorithm Name	Time complexity
1	binary search	O(log n)
2	linear search	O(n)

Sample Source Code[*Depict the routine of searching,Sorting,CRUD and Storage options]

```cpp
#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;
    }
};
```

```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};
    }
}

```

```
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;
}

```

```
102 void squad_University_create(vector<University> &universities) {
103     int ID;
104     string univ_code, univ_name, univ_address, univ_mail, univ_website;
105     cout << "Enter University ID: ";
106     cin >> ID;
107     cout << "Enter University Code: ";
108     cin >> univ_code;
109     cout << "Enter University Name: ";
110     cin.ignore();
111     getline(cin, univ_name);
112     cout << "Enter University Address: ";
113     getline(cin, univ_address);
114     cout << "Enter University Email: ";
115     getline(cin, univ_mail);
116     cout << "Enter University Website: ";
117     getline(cin, univ_website);
118 
119     universities.emplace_back(ID, univ_code, univ_name, univ_address, univ_mail,
        univ_website);
120 
121     updateSortedVectors(universities);
122     cout << "University added successfully!" << endl;
123 }
124 
```

```
80  void updateSortedVectors(const vector<University> &universities) {
81      sort_by_code.clear();
82      sort_by_name.clear();
83      sort_by_address.clear();
84      sort_by_email.clear();
85      sort_by_web.clear();
86
87      for (const auto &univ : universities) {
88          sort_by_code.push_back({univ, univ.getUnivCode()});
89          sort_by_name.push_back({univ, univ.getUnivName()});
90          sort_by_address.push_back({univ, univ.getUnivAddress()});
91          sort_by_email.push_back({univ, univ.getUnivMail()});
92          sort_by_web.push_back({univ, univ.getUnivWebsite()});
93      }
94
95      squad_university_insertionSort(sort_by_code);
96      squad_university_insertionSort(sort_by_name);
97      squad_university_insertionSort(sort_by_address);
98      squad_university_insertionSort(sort_by_email);
99      squad_university_insertionSort(sort_by_web);
100 }
101
```

```
        }
    }
    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;
```

```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;
```

```
200             cout << "Match found" << endl;
201             sort_by_code[index].first.displayInfo();
202             found = true;
203         }
204         break;
205     }
206     case 3: {
207         string name;
208         cout << "Enter University Name: ";
209         cin.ignore();
210         getline(cin, name);
211         index = squad_university_binarySearch(sort_by_name, name);
212         if (index != -1) {
213             cout << "Match found" << endl;
214             sort_by_name[index].first.displayInfo();
215             found = true;
216         }
217         break;
218     }
219     case 4: {
220         string address;
221         cout << "Enter University Address: ";
222         cin.ignore();
223         getline(cin, address);
224         index = squad_university_binarySearch(sort_by_address, address);
```

```
175     cout << "6. Search by Website" << endl;
176     cout << "Enter your choice: ";
177     cin >> searchChoice;
178 
179     bool found = false;
180     int index;
181     switch (searchChoice) {
182         case 1: {
183             int ID;
184             cout << "Enter University ID: ";
185             cin >> ID;
186             for (const auto &univ : universities) {
187                 if (univ.getID() == ID) {
188                     univ.displayInfo();
189                     found = true;
190                 }
191             }
192             break;
193         }
194         case 2: {
195             string code;
196             cout << "Enter University Code: ";
197             cin >> code;
198             index = squad_university_binarySearch(sort_by_code, code);
199             if (index != -1) {
```

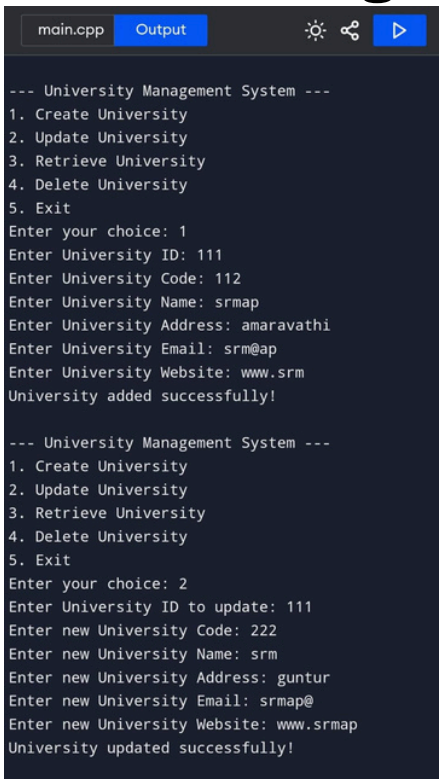
```
229             }
230             break;
231         }
232         case 5: {
233             string mail;
234             cout << "Enter University Email: ";
235             cin >> mail;
236             index = squad_university_binarySearch(sort_by_email, mail);
237             if (index != -1) {
238                 cout << "Match found" << endl;
239                 sort_by_email[index].first.displayInfo();
240                 found = true;
241             }
242             break;
243         }
244         case 6: {
245             string website;
246             cout << "Enter University Website: ";
247             cin >> website;
248             index = squad_university_binarySearch(sort_by_web, website);
249             if (index != -1) {
250                 cout << "Match found" << endl;
251                 sort_by_web[index].first.displayInfo();
252                 found = true;
253             }
```

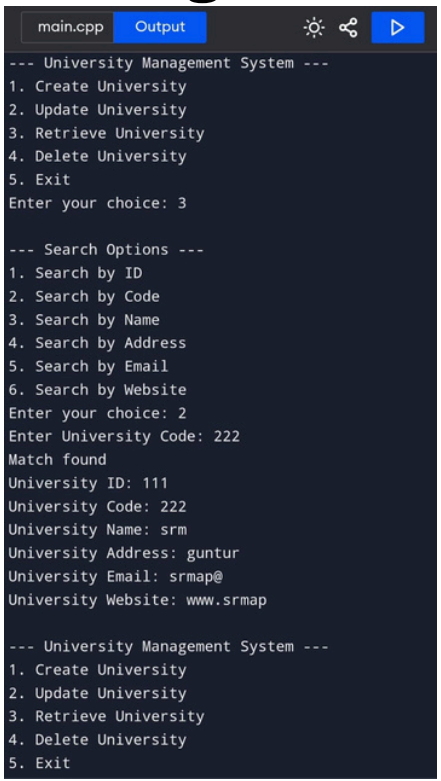
```
254             break;
255         }
256         default:
257             cout << "Invalid search option." << endl;
258             return;
259     }
260 
261     if (!found) {
262         cout << "No matching University found." << endl;
263     }
264 }
265 
266 int main() {
267     vector<University> universities;
268     int choice, ID;
269 
270     do {
271         cout << "\n--- University Management System ---" << endl;
272         cout << "1. Create University" << endl;
273         cout << "2. Update University" << endl;
274         cout << "3. Retrieve University" << endl;
275         cout << "4. Delete University" << endl;
276         cout << "5. Exit" << endl;
277         cout << "Enter your choice: ";
278         cin >> choice;
```

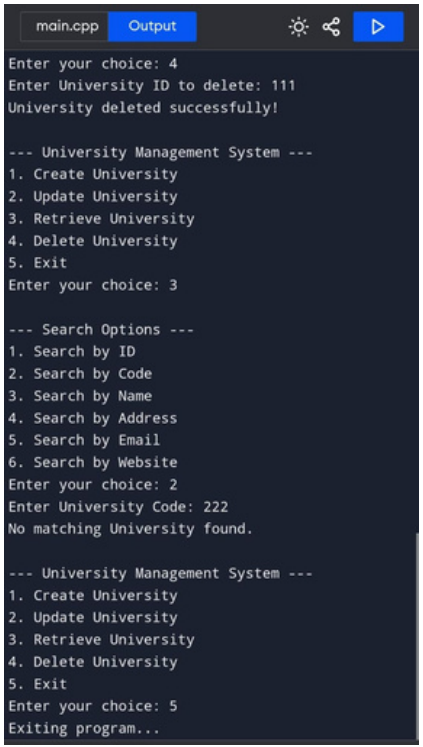
```
279
280         switch (choice) {
281             case 1:
282                 squad_University_create(universities);
283                 break;
284             case 2:
285                 cout << "Enter University ID to update: ";
286                 cin >> ID;
287                 squad_University_update(universities, ID);
288                 break;
289             case 3:
290                 squad_University_retrive(universities);
291                 break;
292             case 4:
293                 cout << "Enter University ID to delete: ";
294                 cin >> ID;
295                 squad_University_delete(universities, ID);
296                 break;
297             case 5:
298                 cout << "Exiting program..." << endl;
299                 break;
300 
301             default:
302                 cout << "Invalid choice. Please try again." << endl;
303         }
```

```
304         } while (choice != 5);
305
306         return 0;
307     }
308
```

Sample Screen Shots[*Screen shot of CRUD,Sorting,Searching,Comparison(both sorting and Searching and Storage)]







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++.

SRM UNIVERSITY AP
Amaravati

Thank You