DATA STRUCTURES AND ALGORITHMS

Complex Computing Problem

Bahria University
Discovering Knowledge

DR. SAMABIA TEHSEEN

BS(AI) 3A

M. USMAN ABUBAKAR : 02-136221-031

HASSAM AZAM SIDDIQUI : 02-136221-035

Why Doubly Linked List?

In the given scenario where we are managing a system for university which must have a large amount of students getting enrolled, we need am efficient system which uses less memory and less time as well in order to have a good system. A Doubly Linked List would be suitable in this case as it allows efficient insertion and deletion of nodes at any position and at any time using dynamic memory allocation which is important in enrollment system. It also allows us to traverse in both directions. It also allows easy implementation for sorting based on different criteria such as name or roll number which is common requirement in enrollment system.

An array might not be as efficient as doubly because its size is fixed once it's allocated, in order to dynamically resize it we would require allocating a new array with larger or smaller size and copy all of the existing elements into the new array and then add new element to it which seems time consuming and inefficient. When inserting or deleting an element in array, it requires to shift all elements which will not be efficient for large arrays. In contrast to this, doubly linked list provides easy implementation in these scenarios as we just have to change links when deleting or adding an element.

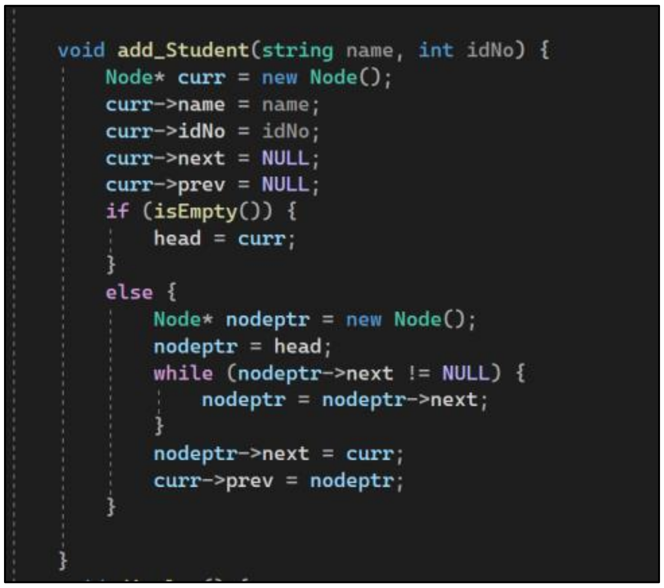
But arrays do have some pros over doubly like memory efficiency and faster access of elements. It uses less memory because each element of an array only stores its own info while in doubly each element stores its own data with the info about previous and next node.

Doubly Linked List for Enrollment System:

As we have used doubly linked list to implement the enrollment system. We can fulfill the required tasks of enrollment system efficiently. We have to perform tasks like adding new students who get enrolled, remove students from anywhere in between the list who leave university, and sort students using name and id number as well. Let's see some implementation of these:

Adding Students:

As shown in **Figure 1.1** We first check if the list is empty or not, if it is then the new node added will be the head node, else we will traverse the list to reach the last node and place it in the end of the list. As the temporary node is on the last node, it will connect its 'next' to new node and new node will connect its 'prev' to temporary node which is the second last node of array now.

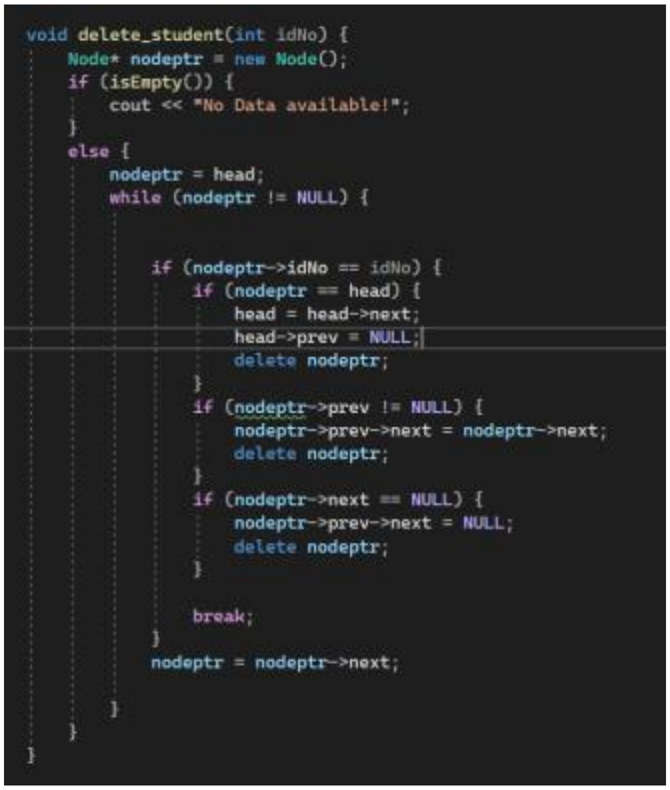


```
void add_Student(string name, int idNo) {
    Node* curr = new Node();
    curr->name = name;
    curr->idNo = idNo;
    curr->next = NULL;
    curr->prev = NULL;
    if (isEmpty()) {
        head = curr;
    }
    else {
        Node* nodeptr = new Node();
        nodeptr = head;
        while (nodeptr->next != NULL) {
            nodeptr = nodeptr->next;
        }
        nodeptr->next = curr;
        curr->prev = nodeptr;
    }

}
```

Figure 1.1 Add Students

Removing Students:

- Check if list is empty by using IsEmpty(), if yes then nothing to remove.
- If not empty then traverse through list and compare the input id by each node's id. There will be some special cases here:
- If the node to be deleted is head node then head node will be changed to 2nd node which is 'next' of temporary node, and set 'prev' of 2nd node to null as it's the head node now as seen in **Figure 1.2**.
- If the node to be deleted is in the middle of the list, then the 'prev' of node to be deleted will be connected to the 'next' of the node to be deleted. Then delete the node.



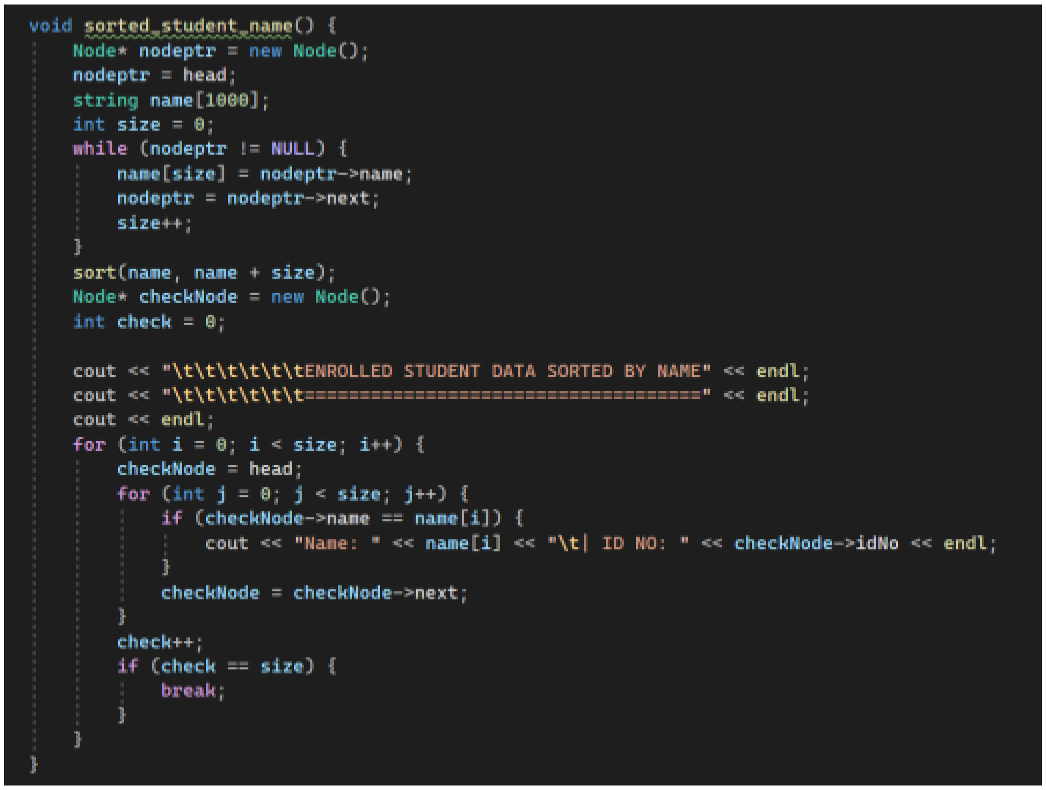
```
void delete_student(int idNo) {
    Node* nodeptr = new Node();
    if (isEmpty()) {
        cout << "No Data available!";
    }
    else {
        nodeptr = head;
        while (nodeptr != NULL) {


            if (nodeptr->idNo == idNo) {
                if (nodeptr == head) {
                    head = head->next;
                    head->prev = NULL;
                    delete nodeptr;
                }
                if (nodeptr->prev != NULL) {
                    nodeptr->prev->next = nodeptr->next;
                    delete nodeptr;
                }
                if (nodeptr->next == NULL) {
                    nodeptr->prev->next = NULL;
                    delete nodeptr;
                }

                break;
            }
            nodeptr = nodeptr->next;

        }
    }
}
```

Figure 1.2 Remove Students

- if the node to be deleted is the last node in list
  then it's previous nodes 'next' will be set to null as it will be the last node of the list.

Display List sorted by Names:

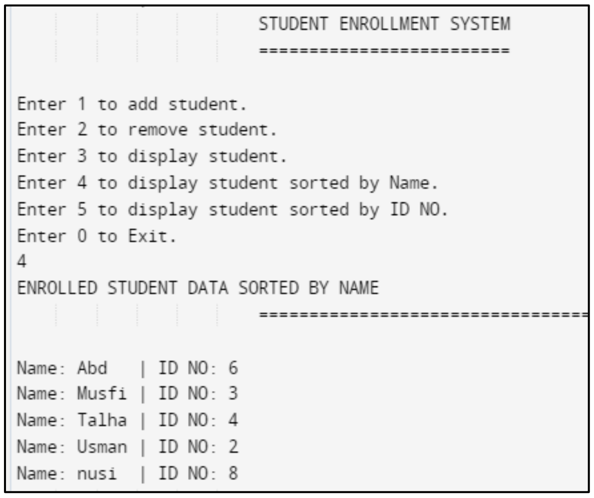
Here first we convert the linked list to array as shown in **Figure1.3**, then take the size of array using while loop, then use 'sort()' function which is a built-in function in C++ STD library 'Algorithm'. Now the array is sorted. Now we make a temporary node and compare each name in list with that first name in array one by one, eventually the node will be printed which will compare correctly with the sorted array name. The process will go on until the array size is reached.



```cpp
void sorted_student_name() {
    Node* nodeptr = new Node();
    nodeptr = head;
    string name[1000];
    int size = 0;
    while (nodeptr != NULL) {
        name[size] = nodeptr->name;
        nodeptr = nodeptr->next;
        size++;
    }
    sort(name, name + size);
    Node* checkNode = new Node();
    int check = 0;

    cout << "\t\t\t\t\t\tENROLLED STUDENT DATA SORTED BY NAME" << endl;
    cout << "\t\t\t\t\t\t====================================" << endl;
    cout << endl;
    for (int i = 0; i < size; i++) {
        checkNode = head;
        for (int j = 0; j < size; j++) {
            if (checkNode->name == name[i]) {
                cout << "Name: " << name[i] << "\t| ID NO: " << checkNode->idNo << endl;
            }
            checkNode = checkNode->next;
        }
        check++;
        if (check == size) {
            break;
        }
    }
}
```

Figure 1.3 Display info sorted by Names

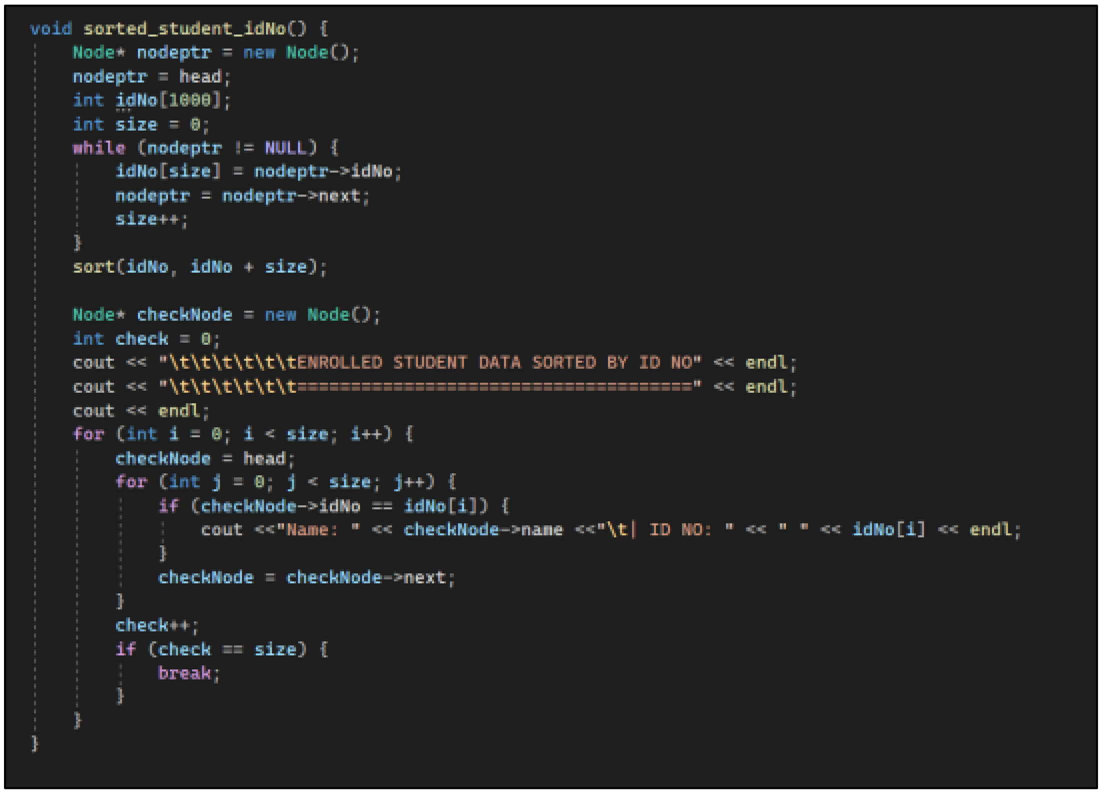
This gives us the desired result, all names sorted with ease. Output can be seen below:



```
                         STUDENT ENROLLMENT SYSTEM
                         =========================

Enter 1 to add student.
Enter 2 to remove student.
Enter 3 to display student.
Enter 4 to display student sorted by Name.
Enter 5 to display student sorted by ID NO.
Enter 0 to Exit.
4
ENROLLED STUDENT DATA SORTED BY NAME
                         ==================================

Name: Abd    | ID NO: 6
Name: Musfi | ID NO: 3
Name: Talha | ID NO: 4
Name: Usman | ID NO: 2
Name: nusi   | ID NO: 8
```

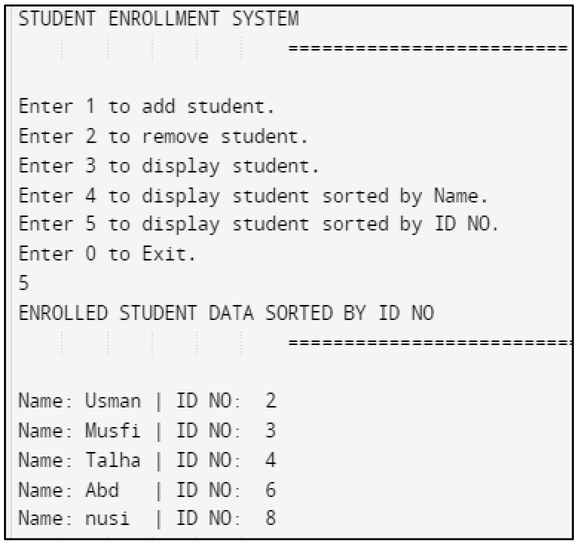
Display List sorted by ID:

Here we do the same steps again. First we convert the list into array, then get the size of list and sort the list with 'sort' built-in function. Then we compare every node with first id in array, then print the info of that node, then move to second cell of array and compare it with every node again. This will go on until the array size is reached. The code can be seen in **Figure 1.4** with output in **Figure 1.5.**



```cpp
void sorted_student_idNo() {
    Node* nodeptr = new Node();
    nodeptr = head;
    int idNo[1000];
    int size = 0;
    while (nodeptr != NULL) {
        idNo[size] = nodeptr->idNo;
        nodeptr = nodeptr->next;
        size++;
    }
    sort(idNo, idNo + size);

    Node* checkNode = new Node();
    int check = 0;
    cout << "\t\t\t\t\t\tENROLLED STUDENT DATA SORTED BY ID NO" << endl;
    cout << "\t\t\t\t\t\t=====================================" << endl;
    cout << endl;
    for (int i = 0; i < size; i++) {
        checkNode = head;
        for (int j = 0; j < size; j++) {
            if (checkNode->idNo == idNo[i]) {
                cout <<"Name: " << checkNode->name <<"\t| ID NO: " << " " << idNo[i] << endl;
            }
            checkNode = checkNode->next;
        }
        check++;
        if (check == size) {
            break;
        }
    }
}
```

Figure 1.4 Display list sorted by ID



```
STUDENT ENROLLMENT SYSTEM
                         ==========================


Enter 1 to add student.
Enter 2 to remove student.
Enter 3 to display student.
Enter 4 to display student sorted by Name.
Enter 5 to display student sorted by ID NO.
Enter 0 to Exit.
5
ENROLLED STUDENT DATA SORTED BY ID NO
                         ==========================


Name: Usman | ID NO:  2
Name: Musfi | ID NO:  3
Name: Talha | ID NO:  4
Name: Abd   | ID NO:  6
Name: nusi  | ID NO:  8
```

Figure 1.5 Output of sorted with ID

SYNTAX:

```
#include <iostream>
#include <algorithm>

using namespace std;

struct DoubleyLL {
    string name;
    int idNo;
    DoubleyLL* prev;
    DoubleyLL* next;
};

class Student_Enrollment_DLL {
private:
    DoubleyLL* head;

public:
    Student_Enrollment_DLL() {
        head = NULL;
    }

    void add_Student(string name, int idNo) {
        DoubleyLL* curr = new DoubleyLL();
        curr->name = name;
        curr->idNo = idNo;
        curr->next = NULL;
        curr->prev = NULL;
        if (isEmpty()) {
            head = curr;
        }
        else {
            DoubleyLL* nodeptr = new DoubleyLL();
            nodeptr = head;
            while (nodeptr->next != NULL) {
                nodeptr = nodeptr->next;
            }
            nodeptr->next = curr;
            curr->prev = nodeptr;
        }

    }
    void display() {
        if (isEmpty()) {
            cout << "no data!" << endl;
        }
        else {
```

```
        DoubleyLL* nodeptr = new DoubleyLL();
        nodeptr = head;
        cout << "\t\t\t\t\t\tENROLLED STUDENT DATA" << endl;
        cout << "\t\t\t\t\t\t====================" << endl;
        cout << endl;

        while (nodeptr != NULL) {
            cout << "Name: " << nodeptr->name << "\t|" << "ID NO: " << nodeptr->idNo << endl;
            nodeptr = nodeptr->next;
        }
    }
}

void sorted_student_name() {
    DoubleyLL* nodeptr = new DoubleyLL();
    nodeptr = head;
    string name[1000];
    int size = 0;
    while (nodeptr != NULL) {
        name[size] = nodeptr->name;
        nodeptr = nodeptr->next;
        size++;
    }
    sort(name, name + size);
    DoubleyLL* checkNode = new DoubleyLL();
    int check = 0;

    cout << "\t\t\t\t\t\tENROLLED STUDENT DATA SORTED BY NAME" << endl;
    cout << "\t\t\t\t\t\t==================================" << endl;
    cout << endl;
    for (int i = 0; i < size; i++) {
        checkNode = head;
        for (int j = 0; j < size; j++) {
            if (checkNode->name == name[i]) {
                cout << "Name: " << name[i] << "\t| ID NO: " << checkNode->idNo << endl;
            }
            checkNode = checkNode->next;
        }
        check++;
        if (check == size) {
            break;
        }
    }
}

void sorted_student_idNo() {
    DoubleyLL* nodeptr = new DoubleyLL();
    nodeptr = head;
    int idNo[1000];
    int size = 0;
    while (nodeptr != NULL) {
        idNo[size] = nodeptr->idNo;
        nodeptr = nodeptr->next;
        size++;
    }
    sort(idNo, idNo + size);

    DoubleyLL* checkNode = new DoubleyLL();
    int check = 0;
    cout << "\t\t\t\t\t\tENROLLED STUDENT DATA SORTED BY ID NO" << endl;
    cout << "\t\t\t\t\t\t=====================================" << endl;
    cout << endl;
    for (int i = 0; i < size; i++) {
        checkNode = head;
        for (int j = 0; j < size; j++) {
            if (checkNode->idNo == idNo[i]) {
                cout <<"Name: " << checkNode->name <<"\t| ID NO: " << " " << idNo[i] << endl;
            }
            checkNode = checkNode->next;
        }
        check++;
        if (check == size) {
            break;
        }
    }
}


void delete_student(int idNo) {
    DoubleyLL* nodeptr = new DoubleyLL();
    if (isEmpty()) {
        cout << "No Data available!";
    }
    else {
        nodeptr = head;
        while (nodeptr != NULL) {


            if (nodeptr->idNo == idNo) {
                if (nodeptr == head) {
                    head = head->next;
                    head->prev = NULL;
                    delete nodeptr;
                }
                if (nodeptr->prev != NULL) {
                    nodeptr->prev->next = nodeptr->next;
```

```
                        delete nodeptr;
                    }
                    if (nodeptr->next == NULL) {
                        nodeptr->prev->next = NULL;
                        delete nodeptr;
                    }

                    break;
                }
                nodeptr = nodeptr->next;

            }
        }
    }

    bool isEmpty() const {
        return head == NULL;
    }
};

int main() {
    Student_Enrollment_DLL std;
    int choice;

    string name;
    int idNo, loop;

    bool check = true;
    while (check) {
        cout << "\t\t\t\t\t\tSTUDENT ENROLLMENT SYSTEM" << endl;
        cout << "\t\t\t\t\t\t========================" << endl;
        cout << endl;
        cout << "Enter 1 to add student." << endl;
        cout << "Enter 2 to remove student." << endl;
        cout << "Enter 3 to display student." << endl;
        cout << "Enter 4 to display student sorted by Name." << endl;
        cout << "Enter 5 to display student sorted by ID NO." << endl;
        cout << "Enter 0 to Exit." << endl;
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter number of students you wanna enroll:";
            cin >> loop;
            for (int i = 0; i < loop; i++) {
                cout << "Enter Name of the student:";
                cin >> name;
                cout << "Enter ID NO:";
            cin >> idNo;
            std.add_Student(name, idNo);
            cout << endl;
          }
          break;
        case 2:
          std.display();

          cout << endl;

          cout << "Enter ID NO:";
          cin >> idNo;
          std.delete_student(idNo);

          break;
        case 3:
          std.display();
          break;
        case 4:
          std.sorted_student_name();
          break;
        case 5:
          std.sorted_student_idNo();
          break;

        case 0:
          check = false;
          break;
        default:
          cout << "enter valid choice!" << endl;
          break;
        }
      }
      return 0;
    }
```

<u>END</u>