**Data Structure Practical Exam**

B.Sc.(H) Computer Science (3rd Semester)

Practical Exam (4 Hrs.)

Dated: 19 Nov, 2020 Time: 11 a.m.

Submitted By –

Anshul Verma

Roll No – 19/78065

**Q1. Create two singly linked lists and merge them in descending order.**

**Code:**

#include <iostream>

using namespace std;

struct node

{

    int data;

    struct node \*next;

};

/\* SinglyLinkedList Class \*/

class SinglyLinkedList

{

    public:

    struct node \*head, \*temp, \*ptr, \*tail;

    SinglyLinkedList()

    {

        head = NULL;

    }

    ~SinglyLinkedList()

    {

        ptr = head;

        while (ptr != NULL)

        {

            temp = ptr->next;

            delete ptr;

            ptr = temp;

        }

    }

    bool is\_empty()

    {

        if (head == NULL)

            return true;

        else

            return false;

    }

    void insert\_at\_end(int n)

    {

        temp = new node;

        temp->data = n;

        temp->next = NULL;

        if (is\_empty())

        {

            head = temp;

            tail = head;

        }

        else

        {

            tail->next = temp;

            tail = temp;

        }

    }

    void display()

    {

        cout << "\n\t";

        if (is\_empty())

            cout << "Linked List is empty.";

        ptr = head;

        while (ptr != NULL)

        {

            if (ptr == head)

                cout << ptr->data;

            else

                cout << " --> " << ptr->data;

            ptr = ptr->next;

        }

        cout << endl;

    }

};

// Function to merge two lists

void mergeLists(SinglyLinkedList &list1, SinglyLinkedList &list2)

{

    cout << "\nFirst List: ";

    list1.display();

    cout << "\nSecond List: ";

    list2.display();

    // Checking for NULL conditions

    if(list1.head != NULL || list2.head != NULL)

    {

        if(list1.head == NULL && list2.head != NULL)

            list1 = list2;

        else if(list1.head != NULL && list2.head != NULL)

        {

            list1.tail->next = list2.head;

            list1.tail = list2.tail;

        }

        // Sort the linked list1 using bubble sort

        node\* curr = list1.head;

        node\* temp = list1.head;

        while (curr->next != NULL) {

            temp = curr->next;

            while (temp != NULL) {

                if (temp->data > curr->data) {

                    int t = temp->data;

                    temp->data = curr->data;

                    curr->data = t;

                }

                temp = temp->next;

            }

            curr = curr->next;

        }

    }

    cout << "\nMerged List: ";

    list1.display();

}

// Main function

int main()

{

    SinglyLinkedList s1, s2;

    s1.insert\_at\_end(20);

    s1.insert\_at\_end(10);

    s1.insert\_at\_end(15);

    s1.insert\_at\_end(30);

    s1.insert\_at\_end(50);

    s2.insert\_at\_end(60);

    s2.insert\_at\_end(80);

    s2.insert\_at\_end(100);

    s2.insert\_at\_end(45);

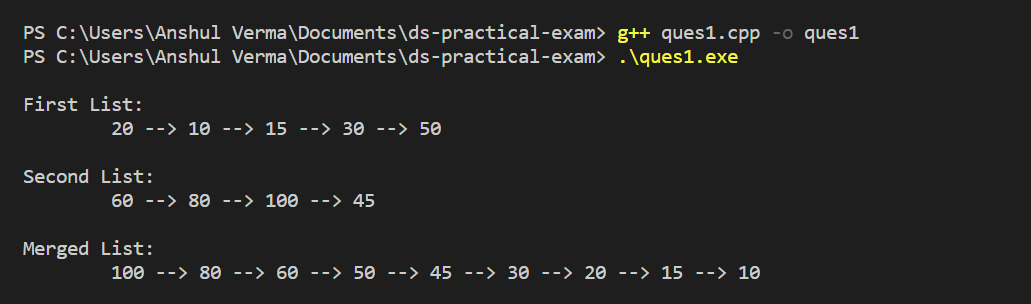
    mergeLists(s1, s2);

    cout << endl;

    return 0;

}

**Output:**

****

**Q2. Find largest element in an array using recursion.**

**Code:**

#include <iostream>

using namespace std;

// findLargest recursive function

 int findLargest(int arr[], int pos, int largest)

{

    if (pos == 0)

        return largest;

    if (pos > 0)

    {

        if (arr[pos] > largest)

        {

            largest = arr[pos];

        }

        return findLargest(arr, pos - 1, largest);

    }

    else

        return -1;

}

// Main Function

int main()

{

    int arr[] = {10, 14, 44, 6, -50, 12, 20};

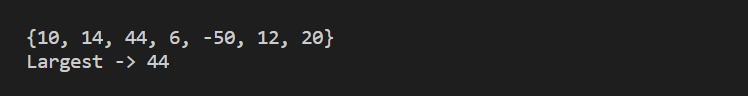
    cout << "\n{10, 14, 44, 6, -50, 12, 20}" << endl;

    cout << "Largest -> " << findLargest(arr, 7-1, arr[0]) << "\n\n";

    return 0;

}

**Output:**

****