



NUMBER : NAME :	EVALUATION				
	1[.....]	2[.....]	3[.....]	4[.....]	SUM[.....]
SIGNATURE :					
Faculty of Engineering Exam Directives must be followed. Questions are related to 1,4,12 of the Program Learning Outcomes					

```
void CircularlyLinkedList::removeOrdered(const
string& e, const int& i, CircularlyNode* previous)
{
    if (empty())
    {
        cout <<"List is empty !"<< endl; return;
    };

    if (cursor->next == cursor)
    {
        delete cursor; cursor = NULL; return;
    }

    if (previous->next != cursor)
    {
        if ((e.compare(previous->next->elem) == 0) &&
            (previous->next->score == i))
        {
            CircularlyNode* temp = previous->next;
            previous->next = previous->next->next;
            delete temp;
            return;
        }

        removeOrdered(e, i, previous->next);
    }
    else
    {
        if ((e.compare(previous->next->elem) == 0) &&
            (previous->next->score == i))
        {
            CircularlyNode* temp = previous->next;
            previous->next = previous->next->next;
            delete temp;
            cursor = previous;
        }
        else cout << e << " is not found" << endl;
    }
}

int main()
{
    CircularlyLinkedList list;

    list.insertOrdered("Paul", 720);
    list.insertOrdered("Rose", 590);
    list.insertOrdered("Anna", 660);
    list.insertOrdered("Mike", 1105);
    list.insertOrdered("Rob", 750);
    list.insertOrdered("Jack", 510);
    list.insertOrdered("Jill", 740);

    list.removeOrdered("Mike", 1105, list.cursor);
}
```

1. How many times does the function removeOrdered() call itself recursively when removing **Mike**? (30P)

```

void BinaryTree::eulerLike(TreeNode* v) const
{
    if (v->left != NULL)
    {
        eulerLike(v->left);
        cout << v->elem;
    }

    if (v->right != NULL)
    {
        eulerLike(v->right);
    }

    cout << v->elem;
}

int main()
{
    BinaryTree binaryTree;
    binaryTree.addNode(binaryTree.root, 7);
    binaryTree.addNode(binaryTree.root, 3);
    binaryTree.addNode(binaryTree.root, 11);
    binaryTree.addNode(binaryTree.root, 1);
    binaryTree.addNode(binaryTree.root, 5);
    binaryTree.addNode(binaryTree.root, 9);
    binaryTree.addNode(binaryTree.root, 13);

    binaryTree.eulerLike(binaryTree.root);
}

```

2. What is the output of the program above? (20P)

--	--	--	--	--	--	--	--	--	--

3. How many times does the function `eulerLike()` call itself recursively? (20P)

--

```

SinglyLinkedList SinglyLinkedList::
sameLists(SinglyLinkedList list2)
{
    SinglyLinkedList sameList;

    SinglyNode* plist1 = head;
    SinglyNode* plist2 = list2.head;

    while ((plist1 != NULL) && (plist2 != NULL))
    {
        if (plist1->score < plist2->score)
        {
            plist1 = plist1->next;
            if (plist1 == NULL) break;
        }

        if (plist1->score > plist2->score)
        {
            plist2 = plist2->next;
            if (plist2 == NULL) break;
        }

        if (plist1->score == plist2->score)
        {
            sameList.addBack(plist1->score);
            plist1 = plist1->next;
            plist2 = plist2->next;
        }
    }

    return sameList;
}

int main()
{
    SinglyLinkedList list1;
    list1.addBack(1);
    list1.addBack(2);
    list1.addBack(3);
    list1.addBack(4);
    list1.addBack(5);

    SinglyLinkedList list2;
    list2.addBack(2);
    list2.addBack(4);
    list2.addBack(6);
    list2.addBack(8);
    list2.addBack(10);

    SinglyLinkedList sameList =
        list1.sameLists(list2);
    sameList.print();
}

```

4. What is the output of the program above? (30P)

--