```
#include <string>
#include <iostream>
using namespace std;

struct SinglyNode
{
	string elem;                                                        // element name
	int score;                                                                // element score
	SinglyNode* next;                                                   // next item in the list
};

struct SinglyLinkedList
{
	SinglyNode* head;                                                   // pointer to the head of list

	SinglyLinkedList();                                                 // empty list constructor
	~SinglyLinkedList();                                                // destructor

	bool empty() const;                                                 // is list empty?
	const string& front() const;                                  // get front element
	void addFront(const string& e, const int& i);       // add to front of list
	void addBack(const string& e, const int& i);        // add to back of list
	void removeFront();                                                 // remove front item of list
	void removeBack();                                                  // remove back item of list
	void print();

	void insertOrdered(const string& e, const int& i) ;
	void insertOrdered(SinglyNode* newNode, SinglyNode* previous);

	void removeOrdered(const string& e, const int& i) ;

	SinglyLinkedList* mergeLists(SinglyLinkedList* list2) ;
};

SinglyLinkedList::SinglyLinkedList()                                // constructor
{
	head = NULL;
}
```

```
SinglyLinkedList::~SinglyLinkedList()                                   // destructor
{
      while (!empty()) removeFront();
}

bool SinglyLinkedList::empty() const                                    // is list empty?
{
      return head == NULL;
}

const string& SinglyLinkedList::front() const                          // get front element
{
      return head->elem;
}

// [Fatih]->[Oguzhan]->[Omer]
void SinglyLinkedList::addFront(const string& e, const int& i)    // add to front of list
{
      SinglyNode* v= new SinglyNode;                                    // create new node
      v->elem                   = e;                                                    // store data
      v->score            = i;
      v->next                   = head;                                                       // head now
follows v
      head                = v;                                                  // v is now the head
}

void SinglyLinkedList::removeFront()                                    // remove front item
{
      if (empty())
      {
            cout << "List is empty !" << endl;
            return;
      }

      SinglyNode* temp    = head;                                               // save current head
      head                      = head->next;                                 // skip over old head
      delete temp;                                                              // delete the old head
}
```

```
// [Omer]->[Oguzhan]->[Fatih]
void SinglyLinkedList::addBack(const string& e, const int& i)
{
      SinglyNode* v= new SinglyNode;
      v->elem                   = e;
      v->score            = i;
      v->next                   = NULL;

      if (head == NULL) head = v;
      else
      {
            SinglyNode* first = head;
            while (first->next != NULL) first = first->next;
            first->next = v;
      }
}

// 2017-2018 Güz Vize 1. Soru
void SinglyLinkedList::removeBack()
{
      if (empty())
      {
            cout << "List is empty !" << endl;
            return;
      }

      SinglyNode* previous = head;

      // kalan tek düğümü sil
      if (previous->next == NULL)
      {
            head = NULL;
            delete previous;
      }
      // en az iki düğüm var
      else
      {
            while (previous->next->next != NULL)
                  previous = previous->next;
```

```
		delete previous->next;									// previous-
>next 'i silinir
		previous->next = NULL;
	}
}

//void SinglyLinkedList::removeBack() // [*** OLD VERSION ***]
//{
//	if (empty())
//	{
//		cout << "List is empty !" << endl;
//		return;
//	}
//
//	SinglyNode* last = head;
//	SinglyNode* prev = head;
//
//	while (last->next != NULL)
//	{
//		prev = last;
//		last = last->next;
//	}
//
//	prev->next = NULL;
//	if (last == head) head = NULL;
//	delete last;
//}

// [Jack,510]->[Rose,590]->[Anna,660]->[Paul,720]->[Rob,750]->[Mike,1105] // Jill eklenmeden önce
void SinglyLinkedList::insertOrdered(const string& e, const int& i)		// previous->next 'ine eklenir
{
	//string f = "Omer";  // https://www.quora.com/What-does-const-string-in-C++-mean

	SinglyNode* newNode	= new SinglyNode;
	newNode->elem		= e;
	newNode->score			= i;
	/*newNode->next			= NULL;*/
```

```
        // Liste boş mu?
        if(head == NULL)
        {
                head = newNode;
                newNode->next = NULL;
                return;
        }

        // newNode listenin başına mı eklenecek?
        if(newNode->score < head->score)
        {
                newNode->next= head;
                head                = newNode;
                return;
        }

        SinglyNode* previous = head;
        while (previous->next != NULL)
        {
                if(newNode->score >= previous->next->score)
                        previous = previous->next;
                else
                        break;
        }
        // newNode'u previous'tan sonra ekle
        newNode->next= previous->next;
        previous->next      = newNode;
}

//void SinglyLinkedList::insertOrdered(const string& e, const int& i) // [*** OLD VERSION ***]
//{
//      SinglyNode* newNode = new SinglyNode;
//      newNode->elem = e;
//      newNode->score = i;
//      newNode->next = NULL;
//
//      // Liste bos mu?
//      if (head == NULL)
//      {
```

```
//            head = newNode;
//            return;
//      }
//
//      // newNode listenin basina mi eklenecek?
//      if (newNode->score < head->score)
//      {
//            newNode->next = head;
//            head = newNode;
//            return;
//      }
//
//      SinglyNode* current = head;
//      SinglyNode* previous = NULL;
//      while (current != NULL)
//      {
//            if (newNode->score >= current->score)
//                  previous = current;
//            else
//                  break;
//            current = current->next;
//      }
//
//      // newNode'u previous'tan sonra ekle
//      newNode->next = previous->next;
//      previous->next = newNode;
//}

// 2018-2019 Güz Bütünleme 1. Soru
void SinglyLinkedList::insertOrdered(SinglyNode* newNode, SinglyNode* previous)
{
      if ((previous->next == NULL) || (newNode->score <= previous->next->score))
      {
            newNode->next= previous->next;
            previous->next     = newNode;
      }
      else
            insertOrdered(newNode, previous->next);
}
```

```
// 2016-2017 Güz Bütünleme 3. Soru
// [Rose,590]->[Anna,660]->[Paul,720]->[Jill,740]->[Rob,750] // Paul silinmeden önce
void SinglyLinkedList::removeOrdered(const string& e, const int& i)          // previous->next 'i silinir
{
      // Liste boş mu?
      if (empty())
      {
            cout << "List is empty !" << endl;
            return;
      }

      // Listenin ilk elemanı mı silinecek?
      if ((e.compare(head->elem) == 0) && (head->score == i))
      {
            SinglyNode* temp = head;
            head = head->next;
            delete temp;
            return;
      }

      SinglyNode* previous = head;
      while (previous->next != NULL)
      {
            if ((e.compare(previous->next->elem) == 0) && (previous->next->score == i))
            {
                  SinglyNode* temp = previous->next;
                  previous->next = previous->next->next;
                  delete temp;                                                   // previous->next 'i silinir
                  return;
            }

            previous = previous->next;
      }

      if (previous->next == NULL) cout << "\n" << e << " is not found" << endl;
}
```

```
//void SinglyLinkedList::removeOrdered(const string& e, const int& i)          // current silinir   [*** OLD VERSION
***]
//{
//     // Liste bos mu?
//     if (empty())
//     {
//           cout << "List is empty !" << endl;
//           return;
//     }
//
//     // Listenin ilk elemani mi silinecek?
//     if ((e.compare(head->elem) == 0) && (head->score == i))
//     {
//           SinglyNode* temp = head;
//           head = head->next;
//           delete temp;
//           return;
//     }
//
//     SinglyNode* previous = head;
//     SinglyNode* current = head->next;
//
//     while (current != NULL)
//     {
//           if ((e.compare(current->elem) == 0) && (current->score == i))
//           {
//                 previous->next = current->next;
//                 delete current;                  // current silinir
//                 return;
//           }
//
//           previous = current;
//           current = current->next;
//     }
//
//     if (current == NULL) cout << "\n" << e << " is not found" << endl;
//}

SinglyLinkedList* SinglyLinkedList::mergeLists(SinglyLinkedList* list2)
```

```
{
	SinglyLinkedList* mergedList = new SinglyLinkedList();

	SinglyNode* plist1 = this->head;
	SinglyNode* plist2 = list2->head;

	while ((plist1 != NULL) || (plist2 != NULL))
	{
		if (plist1 == NULL)
		{
			mergedList->addBack(plist2->elem, plist2->score);
			plist2 = plist2->next;
			continue;
		}

		if (plist2 == NULL)
		{
			mergedList->addBack(plist1->elem, plist1->score);
			plist1 = plist1->next;
			continue;
		}

		if (plist1->score <= plist2->score)
		{
			mergedList->addBack(plist1->elem, plist1->score);
			plist1 = plist1->next;
		}
		else
		{
			mergedList->addBack(plist2->elem, plist2->score);
			plist2 = plist2->next;
		}
	}

	return mergedList;
}

void SinglyLinkedList::print()
{
	if (empty())
	{
		cout << "List is empty !" << endl;
		return;
	}

	SinglyNode* first = head;
	while (first != NULL)
	{
		cout << first->elem << "\t" << first->score << endl;
		first = first->next;
	}
}

int main()
{
	SinglyLinkedList list;

	list.addFront("Omer", 1000);
	list.addFront("Oguzhan", 1500);
	list.addFront("Fatih", 1250);
	list.print();

	//cout << endl;
	//list.removeFront();
	//list.print();

	//cout << endl;
	//list.removeFront();
	//list.print();

	//cout << endl;
	//list.removeFront();
	//list.print();

	//SinglyLinkedList 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);

//cout << "List after insertions :" << endl;
//list.print();

// Recursive insertOrdered()
//SinglyLinkedList list;
//SinglyNode* newNode;

//list.head = new SinglyNode;
//list.head->elem = "NoName";
//list.head->score = 0;
//list.head->next = NULL;

//newNode = new SinglyNode;
//newNode->elem = "Paul";
//newNode->score = 720;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
//newNode->elem = "Rose";
//newNode->score = 590;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
//newNode->elem = "Anna";
//newNode->score = 660;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
//newNode->elem = "Mike";
//newNode->score = 1105;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
```

```
//newNode->elem = "Rob";
//newNode->score = 750;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
//newNode->elem = "Jack";
//newNode->score = 510;
//list.insertOrdered(newNode, list.head);

//newNode = new SinglyNode;
//newNode->elem = "Jill";
//newNode->score = 740;
//list.insertOrdered(newNode, list.head);

//cout << "List after insertions :" << endl;
//list.print();

//list.removeOrdered("Adam", 610);     // Bu eleman listede yok !

//list.removeOrdered("Jack", 510);
//list.removeOrdered("Mike", 1105);
//list.removeOrdered("Paul", 720);

//cout << "\nList after removals (Jack, Mike, Paul) :" << endl;
//list.print();

//list.removeOrdered("Rose", 590);
//list.removeOrdered("Rob",  750);
//list.removeOrdered("Anna", 660);
//list.removeOrdered("Jill", 740);

//cout << "\nList after removals (Rose, Rob, Anna, Jill ) :" << endl;
//list.print();

//SinglyLinkedList* list1 = new SinglyLinkedList();
//list1->insertOrdered("Mike", 1105);
//list1->insertOrdered("Rob", 750);
//list1->insertOrdered("Paul", 720);
//list1->insertOrdered("Anna", 660);
```

```
	//cout << "list1 : " << endl;
	//list1->print();
	//cout << endl;

	//SinglyLinkedList* list2 = new SinglyLinkedList();
	//list2->insertOrdered("Rose", 590);
	//list2->insertOrdered("Jack", 510);
	//list2->insertOrdered("Jill", 740);
	//list2->insertOrdered("Adam", 610);
	//cout << "list2 : " << endl;
	//list2->print();
	//cout << endl;

	//SinglyLinkedList* mergedList = list1->mergeLists(list2);

	//cout << "Merged list :" << endl;
	//mergedList->print();

	getchar();
}
```