MIDTERM EXAMINATION

Course: **CS162 – COMPUTER SCIENCE II**	
Time: **90** minutes	Term: **2** – Academic year: **2015-2016**
Lecturer(s): **Dr. Dinh Ba Tien**	
Student name:	Student ID:

(Notes: Closed book exam)

Question 1. What are the differences of a **dynamically allocated array** comparing to a normal array? How to **allocate** and **de-allocate** a dynamically allocated array?

Question 2. What are the **advantages** and **disadvantages** of a singly linked list comparing to an array?

Question 3. Given a singly linked list, you are asked to implement the following functions:

a. Count and return **the number of nodes** in the list.
b. Remove the **last node** of the list.
c. Insert a value K into **the end** of the list. Please note if K is already in the list, don't insert it.
d. The list could have many duplicated nodes. **Remove and keep only one** node for each group of the duplicated ones.

E.g: List: 44 → 18 → 44 → 62 → 18 → 44 → 62 → NULL

List after being removed: 44 → 18 → 62 → NULL

Question 4. Given a singly linked list whose values are sorted in **ascending order** (i.e. nodes' values are increasing), write a function to **insert a new node** with value K into the list and still keep it sorted.

E.g: List: 14 → 18 → 37 → 46 → 46 → 83 → NULL

Insert 67: ==> List: 14 → 18 → 37 → 46 → 46 → **67**→ 83 → NULL

-- GOOD LUCK --

MIDTERM EXAMINATION

Course: **PROGRAMMING TECHNIQUES**	
Time: **90** minutes	Term: **2** – Academic year: **2015-2016**
Lecturer(s): **Dr. Dinh Ba Tien**	
Student name:	Student ID:

(Notes: Closed book exam)

Question 1. What is a **pointer**? What happens if we **forget to deallocate** a pointer after we finish using it?

Question 2. In a singly linked list, if there are both **pHead** and **pTail** pointers to point to the first node and the last node, to remove a node at **the beginning** or at **the end**, which one is easier and faster? Why?

Question 3. Assuming that we have a **singly linked list** controlled by a **pHead** pointer, you are asked to implement the following functions:

a. Return the **pointer** to the node with the **biggest value**:

```
Node* getBiggest(Node* pHead);
```

b. Remove **all the prime numbers** from the linked list.

```
void removePrimeNumbers(Node* &pHead);
```

c. Print out all the nodes whose values are bigger than the previous node

E.g: Linked list: 34 → 43 → 15 → 62 → 39 → NULL

Print out: 43 62

d. **Insert** the odd counting numbers 1, 3, 5... before the nodes at the odd position of the list:

```
void insertOddCounting(Node* &pHead);
```

E.g: List: 34 → 43 → 15 → 62 → 39 → NULL

Become: **1** → 34 → 43→ **3** → 15 → 62 → **5** → 39 → NULL

e. **Reverse** the list:

```
void reverse(Node* &pHead);
```

E.g: Linked list: 34 → 43 → 15 → 62 → 39 → NULL

New list: 39 → 62→ 15 → 43 → 34 → NULL

*** GOOD LUCK ***