Karadeniz Technical University
Department of Computer Engineering
Lecturer Ömer ÇAKIR

COM 2005 Data Structures
Midterm Exam, 12.11.2019, 15:00
Duration : **90** Minutes

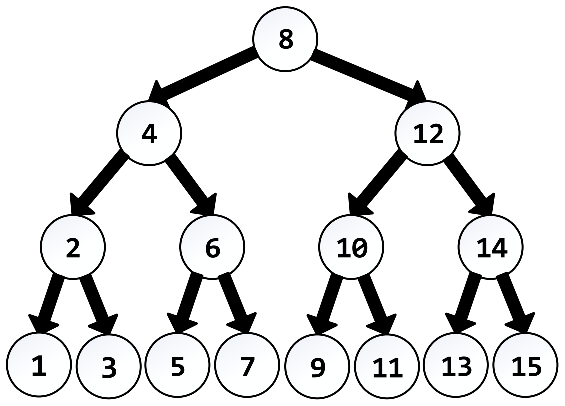
NUMBER : .... **NAME :** .... **SIGNATURE :**	**EXAM GRADE**	
	[........]	....

Students have to obey Engineering Faculty Exam Execution Instructions.
Questions are related to 1,4,12 of Program Learning Outcomes

```
void traverse(TreeNode* v)
{
    if (v->left != NULL)
    {
        traverse(v->left);
        cout << v->elem << " ";
    }

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

1. What is the output of the function **traverse()** that is called with the **root** of the tree below as the argument? **(25P)**



1 2 3 4 5 6 7 8

2. Assume that the numbers above are inserted into a binary tree. Which of the outputs of the `inorder`, `preorder` and `postorder` traversals is different from the other two? **(25P)**

*You'll loose **5P** from wrong answer.*

(A) inorder

(B) preorder

(C) postorder

```
void insertOrdered(string& e, int& i)
{
   DoublyNode* newNode      = new DoublyNode;
   newNode->elem            = e;
   newNode->score           = i;

   DoublyNode* current      = header;

   while (current->next != trailer)
   {
      if (newNode->score >= current->next->score)
        current = current->next;
      else
        break;
   }

   newNode->next     = current->next;
   newNode->prev     = current;
   ...............   = ...............;
   ...............   = ...............;
}
```

3. Considering the two lines of the **insertOrdered()** function that are indicated by "**.....**", which of the following choises add a node to a doubly linked list <u>erroneously</u>? **(25P)** *You'll loose **5P** from wrong answer.*

```
(A)    newNode->prev->next       = newNode;
       newNode->next->prev       = newNode;

(B)    newNode->next->prev       = newNode;
       newNode->prev->next       = newNode;

(C)    current->next->prev       = newNode;
       current->next             = newNode;

(D)    current->next->prev       = newNode;
       newNode->prev->next       = newNode;

(E)    newNode->prev->next       = newNode;
       current->next->prev       = newNode;
```

```
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)
     {
        ........................................

        ........................................

     }
     else
     {
        ........................................

        ........................................

     }
  }
  return mergedList;
}
```

4. Complete the function **mergeLists()** above. **(25P)**