BBM201 – Data Structures – Fall 2017
2nd Midterm
14.12.2017 – 13:00-15:00

Name Surname: _____

Student ID : _____ **Section:**_____

Duration: 120 minutes

Question	1	2	3	4	5	6	7	Total
Points	12	12	16	18	12	12	18	100
Grade								

Question 1. Evaluation of expressions.

a. If a= 10, b= 3, c=−2, evaluate the following postfix expression:

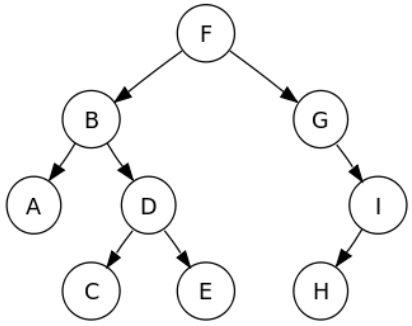
aa+bc-* = 100

b. Convert the following fully parenthesized infix expression to postfix and prefix notation:

(x-((x+y)*((z/r)+p)))

postfix : xxy+zr/p+*-
prefix : -x*+xy+/zrp

Question 2. Trees. Give the traversal order of the accessed elements for each traversal algorithm for the given tree.



Preorder: FBADCEGIH

Inorder: ABCDEFGHI

Postorder: ACEDBHIGF

Question 3. Array based linked list. According to the given array based linked list that is defined as follows:

```
#define MAX_LIST 10

typedef struct{
      char name[7];
      int link;
    }item;
item linkedlist[MAX_LIST];
int free_; //the index of the first free space in the list
int* list; //the index of the first item in the list
```

Imagine we begin with a sorted list in the first table given below. After we insert "Adil" and delete "Leyla" successively in the same sorted list, what will be the contents of the list? Please fill in the tables and write down the values of free_ and *list.

Initial table

	name	link
[0]	"Banu"	1
[1]	"Irem"	4
[2]	"Zerrin"	-1
[3]	"Ali"	0
[4]	"Leyla"	6
[5]	"Ahmet"	3
[6]	"Mehmet"	2
[7]		8
[8]		9
[9]		-1

free_= 7
*list = 5

Insert "Adil"

	name	link
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		

free_=
*list =

Delete "Leyla"

	name	link
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		
[9]		

free_=
*list =

Question 4. ***Linked lists.*** The information of athletes applying for a long distance run will be stored as a linked list. Assume that the athelete list is not globally defined.

```
#define ARRAY_SIZE 21

typedef struct athlete {
      int no;
      char name[ARRAY_SIZE];
      double runningTime;
      struct athlete *next;
   } athleteType;
```

a. Write the function named ***printList*** that displays all the information of the athletes whose running times are smaller than the time given as a parameter to the function.

```
void printList(athleteType *list, double time)
{
      for(;list != NULL; list = list->next)
         if(list->runningTime < time)
            printf("%d  %s  %lf  \n",  list->no,  list->name,  list->runningTime);
}
```

b. Write the function named ***append*** that adds a new athlete with the given no, name and time information to the end of the list.

```
void append(athleteType **list, int no, char name[ARRAY_SIZE], double time)
{
    athleteType *temp = *list;
     athleteType                                          *newA=(athleteType*)malloc(sizeof(athleteType));

     newA->no = no;
     strcpy(newA->name, name);
     newA->runningTime = time;
     newA->next = NULL;


     if(*list == NULL)
        *list = new;
     else{
       for(;temp->next != NULL; temp = temp->next)
           ;
       temp->next = newA;
     }
}
```

c. Complete the main method given below according to the above implementations.

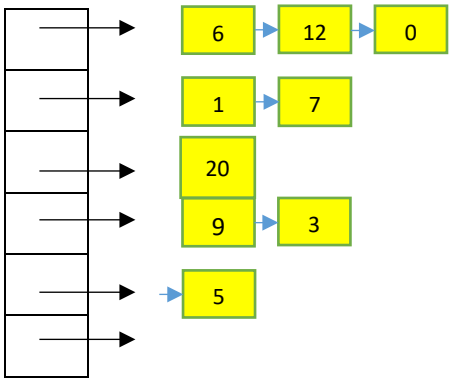
```
int main (void) {
   athleteType *list = NULL;
   append(  &list    , 10, "Ang", 14.80);//the list becomes 10 -> NULL
   append(  &list    , 8, "Appa", 16.20); //the list becomes 10 -> 8 -> NULL
   append(  &list   , 5, "Momo", 18.50);//the list becomes 10 -> 8 -> 5 -> NULL
   printList (   list , 17.00); //output : 10 Ang 14.80 , 8 Appa 16.20
    return 0;
}
```

Question 5. Hashtables.

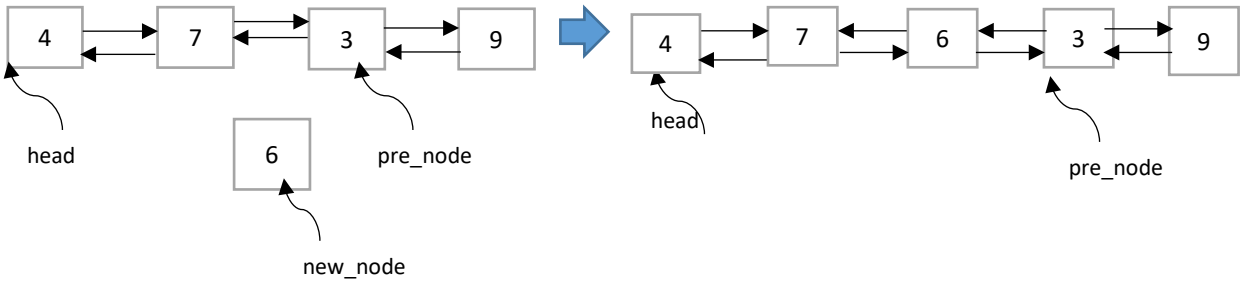
```
#define TABLE_SIZE 6

struct list *list_pointer;
typedef struct list {
    int key;
    list_pointer link;
};
list_pointer hash_table[TABLE_SIZE];

int hash_function(int key){
       return key % TABLE_SIZE;
}
void insert(int key)
{
       int hash_value = hash_function(key);
       list_pointer ptr, trail=NULL, lead=hash_table[hash_value];
       for (; lead; trail=lead, lead=lead->link)
              if (!strcmp(lead->key, key)) {
              printf("The key is in the table\n");
                     exit(1);
        }
        ptr = (list_pointer) malloc(sizeof(list));

         ptr->key = key;
         ptr->link = NULL;
         if (trail) trail->link = ptr;
         else hash_table[hash_value]= ptr;
}
```

According to the above hashtable implementation, fill in the below figure after the following method calls are applied:

insert(9); insert(20); insert(6);insert(12); insert(3); insert(5); insert(0); insert(1); insert(7);



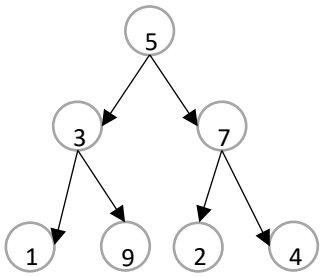
Question 6: Doubly linked list.



Write the method that inserts a new node before the given previous node in a doubly linked list. An example situation is given below.

```
struct node{
      int data;
      struct node* next;
      struct node* pre;
};
void insert(struct node* pre_node, struct node* new_node){

new_node->next = pre_node;
new_node->pre=pre_node->pre;
pre_node->pre->next = new_node;
pre_node->pre = new_node;

}
```

Question 7. Binary trees. We will use an array in order to store a full binary tree.



a) Fill in the given array that will include the tree contents given above.

5	3	7	1	9	2	4

b) Complete the given code that returns the nth node in the given level. For example, find_data(3, 2) will return 9, find_data(2, 1) will return 3, and so on.

```
#define ARRAY_SIZE 7
int tree[ARRAY_SIZE];
int pow (int x, int y); //assume the method is already defined.

int find_data(int level, int n){

return tree[pow(2, level-1)-1+n-1];


}
```