ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Υποεργασία 1

```
1. void loadMatrix (int **F, int size) {

    int i,j; // temp loop variables
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            while ( i!=j && i>j) // defensive mechanism for out-of-
              bound values
            {
                printf("Enter '1' if  users %d and %d friends, '0'
                  otherwise : ",i+1, j+1 );
                scanf("%d",&F[i][j]);
                //printf("%d",F[i][j]);
                while (!(F[i][j]==0 || F[i][j]==1))
                {
                    printf("Accepted values are '1' and '0'. Please
                      try again : ");
                    scanf("%d",&F[i][j]);
                }

                F[j][i]= F[i][j]; // populate duplicate array
                  elements

                break;
            }
        }
    }
}
```

```
2. int findFriends (int **F, int size, int user) {

    int i, count=0; // temp variables

    for (i = 0; i < size; i++) // iterate through user's row
    {
        if (F[user][i]==1) count+=1; /* when element is 1, increment
         count*/
    }
    return count;
}
```

```
3.int commonFriends (int **F, int size, int user1, int user2) {

    int i, count=0; // temp variables

    for (i = 0; i < size; i++)
    { /* Compare user's row elements by column, if both 1 increment
      count */
      if (F[(user1)][i]==1 && F[(user2)][i]==1) count +=1;
    }
    return count;
}
```

```
4. void sortUsers (int **F, int size, int *S) {

    int i, j,k; /* temp loop variables  */
    int temp, swap; /* bubble sort variables. temp is to swap
      elements, swap is to count no. of swaps in each iteration. */

    /* Iterate through F array by row, add each element to S(S's
      elements already initialized to 0 in main).*/
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            S[i] += F[i][j];
        }
    }
    /* bubble sort of S array*/
    for (i = 0; i < size; i++)
    {
        swap=0; // initialize swap count
        /* after the i th iteration, the last i elements are sorted;
            no need to check again  */
        for (j = 0; j < size-(1+i); j++)
        {
            if (S[j] > S[j+1])
            {
                temp = S[j+1];
                S[j+1] = S[j];
                S[j] = temp;
                swap++;
            }
        }
        if (swap==0) break; /* if no swaps occurred in last
      iteration,  array is sorted; exit loop. */
    }
}
```

Υποεργασία 2

```
1.  CListNode *insert_at_end(CListNode *end_ptr, char *a)
{
    CListNode *new;

    if (end_ptr==NULL) /*if list is empty*/
    {
        /*allocate memory for node*/
        new=(CListNode *)malloc(sizeof(CListNode));
        /*copy string to node's name field*/
        strcpy(new->name, a);
        /*since list has only 1 node, it should point to itself.*/
        new->next = new;
    } else
    {
        new=(CListNode *)malloc(sizeof(CListNode));
        strcpy(new->name, a);
        /*node points to the start of the list*/
        new->next = end_ptr->next;
        /*place node to the end of list*/
        end_ptr->next = new;
    }
    return new ;
}

2.  CListNode *initialize_list(int n)
{
    CListNode *tail;
    tail=NULL;
    while (n)
    {
        printf("Enter Person's Name (max. 9 characters): ");
        get_name(a);
        tail = insert_at_end(tail, a);
        n--;
    }
      return tail;
}
```

```
3.  CListNode *delete_next_node(CListNode *end_ptr, CListNode *p)
{

    CListNode *current, *last, *temp;
    last = end_ptr;
    current = last->next; //current points to first node of list.

    if (last==NULL) printf("List is empty.\n");
    /* traverse list in search of p */
    while (last!=current && p!=current) {
            current = current->next;
    }
    /* move 1 node after p*/
    temp=current->next;
    printf("%s removed from the list\n",temp->name);
    /*remove node from the list*/
    current->next=temp->next;
    /*delete node freeing its memory*/
    free(temp);
/*returns pointer to the node after the target, for the next
iteration*/
return current->next;
}
```

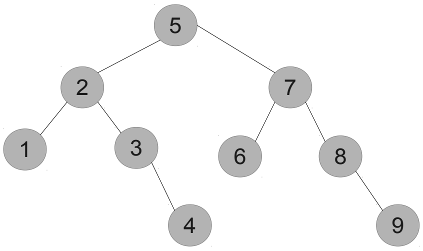
```
4.  void print_list(CListNode *end_ptr)
{
    CListNode *last, *current;
    int i=1;
    last = end_ptr;
    if (last==NULL) printf("List is empty.");
    else
    {
        current = last->next; /*current points to start of list*/
        /*traverse list nodes printing their names*/
        while (current != last)
        {
            printf("%s\t", current->name);
         /*use modulus to delimit no of nodes printed in each line */
            if (!(i%NUM_PER_LINE)) printf("\n");
            i++;
            current = current->next;
        }
        printf("%s\t", last->name);
    }
}
```

5.
```
CListNode *Select(CListNode *end_ptr, int k)
{
    int i;
    CListNode *current, *last;
    last = end_ptr;
    /*current points to start of list*/
    current = last->next;
    if (last==NULL) printf("List is empty.\n");
    /*notify if only 1 node in list*/
    else if (last==current){
       printf("Cannot perform Operation. List contains only 1
      element.\n");
    }
    else
    {
        /*keep performing the code below until only 1 node remains*/
        while (last!=current)
        {
        /*move to the k node*/
            i=k;
        while (i)
        {
            current = current->next;
            i--;
        }
        /*delete the node and return the node it pointed to.*/
        last = delete_next_node(last, current);
        }
    }
    /*return the 1 remaining node*/
    return last;
}
```

Υποεργασία 3

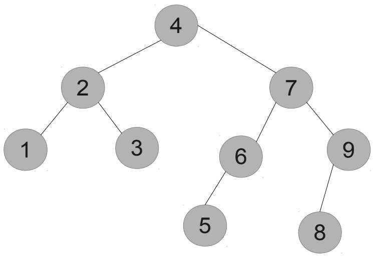
1. **Βαθμολογία Παικτών** :

Index	1	2	3	4	5	6	7	8	9	Score
PRE	5	2	1	3	4	7	6	8	9	3
POST	1	4	3	2	6	9	8	7	5	2



2.

Index	1	2	3	4	5	6	7	8	9	Score
PRE	4	2	1	3	7	6	5	9	8	2
POST	1	3	2	5	6	8	9	7	4	1



MOVES	
PRE	POST
4	7
2	9
6	3
8	5
1	

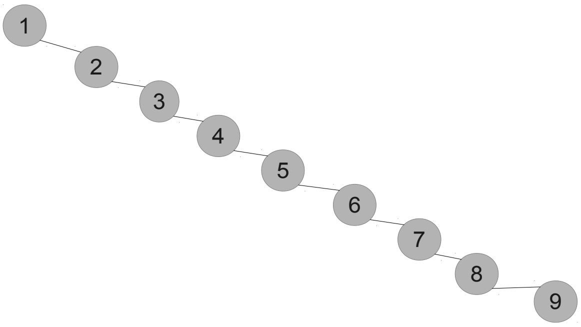
3.

Θεωρητικά, και οι δύο παίκτες μπορούν να επιτύχουν την μέγιστη βαθμολογία (9). Για τον μεν PRE, καθώς έχει την πρώτη επιλογή και η δια-πέραση του αρχίζει από τη ρίζα του κόμβου, ένα παιχνίδι στο οποίο οι παίκτες επιλέγουν τους αριθμούς σειριακά (με αύξουσα σειρά), θα απέφερε στον PRE την μέγιστη βαθμολογία.

Βέλτιστο σενάριο PRE :

Index	1	2	3	4	5	6	7	8	9	**Score**
PRE	1	2	3	4	5	6	7	8	9	**9**
POST										

MOVES	
PRE	**POST**
1	2
3	4
5	6
7	8
9	

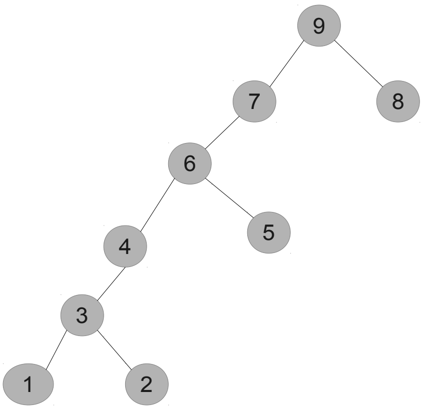


Για τον POST, δεδομένου η δια-πέραση ξεκινά από τα φύλλα (αριστερό -> δεξιό) και ανεβαίνει στην ρίζα, καταλήγουμε στο εξής σενάριο :

Βέλτιστο σενάριο PRE :

Index	1	2	3	4	5	6	7	8	9	**Score**
PRE										
POST	1	2	3	4	5	6	7	8	9	**9**

MOVES	
PRE	**POST**
9	7
6	5
8	4
3	1
2	



Υποεργασία 4

```
1.  int insertElement (int x, int counter, t_htentry *ht, int size){
    int ht_Index;
    // using the hash function
    ht_Index = x % HTSIZE;

    //build a node to host the number
    t_listnode *hashNode=malloc(sizeof(t_listnode));
    if(NULL == hashNode)
    {
        printf("\n Memory Allocation of hash Node failed \n");
        return 0;
    }
    //populate the node's fields
    hashNode->value = x;
    hashNode->counter = counter;
    hashNode->next = NULL;
    /*position the node to the linked list*/
    /* if position is empty, the node populates both head and tail
pointers*/
    if ((ht + ht_Index)->head ==NULL)
    {
        (ht + ht_Index)->head = (ht + ht_Index)->tail = hashNode;
    }
    /*if there is a node, have its *next point to the new node, and
set new node as tail*/
    else
    {
        (ht + ht_Index)->tail->next = hashNode;
        (ht + ht_Index)->tail = hashNode;

    }

return 1;
}
```

```
2.  int searchElement (int x, t_htentry *ht, int size) {
     int ht_Index;
    // using the hash function
    ht_Index = x % HTSIZE;

    t_listnode *searchNode = (ht + ht_Index)->head;
    /*Search is pointless if there are no nodes in that index.*/
    if (searchNode==NULL) return 0;
    /*traverse the list in ht[ht_Index] in search of the number*/
    while (searchNode->next!=NULL)
    {
        if ((searchNode->value)==x )
        {
           return searchNode->counter;
        }
        else searchNode = searchNode->next;
    }
/*the code will get here if the search was negative.*/
return 0;
}
```

```
3.  int generate(int a, int c, int m, int x) {

    int nextRand=0;
    nextRand = (a*x + c)%m ;
    return nextRand;
}
```

```
4.   int findPeriod(int a, int c, int m, int x) {
    int i, period, check;
    /*allocate memory for the hashtable and initialize each of the
      element's pointers*/
    t_htentry *hashTable = malloc(HTSIZE * sizeof(t_htentry));
    if(NULL == hashTable)
    {
        printf("\n Memory Allocation of hash Table failed \n");
        return 0;
    }
    for (i = 0; i < HTSIZE; i++)
    {
        (hashTable+i)->head=(hashTable+i)->tail=NULL;
    }
    /*initialize random no to SEED*/
    int randNo = SEED;
    int counter =0; //initialize counter

    for (i = 0; i < M; i++)
    {
        //generate random number
        randNo = generate(a,c,m,randNo);
        /*check to see if an occurrence of the number exists in the
            hash table*/
        check = searchElement(randNo,hashTable,HTSIZE);
        /*if prior occurrence exists, subtract from counter to get
            the period, and return the value*/
        if (check)
        {
            period = counter - check;
            return period;
        }
        /*otherwise populate the hash table and increment the counter
            value*/
        else counter +=insertElement(randNo, counter, hashTable,
HTSIZE);

    }
return 0;
}
```