Name/Surname: Id:

CENG 112 – DATA STRUCTURES

SPRING 2014-2015 / Final Exam

11.06.2015

- Exam duration is 110 minutes
- No written notes
- No electronic devices
- ...Good Luck...

	Q1	Q2	Q3	Q4	Q5	Total
Points	15	25	20	20	20	**100**
Grade						

Q1. (15 Points, HW6) Hash Tables and Linear Probing

a) Insert the following string-integer pairs into the hash table of size 8 buckets using linear probing as the collision resolution technique and the given table of hash codes:

<”ABC”,42>, <”DEF”,23>, <”GHI”,56>, <”JKL”,98>, <”MNO”,23>, <”PQR”,98>, <”STU”,44>

Key	**“ABC”**	**“DEF”**	**“GHI”**	**“JKL”**	**“MNO”**	**“PQR”**	**“STU”**
Hash	5	4	2	5	3	5	2

Answer:

Index	0	1	2	3	4	5	6	7
Key								
Value								

Name/Surname: Id:

b) Given the following definitions for a hash table structure that holds string keys and integer values using linear probing and a function for hashing strings, fill in the following function that searches for a given string key in the table and returns the associated value or INT_MIN if the key is not in the table:

```
struct HTNode {
        char *key;
        int value;
};

struct HashTable {
        int n_buckets;
        struct HTNode *buckets;
};

int hash_string(const char *s)
{
        int h = 0;
        while (*s != '\0')
                h = 31*h + (int)*(s++);

        return h;
}
```

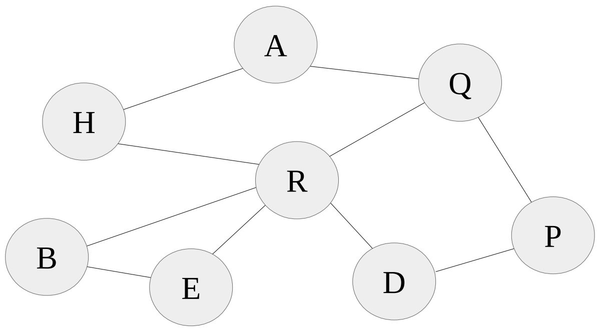
Answer:

```
int hash_table_find(struct HashTable *ht, char *key)
{

}
```

Name/Surname: Id:

Q2. (25 Points, HW7) Graphs and Graph Traversal



a) For the given graph above fill in the following adjacency matrix:

	A	**B**	**D**	**E**	**H**	**P**	**Q**	**R**
A								
B								
D								
E								
H								
P								
Q								
R								

b) Assuming that the neighbor nodes are processed in alphabetical order, write the order of nodes visited for

i) Breadth-first search starting at A: **A,**

ii) Depth-first search starting at A : **A,**

c) Implement the following function that decides whether any of the neighbors of a given node are connected to each other or not. For example the function should return TRUE for E since B and R are connected, and return FALSE for Q since its neighbors A, R, and P are not directly connected.
HINT: adj_mat[i][j] gives the ith row of the jth column in the above matrix.

```
#define TRUE 1
#define FALSE 0
int is_neighbors_connected(int **adj_mat, int node_id) {








}
```

d) Which representation is more efficient for the implementation of this function adjacency list or adjacency matrix and why:

Name/Surname: Id:

Q3. (20 Points) Arrays and Queues

a) Write a function that prints the last character of every string in an array of strings of size n (It should not print anything if the string is NULL or the empty string “”):

```
void print_last_chars(int n, char **arr) {








}
```

b) We can implement a fixed size queue using an array and two indices to store the head and tail positions. The following structure and function implements such a queue:

```
#define QUEUE_SIZE 5
struct Queue { float elems[QUEUE_SIZE]; int head; int tail; };
void enqueue_tail(struct Queue *q, float f) {
      q->elems[q->tail++] = f;
      if (q->tail == QUEUE_SIZE) q->tail = 0;
}
float dequeue_head(struct Queue *q) {
      float v = q->elems[q->head++];
      if (q->head == QUEUE_SIZE) q->head = 0;
      return v;
}
```

Write output of the following program:

```
int main(int argc, char **argv)
{
        struct Queue q;
        q.head = 0;
        q.tail = 0;

        for (int i = 0; i < 9; ++i) {
                enqueue_tail(&q, i);
                if (i % 2 == 1)
                        printf("%.2f\n", dequeue_head(&q));
                printf("Head = %d, Tail = %d\n", q.head, q.tail);
        }

        return EXIT_SUCCESS;
}
```

Output:

Name/Surname: Id:

Q4. (20 Points) Linked Data Structures

Given the following node structure for linked lists

```
struct Node {
	int data;
	struct Node *next;
};
```

a) Write an **iterative** function that compares the contents of two list and returns TRUE if and only if both lists are of the same size, the data items are the same, and in the same order. Otherwise the function should return FALSE.

```
int is_equal(struct Node *head1, struct Node *head2) {










};
```

b) Write a **recursive** version of the same function:

```
int is_equal(struct Node *head1, struct Node *head2) {










};
```

c) Write a function that takes a linked list and returns the head of a list containing the same items in reverse order (You can modify the given list).

```
struct Node *reverse_list(struct Node *head) {










};
```

Name/Surname: Id:

Q5. (20 Points) Sorting

a) Fill in the following function that exchanges two elements of a given array at the speficied indices:

```
void exchange(int *x, int i, int j) {






}
```

b) Given the following function for partitioning an array from the leftmost till the rightmost index, fill in the implementation of the quicksort algorithm:

```
int partition(int *x, int l, int r)
{
        int i = l-1; int j = r; int v = x[r];

        while (1) {
                while (x[++i] < v)
                        ;
                while (v < x[--j])
                        if (j == l)
                                break;
                if (i >= j)
                        break;
                exchange(x, i, j);
        }
        exchange(x, i, r);
        return i;
}
```

Answer:

```
void quicksort(int *x, int l, int r)
{









}
```

c) Using the same partition function, fill in the implementation of the quickselect algorithm that puts the k^th^ item in its correct place in the sorted array, without sorting the complete array.
HINT: The placement of the items in the correct place is achieved by calling partition, which puts the pivot element in the correct place.

```
void quickselect(int *x, int l, int r, int k)
{











}
```