Name: ID:

CENG 112 – DATA STRUCTURES

FALL 2014-2015 / MIDTERM I

02.04.2015

- Exam duration is 120 minutes
- No written notes
- No electronic devices
- ...Good Luck...

	Q1	Q2	Q3	Q4	Q5	Total
Points	20	10	25	20	25	**100**
Grade						

Q1. (20 Points, HW1) Auto Capitalization

a) Write a program that reads up to 10 *characters* from the standard input using a for loop and prints them on the standard output.

Hint: You can use the *getchar* function from <stdio.h>:
```
int getchar(void);
```
and you need to check for EOF.

b) Write a program that reads up to 10 *words* seperated by whitespace (' ', '\t', '\n') from the standard input and capitalizes (makes the first letter uppercase) each one.

Hint: You can use the *toupper* function from <ctypes.h>:
```
int toupper(int c);
```

Q2. (10 Points, HW2) Resizable Arrays

Fill in the blanks in the following program according to the comments.
Hint: *void *malloc(size_t size);*
*void *realloc(void *ptr, size_t size);*

```
// Program reads as many integers from stdin as possible and stores them in the array 'numbers'.
int main(void)
{
        int n = 0;
        int max_n = 2;
        int *numbers = ____________________________________ // allocate max_n integers with
malloc
        if (numbers == NULL) return -1;

        while (scanf("%d \n",&numbers[n]) == 1) {
                if (++n == max_n) {
                     // reallocate a larger int array, store it in 'numbers'
                     // and the maximum size in 'max_n'. You may use as many lines as you wish.

                     _________________________________

                     _________________________________

                     _________________________________
                }
        }

       return 0;
}
```

Name: ID:

Q3. (25 Points, HW3) Queue ADT

```
struct Queue;
struct Queue *queue_new(int elem_size);
void queue_free(struct Queue *q);
void queue_put(struct Queue *q, void *elem);
void queue_get(struct Queue *q, void *elem);
int queue_size(const struct Queue *q);
int queue_is_empty(const struct Queue *q);
```

a) Given the queue interface above, complete the following program so that it performs the following operations in the same order:

- it creates a queue that stores int's,
- puts the numbers 42 and 314159 into the queue,
- gets numbers from the queue until it is empty using a *while* loop, and prints them,
- frees the memory used by the program.

```
int main(void) {
       struct Queue *q =









}
```

b) Same as in part a for the following operations:

- it creates a queue that stores char *'s,
- puts the strings “That is no moon” and “I have a bad feeling about this” into the queue,
- gets strings from the queue until it is empty, computes and prints the longest one,
- frees the memory used by the program.

```
int main(void) {
       struct Queue *q =









}
```

IMPORTANT NOTE: For both part a and b, DO NOT hard code the output for the third operation!

Name: ID:

Q4. (20 Points) Pointers and Arrays

a) Check the statements below that contain an error (invalid statement or a mismatch with the comment) and then write the correct version matching the comment in the space provided:

```
int main(void) {
        int a = 112;
        char *s = “112”;
        int *p;
        int **pp;

[ ]     p = &a;         // Store the address of a in p

                        _____________________________
[ ]     *p = &211;      // Set a to 211

                        _____________________________
[ ]     pp = &p;        // Store the address of p in pp

                        _____________________________
[ ]     *pp = 389;      // Set a to 389

                        _____________________________
[ ]     a = *s[1];      // Store the character code of the second character of s in a

                        _____________________________
[ ]     p = malloc(10*sizeof(int)); // Create a new int array with 10 elements

                        _____________________________
[ ]     **(p+4) = a; // Set the value of the 4th array element to value of a

                        _____________________________
}
```

b) Write the output of the following program in the space provided

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
        char *s = malloc(10*sizeof(char));
        *s = 'A';
        for (int i = 1; i < 10; ++i)
                s[i] = s[i-1] + 2;
        s[5] = '\0';
        printf(“%s\n”, s);    // Line 0

        int a[] = { 0, 1, 2, 3, 4 };
        int n = 4;
        while (--n > 0) {
                a[n] += a[n-1];
                printf(“%d “, a[n]);
        }
        printf(“\n”); // Line 1
        printf(“%d\n”, a[0]); // Line 2

        int b[3] = { 0, 1, 2 };
        printf(“%d %d %d\n”, b[0], b[1], b[2]);    // Line 3

        int *p[3] = { &b[1], &b[0], &b[2] };
        printf(“%d %d %d\n”, *p[0], *p[1], *p[2]); // Line 4
}
```

Output:

L0:
L1:
L2:
L3:
L4:

Name: ID:

Q5. (25 Points) Linked Lists and Recursion

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

The following program uses the above node structure to create and manipulate linked lists:

```
int main(void) {
       struct Node *head = NULL;

       for (int i = 0; i < 10; ++i) {
              struct Node *n = malloc(sizeof(*n));
              n->data = i;
              n->next = head;
              head = n;
       }

       print_list(head);
       printf("Sum of the elements is %d\n", sum_list(head));
       free_list(head);
}
```

Its output is:

```
9 8 7 6 5 4 3 2 1 0
Sum of the elements is 45
```

a) Fill in the definition of *print_list* so that it prints a line containing list data.

```
void print_list(struct Node *n) {









}
```

b) Fill in the definition of *sum_list* to **ITERATIVELY** calculate&return the sum of list data.

```
int sum_list(struct Node *n) {









}
```

c) Fill in the definition of *sum_list* to **RECURSIVELY** calculate&return the sum of list data.

```
int sum_list(struct Node *n) {









}
```

d) Fill in the definition of *list_free* to **RECURSIVELY** free allocated memory.

```
void free_list(struct Node *n) {









}
```

Name: ID:

Name: ID: