Faculty of Electrical and Electronics Engineering

Introduction to Programming Language (C) (EEF 110E)

MIDTERM EXAM

Question: 1

Definition: Write a function to move all zeroes to the end of a given integer array.

Assumption: Only the first element of the array is not zero, every other element can be zero!

```
void sort_q1(int arr[], int len)
{
    ...
}
```

Example Code:

```
int arr1[] = {2, 0, 5,0, 7, 0, 4, 0, 7, -5, 8, 0};
int len = sizeof(arr1) / sizeof(arr1[0]);
printf("input:  ");
for (int i = 0; i < len; i++) printf("%3d", arr1[i]);
printf("\n");
sort_q1(arr1, len);
printf("output: ");
for (int i = 0; i < len; i++) printf("%3d", arr1[i]);
printf("\n");
```

```
input:    2  0  5  0  7  0  4  0  7 -5  8  0
output:   2  5  7  4  7 -5  8  0  0  0  0  0
```

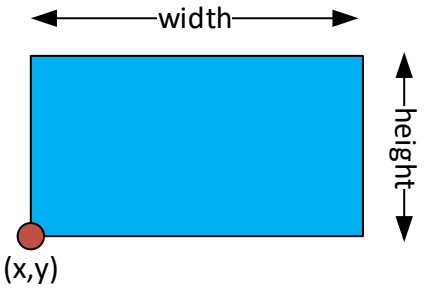
Faculty of Electrical and Electronics Engineering

Introduction to Programming Language (C) (EEF 110E)

MIDTERM EXAM

Question: 2

Definition: Write a function that calculates the percentage of the collided area if two rectangles are colliding in a 2D space. Your function should take two arrays that hold the dimensions of the rectangles (see Figure 1) and return the percentage of the collided (overlapped) area (over the total area) if there is a collision, and zero otherwise. It means that you should return $\frac{\text{area2}}{\text{area1+area2}} 100\ [\%]$ (see Figure 2).




```
float rect[4] = {x, y, width, height}
```

Figure 1: Input data type for rectangles.

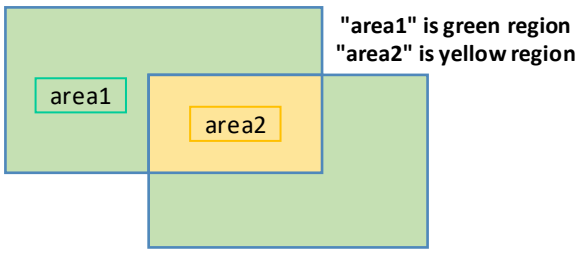


Figure 2: Colliding rectangles and colliding area.

```
float checkCollisionAreaPercent(float rect1[4], float rect2[4])
{
    ...
}
```

Example Code:

```
float R1[4] = {2.2F, 1, 3, 4.4F}, R2[4] = {3, 2, 5.5F, 5};
printf("Collision Percent: %.2f %%", checkCollisionAreaPercent(R1, R2));
```

Collision Percent: 22.52 %

Faculty of Electrical and Electronics Engineering

Introduction to Programming Language (C) (EEF 110E)

MIDTERM EXAM

Question: 3

```
#define ROW_LEN (1+10)
#define CAP 4
char data[ROW_LEN * CAP] = {0};
int count=0;

void dump();
void dump_strings();
int push(char str[], int str_len);
void status();
void pop();
void sort();

int main()
{
    char test1[] = "trabzon";
    char test2[] = "denizli";
    char test3[] = "adana";
    char test4[] = "zonguldak";
    char test5[] = "bolu";

    push(test1, sizeof(test1)/sizeof(test1[0]));
    push(test2, sizeof(test2)/sizeof(test2[0]));
    push(test3, sizeof(test3)/sizeof(test3[0]));
    push(test4, sizeof(test4)/sizeof(test4[0]));
    push(test5, sizeof(test5)/sizeof(test5[0]));

    dump_strings(); sort(); dump_strings(); status(); pop(); pop(); pop(); dump();

    return 0;
}
```

A memory is divided into fixed sized rows, where the first byte is the length of the string going to be stored between the first byte and the last byte of the row. The memory has a capacity for rows. This memory is going to be used to hold strings.

- Implement the **dump()** function that prints the whole memory as characters.
- Implement the **dump_strings()** function that prints only the stored strings.
- Implement the **push()** function which puts the size of the string and the string itself

into memory if capacity is not full and the length of the string does not exceed the limit. The function return 0 if it is not placed into memory and 1 otherwise.

- Implement the **status()** function which computes the sum of the length of the stored strings and prints the percentage of the used capacity.
- Implement the **pop()** function which removes the last string from the memory and fills the memory with dots.
- Implement the **sort()** function with any sorting algorithm you wish to sort the stored strings from A-Z. Note that during the sorting if you move the string, move its length data stored in the first byte with it.

The output generated by the main file is as follows:

```
len:7 | trabzon
len:7 | denizli
len:5 | adana
len:9 | zonguldak

len:5 | adana
len:7 | denizli
len:7 | trabzon
len:9 | zonguldak

70.00% is used.
♣adana.....................................
```

This is just an example, the code should be generic, so that if the **#define** statements are updated your program (functions) should still work.