Introduction to C language (234126)
Moed B, Spring semester 2018
August 1st, 2018

Staff: Dr. Yechiel Kimchi, Mr. Natan Yellin, Mr. Assaf Shomer

Exam Duration: 3hrs
Assisting material: None whatsoever (written, printed, electronic, wire, divine)

Instructions:

- Fill clearly your ID and other items on the front page using pen (black/blue) only.
- You are not allowed to use any external material except pen and brain.
- Never define main(). You are not allowed to use any library function, except I/O functions,
  - Unless explicitly permitted (if you need one, implement it yourself).
- There are 4 problems, 20 pages (including a few extra, empty pages are at the end).
- Read the problem descriptions carefully, and follow the instructions.
- Write your solutions on this exam form only, at the available space. Much space does not
  mean long solutions – all solutions can be rather short.
- Clearly mark your solution, and designate your draft/scratch parts so we don’t grade them.
- You are advised to write your solutions in pen too (black/blue only), but pencils are OK.
- Write your solution clearly – if we cannot understand what you have written, it is wrong
  - You’ll be able to appeal later, if you think it was correct.
- You can implement additional helper functions as you wish, unless instructed otherwise.
- Don’t use global or static variables, nor should you use include/define commands.
- You may use a function from another section in solving a section (no circular dependency)
  - even if you have not solved the other section. Order among functions is irrelevant.
- You don’t have to check input sanity, unless explicitly instructed to do so.
- Solution’s complexity should be reasonable, but is not checked, unless explicitly required.
- You may add verbal explanations about your solution, but it is not considered for grading.
- The grade is for the code you write – not for your intentions.
  - There is no penalty for minor syntax errors that do not affect the algorithm.

We Wish You Success and Good Luck

- 1 -
Problem 1 (20 points)

Here is a complete program. Please answer the questions at the bottom.

```c
void print(int v[], int len)
{
    int i;
    for (i = 0; i < len; ++i)  printf("%d, ", v[i]);
    putchar('\n');
    return;
}

void secret(int v[], int n)
{
    int i, j;
    for (i = 1; i < n; i *= 2) {
        int k = 0;
        for (j = i/2; j < i; ++j) {
            k += v[j] += 1;
        }
        v[j] = k;
    }
    return;
}

int main(void)
{
    int a[N] = {0};  /* N is defined elsewhere */
    printf("Hello world!\n");
    secret(a, N);
    print(a, N);
    return 0;
}
```

Part A (10 pts.): What is the complexity of the function `secret()` in terms of its parameter `n`?

_________ **O(n)** ________ Please explain: __________ Carefully calculate __________

Part B (10 pts.): What is the output of the program when `N` is 12?

___________________________________________________ ________________________________________
Problem 2 (30 points)

This problem deals with a text encoding (obfuscating/ciphering) algorithm:

The function `int encode(char *s)` is given a string that contains a textual message. The text contains only letters (upper and lower-case), simple spaces (' '), and the punctuation characters period ('.'), comma (',') and question mark ('?'). It is guaranteed that the text is legal – no need to check it. The function does not alter the punctuation characters, and only alter some letters (some other letters are not altered). Upper-case letters are altered the same way as their lower-case counterparts. The alteration always replaces letters by digits. Upper-case letters that are not altered are replaced by their lower-case counterparts. The replacement is made **in place** – it modifies the original string. The function returns the length of the string.

Here is the list of the letters that are modified: a => 4 (meaning: an 'a' or an 'A' become '4'), b => 6, c => 3, h => 4, i => 1, l(e)l => 1, o => 0, q => 9, r => 8, s => 5, y => 4, z => 2.

All other letters are not altered (except that uppercase become lowercase). **Example:** The string "aBc DeFghIJ,KLlmNO." will be modified to "46c d3fg4lj,k11mn0.".

**Note:** Use a reasonably efficient algorithm. Also, a too complicated code will lose some points.

```c
int encode(char *s) {
    char key[26] = { '4','6','c','d','3','f','g','4','1','j','k','1','m','n','0','p','9','8',
                    '5','t','u','v','w','x','4','2',
    };

    int len = 0;
    for (len = 0; *s != '\0'; ++len, ++s)
        if (!isPunct(*s))   *s = key[toLower(*s) -'a'];

    return len;
}

int toLower(int c) {
    return ('A' <= c && c <= 'Z') ? c - 'A' +'a' : c; }

int isPunct(int c) {
    return c == ' ' || c == '.' || c == ',' || c == '?'; }
```
**Problem 3 (30 points)**

A divide in an array is an element whose value is 0 and whose left-hand side neighbour is negative while its right-hand side neighbour is positive. The first two parts of this problem deal with arrays that have a single divide. The third part deals with arrays that may have multiple divides. For a single divide, it is guaranteed that the 0 separates all negative numbers from all positive numbers. **Note:** A border-line case is a divide at one of the ends of the array – we do **not** deal with it here. **Example:** The array \{-2, -10, -3, -3, 0, 9, 5, 1, 1, 19\} has a single divide.

**Part 1 (10 points)**

You are required to implement the function `int divide(double a[], int n)`, which returns the index of the divide element in the parameter array `a[]`, which is guaranteed to have a single divide. The function returns the index of the divide. **Note:** Efficiency is essential. **Note:** The implementation must be standalone, using no function calls whatsoever.

```c
int divide(double a[], int n) {
    int b = 0, t = n - 1;
    while (b < t - 1) {
        int mid = (b + t)/2;
        if (a[mid] < 0) {
            b = mid;
        } else if (a[mid] > 0) {
            t = mid;
        } else {
            return mid;
        }
    }
    return a[b] == 0 ? b : t;
}
```
Part 2 (10 points)
Implement `int divide(double a[], int n)`, for the same functionality using recursion. **Note:** Efficiency is essential. **Note:** No loops are allowed.

```c
int divide(double a[], int n)
{
    if (n == 2) return a[0] != 0;
    if (a[n/2] < 0)
        return n/2 + divide(a + n/2, n - n/2);
    if (a[n/2] > 0)
        return divide(a, n/2 + 1);
    return n/2;
}
```
Part 3 (10 points)

Implement `int divide(double a[], int n)`, for an array that may have multiple divides. The function returns an index of a divide, any divide.

**Note:** An array with multiple divides satisfies the following property that is shared by a single divide: Whenever we have $i < j$, with $a[i] < 0$ and $a[j] > 0$, then there is $k$, such that $i < k < j$ and $a[k] == 0$; Therefore, the array `{ -1, 0, 2, -2, 3, -3, 0, 4 }` is illegal, because there is no $0$ between $-2$ and $3$.

The leftmost non-0 elements are all negative, and the rightmost non-0 elements are all positive.

**Example:** The array `{ -2, -10, -3, 0, 9, -4, 0, 1, -1, 0, 9, -9, 0, 5, 1, 1, 1 }` has multiple divides.

**Note:** Any efficient implementation is OK.

```c
int divide(double a[], int n)
```

Use (redirect to) any one of the functions of part-1 or part-2

(by 5th bullet from bottom, on the instructions page).
Problem 4 (20 points)

An N×N matrix m (2-dimensional array) is said to have an intersection at (i,j) if row i and column j (that intersect at (i,j)) satisfy

For All $k < N (m[i][k] = m[k][j])$

Example: The table on the right-hand side has two intersections – at (1,4), and (3,5).

In this problem you have to implement the function

```c
int hasIntersectionAt(int m[][N], int *row, int *col)
{
    int i;
    for (i = 0; i < N; ++i) {
        int j = 0;
        while (j < N) {
            if (findIntersect(m, i, j++, row, col)) return 1;
        }
    }
    return 0;
}
```

Use a reasonably efficient algorithm.
```c
int findIntersect(int m[][N], int i, int j, int*row, int*col)
{
    int k = 0;
    
    while (k < N) {
        if (m[i][k] != m[k][j]) return 0;
        ++k;
    }
    
    *row = i;
    *col = j;
    
    return 1;
}
```