Introduction to C language (234126)
Moed A, Winter semester 2016/2017
February 9th, 2017

Staff: Dr. Y. Kimchi, Mrs. Y. Erez, Mr. I. Gutter, Mr. M. Wattad

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, 16 pages (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.
- Complexity of the solution is not checked, unless explicitly mentioned that it is important.
- 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
Scratch page for problem 1
**Problem 1 (20 points)**

**Part 1 (7 Points)**

Explain (verbally) what the following function will print for the arguments \( a, b \) \((0 < a, b < 5000)\):

```c
void print(int a, int b) {
    a = a * b;
    b = a / b;
    a = a / b;
    printf("%d %d\n", a, b);
}
```

**This is swap**

---

**Part 2 (13 Points)**

Given the function, manipulate \( (\text{int } a[], \text{int } n, \text{int } z) \) and the array \( b \), defined after the function.

```c
void manipulate(int a[], int n, int z) {
    int x = 0, y = n - 1;
    while (x < y) {
        while (a[x] % z == 0) ++x;
        while (a[y] % z != 0) y--;
        swap(a + y, a + x); /* The usual swap() function */
    }
    return;
}
```

```c
int b[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

**What will be the content of \( b \) after the following activation of the function:**

```c
manipulate(b, 10, 2);
```

\[
\begin{array}{cccccccccc}
0 & 8 & 2 & 6 & 5 & 4 & 3 & 7 & 1 & 9 \\
\end{array}
\]
Problem 2 (20 points)

An array, `int a[N]`, with `N > 2`, is called a *mountain* if it satisfies:
There is `K, 0 < K < N - 1`, such that for all `i` and `j`, we have:
If `0 <= i < j <= K`, then we have `a[i] < a[j]` and if `K <= i < j < N`, we have `a[i] > a[j].`  

`K` is called the *summit* of the mountain (there is only one summit under the above conditions).

Implement the function `int summit(int a[], int n)` that takes an array `a[]` of length `n` which is a mountain, and returns the index of its summit. **Example:** `{1,2,3,4,5,3,2}` (5 is summit).

The array `a[]` is guaranteed to be a mountain. Use the most efficient algorithm that you can.

```c
int summit(int a[], int n)
{
int b = 0, t = n - 1;

while (t - b > 2) {
    int m = (b + t) / 2;
    if (a[m] < a[m + 1]) {
        b = m;
    } else {
        t = m + 1;
    }
}

return (b + t)/2;
}
```
Problem 3 (30 points)

A palindrome is a symmetric text (can be read the same from left to right and from right to left). We deal with strings that contain only lower-case English letters (no spaces, punctuations, etc.).

Examples: “abcbxcbca”, “dberttrcbd”, “xxxxx”, “a” are palindromes, but “abcabc” is not.

You are not allowed to use library functions (unless you implement them by yourself).

Part 1 (10 Points)

Implement the function \( \text{int IsPalindrome}(char *s) \) that takes a string \( s \) and returns 1 if \( s \) is a palindrome, and 0 otherwise. Do not use recursion.

You may assume that \( s \) contains only lower-case English letters.

```c
int IsPalindrome(char *s)
{
    int b = 0, t = strlen(s) - 1;
    while (t - b > 0) {
        if (s[b] != s[t]) return 0;
        ++b; --t;
    }
    return 1;
}
```

```c
int strlen(char *s)
{
    int i = 0;
    while (s[i] != '\0') ++i;
    return i;
}
```
Part 2 (10 Points)

Implement the function `int IsPalindrome(char *s)` using recursion (no loops allowed)

```c
int IsPalindrome(char *s)
{
    int n = strlen(s);
    return rec_palindrome(s, n);
}

int rec_palindrome(char *s, int n)
{
    if (n <= 1) return 1;
    if (s[0] != s[n - 1]) return 0;
    return rec_palindrome(s + 1, n - 2);
}
```
Part 3 (10 Points)

Implement the function `int TwoPalindromes(char *s)` that takes a string that contains only lower-case English letters, and returns 1 if `s` is built of two non-empty palindromes only, and 0 otherwise. **Example:** The following contain two palindromes: “abcdbazzazz” (abcdba+zzazz), “abcddcbaxx” (abcddcba+xx), “xyzyxaxa” (xyzyx+aa), “xyzyxa” (xyzyx+a), “xa” (x+a), but “abcbaabcba”, “abcdeabcba”, “a” do not.

You must use the function `IsPalindrome()` above, even if you have not implemented it. You may use recursion, but you don’t have to. Implement a reasonably efficient algorithm.

```c
int TwoPalindromes(char *s)
{
    int n = strlen(s), i = 0;
    if (n <= 1) return 0;
    if (n == 2) return 1;
    while (i < n - 1) {
        if (s[i] == s[0] && s[i+1] == s[n-1] && rec_palindrome(s,i+1) && IsPalindrome(s+i+1)) return 1;
        ++i;
    }
    return 0;
}
```
Problem 4 (30 points)

A matrix (2-dimensional array) with an internal element that is the sum of all its 8 neighbors, is called a matrix with a summit.

Example: The following matrix has a summit:
\[
m[2][3] == 36\]
is the sum of its 8 neighbors.

Implement the function int HasSummit(int m[][M], int found[])
that takes two arrays: m[N][M] is a matrix of size N*M (where N,M are #defined), and a 1-dimensional array found[K] (K is #defined to be 11).

The function HasSummit() returns 0 if m[][] does not have any summit, and in that case the array found[] is not modified. However, if m[][M] does have a summit, then, the function HasSummit() returns 1, and it modifies the array found[] so that it contains all the necessary information about one (any) summit that was found:
The first two elements contain the indices of the row and column (respectively) of the summit (the middle element), then it contains the summit value, and then it contains the surrounding 8 elements in a clock-wise order, starting from the upper-left corner. Example: For the above matrix, the function HasSummit() returns 1, and modifies the array found[] to contain the values
\[
2 3 36 1 2 3 4 5 6 7 8
\]

int HasSummit(int m[][M], int found[])
int HasSummit(int m[][M], int found[])
{
    int i, j;

    for (i = 1; i < N - 1; ++i) {
        for (j = 1; j < M - 1; ++j) {
            if (check(m, i, j)) {
                return update(found, m, i, j);
            }
        }
    }
    return 0;
}

int check(int m[][M], int i, int j)
{
    int k, l, sum = 0;

    for (k = -1; k < 2; ++k)
        for (l = -1; l < 2; ++l)
            sum += m[i+k][j+l];

    return sum == m[i][j] * 2;
}
```c
int update(int found[], int m[][M], int i, int j) {
    int ii = {-1,-1,-1,0,1,1,1,0}, k,
    found[0] = i; found[1] = j; found[2] = m[i][j];
    for (k = 0; k < K - 3; ++k) { /* K is 11 */
        found[k + 3] = m[i + ii[k]][j + jj[k]];
    }
    return 1;
}
```