Introduction to C language (234126)
Moed A, Winter semester 2017/2018
February 5th, 2018

Staff: Dr. Yechiel Kimchi, Ms. Saifun Naveh, Mr. Yoav Zuriel

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,
  o Unless explicitly permitted (if you need one, implement it yourself).
• There are 4 problems, 18 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
  o 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.
  o 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 (8 Points)

What is the time complexity of the following function (depending on the parameter n)?

```
int f1(int n)
{
    int temp = n, m = n;
    while(temp){
        n += m;
        temp /= 2;
    }
    for (int i=0; i <n; i++)
        for (int j=0; j < 8191; j++)
            printf("0");
    return n;
}
```

Explain your answer!

The value of `temp` begins at `n`, and is reduced to half of itself until it is 0.

It takes $\log_2 n$ steps. It implies that the value of `n` is increased to $n \cdot \log_2 n$.

The next, nested loop, prints $8191 \cdot n \cdot \log n$ 0's, so the complexity is $O(n \cdot \log n)$.

Part 2 (12 Points)

Given the function, `manipulate(int a[], int n)` and the array `b`, defined after the function

```
void manipulate(int a[], int n)
{
    int x = 0, y = n - 1;
    if (n == 1) return;
    manipulate(a + 1, n - 1);
    if (a[0] > a[1]) swap(a,a+1); /*The usual swap() function*/
    return;
}
```

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

what will be the content of `b` after the following activation of the function:

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

```
b = 0 8 4 1 7 5 9 2 6 3
```
**Problem 2 (30 points)**

In the course (recitations and HW) you have met a simple algorithm that discovers the decimal digits that comprise the decimal representation of that number. The algorithm is performed by successive repeated remainder (%) and division (/) operations (e.g., the digits of $x = 123$ are discovered by $a = x \% 10 = 3$, $x = x/10$, $b = x \% 10 = 2$, $x = x/10$, $c = x \% 10 = 1$, hence $cba = 123$).

It’s easy to realize that this algorithm works for any base-representation of numbers, when 10 is replaced by base value $B$; e.g., $B = 2$ for binary representation, or $B = 8$ for octal representation. In this problem you will solve that problem for any base $B$, for any $B, 1 < B < 37$. At the extreme case; i.e., $B = 36$, we need 36 different digits. This will be achieved by using the uppercase letters, A, B, …, Y, Z, as the additional 26 digits (in hexadecimal, base 16 representation, we use A,…,F).

In order to save computations, the translating function uses an array digit[36], with contents of 0, 1, … , 9, A, B, … , Y, Z, so when a digit of value $d$ is needed, its representation is digit[d], say, digit[5] is 5, digit[15] is F, and digit[35] is Z. The result always has < 65 digits.

**Your algorithms should be reasonably efficient.**

**Part 1 (10 Points)**

Implement the function `void printBase(int n, int b)` that takes an integer $n$, a base $b$, $1 < b < 37$ (you may assume that $b$ is within the correct range – no need to test it) and the function prints out the value $n$ as represented in base $b$. The implementation should not call any function, except the output functions `printf()` or `putchar()` from `stdio.h`

```c
void printBase(int n, int b) /* n > 0 */
{
    char digit[36] = “0123…89AB…XYZ”; /*Fill all chars*/
    char number[65] = {’\0’}; int i = 0;

    while (n != 0) {
        number[i++] = digit[n%b];
        n /= b;
    }

    while (i > 0) {
        putchar(number[--i]);
    }

    return;
}
```
Part 2 (10 Points)

Implement the function void printBase(int n, int b) using recursion (no loops allowed). The function printBase() need not be recursive (it may be a wrapper).

**Restriction:** The only array you are allowed to use is the array digit[36]!

void printBase(int n, int b)
{
    char digit[36] = "0123...89AB...XYZ"; /*Not a string*/

    rec_printBase(n, b, digit); /*No duplicate digit[]*/

    return;
}

void rec_printBase(int n, int b, char d[])
{
    char c = d[n % b];

    if (n == 0) return;

    rec_printBase(n/b, b, d); /*No duplicate digit[]*/

    putchar(c);

    return;
}

(*) The printing occurs on the way back (returning) of the recursion
Part 3 (10 Points)

Implement the function int base2str(char *s, int n, int b) that does exactly what the previous printBase() functions do, except that instead of printing, it writes the desired number-representation into a string (character array) that is supplied by the caller.

E.g., base2str(s, 100, 8); printf("%s",s); will print 144 (octal rep. of 100).

You may use recursion, but you don’t have to. Implement a reasonably efficient algorithm.

```c
int base2str(char *s, int n, int b)
{
    char digit[36] = "0123...89AB...XYZ"; /*Fill all chars*/
    int i = 0;
    while (n != 0) {
        s[i++] = digit[n%b];
        n /= b;
    }
    s[i] = '\0';
    reverse_str(s, i);
    return i;
}

void reverse_str(char *s, int n)
{
    char c;
    if (n <= 1) return;
    c = s[0]; s[0] = s[--n]; s[n] = c;
    reverse_str(s+1, --n);
    return;
}
```
**Problem 3 (20 points)**

An array, `int a[N]`, with `N > 2`, contains arbitrary integral numbers.

An ordered-pair is a pair of adjacent elements `a[i], a[i+1]`, in the array, such that `a[i] < a[i+1].`

The only information that we have about the array is that `a[0]` is 0, and that `a[N-1]` is 100.

Implement the function `int pair(int a[], int n)` that takes an array `a[ ]` of length `n` that fulfills the given condition, and returns the left index (i above) of an ordered-pair in the array.

**Example:** For `a[] = {0, -1, -2, 200, 200, 201, 100}` the only such indices are 2 (a[2] = -2) and 4, and any of these two is a correct result.

Use the most efficient algorithm that you can.

```c
int pair(int a[], int n)
{
    int mid = n/2;
    if (n <= 2) return 0;
    if (a[mid] > a[0]) return pair(a, mid+1);
    return mid + pair(a+mid, n-mid);
}
```

**Flip over for a loop solution**
A loop solution:

```c
int pair(int a[], int n)
{
    int bottom = 0, top = n - 1;

    while (top - bottom > 1) { /* or (bottom < top - 1) */
        int mid = (top + bottom)/2;

        if (a[mid] < a[top]) {
            bottom = mid;
        } else {
            top = mid;
        }

        return bottom;
    }
}
```
Problem 4 (30 points)

A table (2-dimensional array) represents chess-like game with the following rules: 0 represents an empty square, odd numbers, 1 – 7, represent white objects, while even numbers, 2 – 8 represent black objects. 7, 8 represent unique queen-like objects: they can move in straight lines in any direction (up, down, sideways, diagonally) provided no object stands in the way. An object of same color, just blocks their way. An object of the opposite color may be hit (but cannot be skipped by jumping over it). Example: The 7 object in the lower-right corner may hit either one of the 2’s, but none of the 4’s, and, obviously, no other black object.

Implement the function int QueenWhite(int m[][M], int rows) that takes two parameters, an array m[][M], a table of size rows*M (where M is #defined). The function returns the number of black objects that the white-queen, on the board, may hit. If no white queen is found, it returns -1. Implement a reasonably efficient algorithm.

int QueenWhite(int m[][M], int rows)
{
    int qi, qj, q = is_queen(m, rows, &qi, &qj);

    if (!q) return -1;

    return count_hits(m, rows, qi, qj);
}

int is_queen(int m[][M], int rows, int *qi, int *qj)
{
    int i, j;

    for (i = 0; i < rows; ++i)

        for (j = 0; j < M; ++j)

            if (m[i][j] == 7) {*qi = i; *qj = j; return 1;}

    return 0;
}

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>6</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>
```c
int count_hits(int m[][M], int rows, int qi, int qj)
{
    return countRow(m, qi, qj) + countColumn(m, rows, qi, qj) + countUpDiag(m, rows, qi, qj) + countDnDiag(m, rows, qi, qj);
}

int countRow(int m[][M], int qi, int qj)
{
    int count = 0, j = qj + 1;
    while (j < M && m[qi][j] == 0)
        ++j;
    count += j < M && m[qi][j]%2 == 0;
    for (j = qj - 1; j >= 0 && m[qi][j] == 0; --j)
        ; /* Empty */
    count += j >= 0 && m[qi][j]%2 == 0;
    return count;
}

int countUpDiag(int m[][M], int rows, int qi, int qj)
{
    return countDiag(m, rows, qi, qj, 1);
}

int countDnDiag(int m[][M], int rows, int qi, int qj)
{
    return countDiag(m, rows, qi, qj, -1);
}
```
```c
int countColumn(int m[][M], int r, int qi, int qj)
{
    int count = 0, i = qi + 1;

    while (i < r && m[i][qj] == 0) ++i;
    count += i < r && m[i][qj]%2 == 0;

    for (i = qi - 1; i >= 0 && m[i][qj] == 0; --i)
        /* Empty */
    count += i >= 0 && m[i][qj]%2 == 0;

    return count;
}

int countDiag(int m[][M], int r, int qi, int qj, int up)
{
    int count = 0, i = qi - up, j = qj + 1;

    while (i >= 0 && i < r && j < M && m[i][j] == 0) {
        i -= up; ++j;
    }

    count += i >= 0 && i < r && j < M && m[i][j]%2 == 0;
    i = qi + up; j = qj - 1;

    for ( ; i>=0 && i<r && j>=0 && m[i][j]==0; --j, i+=up)
        /* Empty */
    count += i>=0 && i<r && j>=0 && m[i][j]%2 == 0;

    return count;
}
```

- 15 -