Introduction To Computer – C (234112)

Spring 2013\Summer 2013

Moed B

Duration: 119 minutes. You are not allowed to exit the classroom to use the bathroom. 
External Material: you are not allowed to use any written, printed or electronic external material.

Guidelines and Instructions:

• Write your answers only on the exam form, in the intended places. Note that the given place doesn't necessarily indicate the length of the correct answer.
• The even pages of the exam form are empty. You can use them as a draft or to write your answers. Write drafts clearly, so they won't be checked.
• Write your answers tidy, clean and clearly as possible. You're allowed to use pencil and an eraser, but you must fill the title page in pen.
• In all of the questions, you're allowed to define (and implement) your own functions.
• You are not allowed to use global and/or static variables, or pre-compilation commands (include\define).
• You are not allowed to use library functions, or functions implemented in class, without implementing them yourself, unless noted explicitly in the question, excluding input/output functions.
• In every question, you are allowed to use functions defined in previous parts of the same question, even if you didn’t solve these parts, though this is not obligated.
• You don’t need to check input correction, unless explicitly noted in the question.
• You don’t need to implement the main function, and the order of writing the functions is not important.
• the complexity of the solution will not be marked, unless explicitly stated.
• It's recommended to add a written explanation of your algorithm. That explanation will not be marked.

מרצים: ד''ר קמחי יחיאל, לילה שניידרמן.
受理：ראובן שטראוס, משה גבל, מג'ד סרור, ישראל גוטר, עומר הרשנזר, רועי גלבהרט.

צוות הקורס 234112 - אביב

מרצים: איתן אביאור.
受理：רג'א ג'יריס, אלון בריפמן.

צוות הקורס 234112 – קיץ
Question 1 (25 points)
An integer is called extremely even if the sum of its digits is even.
Example: 101, -101, 1234 are extremely even. 12, -15, 12345 are not extremely even

Part 1 (10 points)
Write the following function:

```
int veryEven(int n)
```

Its parameters are an integer, n. the function will return 1 if n is extremely even, and 0 (zero) if n is not extremely even.

```
int veryEven(int n) {
    int sum=0;
    if (n<0) 
        n*=-1;
    while(n>0) {
        sum+=n%10;
        n/=10;
    }
    return (sum%2 == 0);
}
```
A **Magnificent Pair** in an array of integers, int a[N] are two adjacent numbers in an array (a[i],a[i+1]), one of them is extremely even, and the other one is not extremely even.

**Example:** for the following array:

```
11 33 22 17 16 27 31 32 36
```

The magnificent pairs are 17,16 (in indexes 3,4), 27,31 (in indexes 3,4), 31,32 (in indexes 6,7).

**Part 2 (15 points)**

Write the following function:

```cpp
int findAmazing(int a[], int n)
```

Its parameters are an array of integers, a, its first element, a[0], is extremely even, and its last element, a[n-1], is not extremely even. The function returns an index i in the array, where a[i], a[i+1] are a magnificent pair.

**Note** – the implementation of the search needs to be efficient, i.e. a serial search in which you scan the array, or part of it, an element at a time, will lose points.

```cpp
int findAmazing(int a[], int n) {
    int left=0, right=n-1, mid;
    while(left < right) {
        mid = (left+right)/2;
        if(isEven(a[mid]) && !isEven(a[mid+1]))
            return mid;
        if(isEven(a[mid]))
            left = mid;
        else
            right = mid;
    }
    /* Note that the function will never reach this point, but since the function returns int, we have to return something at the end */
    return 0;
}
```
**Question 2 (20 points)**

Write the following function:

```c
int substring(char *str, char *substr)
```

Its parameters are two strings. The function returns the number of times the second string appears in the first string.

**Example:** `substring("abcdxyzbcdbcdxyz", "bcd")` will return 3.

No special efficiency requirements, as long as you don't do completely unnecessary things.

```c
int substring(char *str, char *substr) {
    int i, count=0, j;
    char firstC;

    firstC=substr[0];
    i=0;

    while(str[i]!='/0') {
        if(str[i]==firstC) {
            for(j=1; substr[j]==str[i+j]&&substr[j]!='/0'&&
                 str[j+i]!='/0'; j++)
                if(substr[j]=='/0')
                    count++;
        }
        i++;
    }
}
```
Question 3 (30 points)

In this question we will sort strings of characters by their order in the ASCII table.

Part 1 (15 points)
Write the following function:

```c
void sortR(char *s)
```

Its parameter is a string of characters. The function sorts the string, following these guidelines:

- The entire implementation needs to be based off the "bubble sort" algorithm.
- The entire implementation needs to be recursive.
- You are not allowed to use loops
- You are not allowed to use library functions, but you can implement and use your own functions (in their implementation you're not allowed to use loops as well).
- You can implement and use a function similar to `strlen()` (without using loops), but a solution that doesn't calculate the length of the string will get a bonus of 4 points.

```c
void sortR(char *s){
    if(*s != 0) {
        bubbleSortIteration(s);
        sortR(s + 1);
    }
}

void bubbleSortIteration(char* s) {
    if(*(s + 1) != 0) {
        bubbleSortIteration(s + 1);
        if(*s > *(s + 1)) {
            swap(s, s + 1);
        }
    }
}

void swap(char* a, char* b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}
```
Part 2 (15 points)
Write the following function:

void sortLinear(char *s)

Its parameter is a string of characters. The function sorts the string, following these guidelines:

- You're allowed to traverse the string a constant (small, not dependent on the length of the string) number of times!
- You're allowed to use loops.
- You're allowed to use an extra array with a constant, predefined size, which should not depend on the length of the string.

```c
#define N 256

void sortLinear(char* s) {
    int i, j, k;
    char nrOfOccurrencesOfCharacter[N] = {0};

    for(i = 0; s[i] != 0; i++) {
        nrOfOccurrencesOfCharacter[s[i]]++;
    }

    i = 0;
    for(j = 0; j < N; j++) {
        for(k = 0; k < nrOfOccurrencesOfCharacter[j]; k++) {
            s[i++] = j;
        }
    }
}
```
Question 4 (25 points)

Write the following function:

```c
int colRow(int a[][N])
```

Its parameter is a square matrix of size NxN (N is defined in a define statement). The function returns the value i, if there's an i'th row and an i'th column that are completely identical, when you compare the elements of the i'th column from top to bottom, and the elements of the i'th row from right to left. If there are no such row and column, the function will return -1 (minus 1)

Example: for this matrix, the function will return 3.

```
5 0 7 1 6
2 1 9 2 8
3 2 5 3 4
5 2 3 2 1
6 6 6 5 6
```

And for this matrix, the function will return -1.

```
5 0 7 1 6
2 1 9 2 8
3 2 5 3 4
5 4 3 2 1
6 6 6 5 6
```

```c
int colRow(int a[][N])
{
    int i, j;
    for(i=0; i<N; i++)
    {
        for(j=0; (j<N) && (a[i][j]==a[i][N-1-j]) ;j++)
            if(j==N) return i;
    }
    return -1;
}
```