מבוא למחשב – שפת C
semester Spring 2011
11221

Exam Date: 8 March 2012
Duration: 1 hour.
No leaving the room.

Aid Material:
No use of any written, printed or electronic aid.

Guidelines:
• Check that there are 16 questions (3 pages) in the exam, including this page. Pages 3-4 are blank and intended for corrections.
• Write the answers only on the exam sheet, in the designated places. Pay attention that the designated space does not necessarily indicate the length of the answer.
• Even numbered pages are blank. They can be used as practice pages and for writing your answers. Mark your corrections clearly so that they are not examined.
• Write clearly, neatly and orderly. It is recommended to use a pen and a pencil.
• In all questions, you are allowed to define auxiliary functions as you wish.
• Do not use global or static variables, or pre-compiled functions (define/include) unless specified otherwise in the question.
• Do not use unless implemented, library functions, or functions implemented in class other than those specifically mentioned in the question, except input-output functions.
• In all questions, you are allowed to use any previous paragraphs of the same question, even if you didn’t solve them. No need to solve them.
• You do not need to check the data input unless specified otherwise in the question.
• You do not need to run `main`, and the order of writing the functions is not important.
• It is recommended to add a description in words of the algorithms. The description is not marked.

Instructors:
Dr. Ani Veil – Head, Dr. Daniel Kotler, Dan Rabin.
Tutors:
Schneidman Lila (Head Tutor) Steinfield Israel, Borodin Gregory, Gotsman Rani, Kraus Neama, Glick Asaf, Maimir Cohen, Hozar Nir, Stern - Pertush Namir.

Good luck!
Question 1 (35 Points)
Part 1 (15 Points)
Write the following function:

```c
int is_prime(int n) {
    int i;
    if (n < 2) return 0;
    for (i = 2; i < n; i++)
        if (n % i == 0) return 0;
    return 1;
}
```

its parameter is a number, \( n \). the function checks whether \( n \) is prime or not, and returns 1 if it is prime, and 0 otherwise.

Reminder – a prime number is a natural number (whole and positive), greater than one, that can't be presented as a product of two natural numbers that are lower than itself.

The efficiency of the solution will not be checked.
**Part 2 (20 Points)**

In this part we will check Goldbach's Conjecture, which says every even number greater than 2 can be written as a sum of two prime numbers.

Write the following function:

```c
int find_primes(int n, int *p1, int *p2)
```

It's parameters is a whole, even number, `n`, which is greater than 2. If `n` can be presented as two prime numbers, the function returns 1. Otherwise, it returns 0. If `n` can be presented as two prime numbers, the function will return the numbers with the pointers `p1` and `p2`.

If there are several solutions, the function can return any of them.

**Efficiency of the solution will not be checked.**

You can assume `p1` and `p2` point to a legal address in the memory.

```c
int find_primes(int n, int *p1, int *p2) {
    int i;
    for (i = 2; i < n; i++) {
        if (is_prime(i) && is_prime(n - i)) {
            *p1 = i;
            *p2 = n - i;
            return 1;
        }
    }
    return 0;
}
```
**Question 2 (30 Points)**

**Part 1 (15 Points)**

Write the following function:

```c
int does_generate(char *s, char *sub, int len)
```

its parameters are a legal string, s, an array of chars, sub, and its length, len. The function will return 1 if a concatenation of the char array sub makes the string s. The function will return 0 otherwise.

**Examples:**

1. For s="abcabcabcabc", sub=abc, len=3, the function will return 1, because concatenating abc for times makes the string s.
2. For s="abcabcabcab", sub=abc, len=3, the function will return 0, because no concatenation of abc will make the string s.

You can assume the input is only chars and numbers.

**Remainder – you are not allowed to use library functions without implementing them.**

```c
int does_generate(char *s, char *sub, int len) {
    int i = 0;
    int length_s;

    while (s[i] != '\0') {
        if (s[i] != sub[i % len]) {
            return 0;
        }
        i++;
    }

    length_s = i;
    return ((length_s % len) == 0);
}
```
**Part 2 (15 Points)**

A "preface" of a string is an array of characters in the beginning of a string. An example – the string "abca" has four different "prefaces", which are:

- a – the first character
- ab – the first two characters
- abc – the first three characters
- abca – the first four characters

Write the following function:

```c
int shortest_str(char *s)
```

Its parameters are a legal, non-empty string s. The function returns the length of the shortest "preface", which concatenation makes the string s. (For more details, look at Part 1 of this question). You can assume the input is only chars and numbers.

**Remainder – you are not allowed to use library functions without implementing them.**

**Examples:**
1. For s="abcabc", the function will return 3, because abc is the shortest preface that makes the string s.
2. For s="abcab", the function will return 5, because abcab is the shortest preface that makes the string s.

**What is the time complexity of the solution you've written (including Part 1 if you've used it)? _____ (3 points)**

If you’ve solved part 1 in a complexity of $O(N)$, the complexity is $O(N^2)$. If you’ve solved part 1 in a complexity of $O(N^2)$, then the complexity is $O(N^3)$. The analysis assumes N is the length of the string s.

```c
int shortest_str(char *s)
{
    int i = 1;
    while (!does_generate(s,s,i)) i++;
    return i;
}
```
**Question 3 (35 Points)**

The **Stability** of an array is defined as the difference between the sum of the first half of its elements and the sum of the second half of its elements.

In an array with an odd number of elements, the middle element is included in both halves.

**Example 1:** for the array

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

The sum of the first half is a[0]+a[1]=5

The sum of the second half is a[2]+a[3]=7

Therefore the stability is 5-7=-2

**Example 2:** for the array

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

The sum of the first half is a[0]+a[1]+a[2]=11


Therefore the stability is 11-10=1

**Part 1 (15 points)**

Write the following function:

```c
int stability(int a[], int n) {
    int i, j, stab = 0;
    for (i = 0, j = n - 1; i < j; i++, j--) {
        stab += (a[i] - a[j]);
    }
    return stab;
}
```

Its parameters are an array of integers, `a`, and its length, `n`. The function returns the stability of the array.

**You are not allowed to use recursion at all.**
Part 2 (20 points)

Write the following recursive function:

```c
int rec_stability(int a[], int n)
```

Its parameters are an array of integers, `a`, and its length, `n`. The function returns the stability of the array.

You are not allowed to use loops at all.

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