מבוא למחשב – שפת C
ר"א סמסטר חורף תשע"ב
9 בפברואר 2012

 Bahrain
לומר: רון קימל, שניידרמן לילה, שטיינפלד ישראל,
תרגマー: איזף, אוזר ניר, גוטסמן רנית, זינגר עדי,
רומן, גלזר אסף, בינר ד"ר, שילט אסף, בורו גרגורי

페이지 234112

47

เฉך

之內容

- 1 -
**Question 1 (30 Points)**

**Part 1 (15 Points)**

Write the following function:
```c
int similar(char a[], char b[], int n)
```
Its parameters are: two arrays of uppercase and lowercase characters, `a` and `b`, and their identical size, `n`.

The arrays don't necessarily end with '0', therefore their length is given.

The function should return the number of identical characters in the two arrays. Lowercase and uppercase characters are indistinguishable (meaning 'A'='a'), and the order of appearance of characters matters.

**Examples:**
1. For `n=3`, `a="abc", b="abc"`, the function will return 3.
2. For `n=5`, `a="Hello", b="hello"`, the function will return 5.
3. For `n=5`, `a="zllmo", b="Hello"`, the function will return 2.
4. For `n=3`, `a="bca", b="abc"`, the function will return 0.
5. For `n=5`, `a="hello", b="zzzoz"`, the function will return 0.

```c
int similar(char a[], char b[], int n){
    int i, count = 0;
    char A, B;
    for (i = 0; i < n; i++)
    {
        B = (b[i] >= 'A' && b[i] <= 'Z') ? b[i] : b[i] - 'a' + 'A';
        if (A == B)
        {
            count++;
        }
    }
    return count;
}
```
Part 2 (15 Points)
Write the following function:

Int substr(char a[], char b[])

Its parameters are: two legal strings \texttt{a,b} (which end with '0'). The function returns the first index in the string \texttt{a}, in which the string \texttt{b} starts. If the string \texttt{b} is not inside the string \texttt{a}, the function will return -1. Uppercase and lowercase are indistinguishable.

Examples:
1. For \texttt{a="Hello World"}, \texttt{b="WORLD"} the function will return 6.
2. For \texttt{a="Hello world"}, \texttt{b="course"} the function will return -1.
3. For \texttt{a="COurse"}, \texttt{b="course"}, the function will return 0.
4. For \texttt{a="coursd"}, \texttt{b="course"}, the function will return -1.
5. For \texttt{a="coursd"}, \texttt{b="ur"}, the function will return 2.

```c
int substr(char a[], char b[]){
    int i;
    int a_len = strlen(a);
    int b_len = strlen(b);
    for (i = 0; i <= a_len - b_len; i++) {
        if (similar(a + i, b, b_len) == b_len) {
            return i;
        }
    }
    return -1;
}
```

Implementation of strlen:

```c
int strlen(char a[]){
    int i;
    for (i=0; a[i] != '\0'; i++);
    return i;
}
```
Question 2 (40 Points)
Part 1 (20 Points)
Write the following function:

Void print_reorder(int a[], int n)

Its parameters are an array of integers a, and its size n. the function prints the elements of the array from largest to smallest, without repeats.
Spaces and line breaks in the printed text don't matter. You can alter the array.
Example – for n=6 and the array a:

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>
```

The function will print five numbers: 9 8 6 4 3

```c
void print_reorder(int a[], int n){
    int i, j, num, maxVal, maxIndex, temp;
    for(i = 0; i < n; i++){
        maxVal = a[i];
        maxIndex = i;
        /* Find largest value in array, right of current index */
        for(j=i+1; j<n; j++){
            if(a[j]>a[maxIndex]){  
                maxVal = a[j];
                maxIndex = j;
            }
        }
        /* Swap with this found largest value */
        temp = a[i];
        a[i] = a[maxIndex];
        a[maxIndex] = temp;
    }
    num = a[0];
    printf("%d ", num);
    /* Print the sorted values in order */
    for(i = 1; i<n; i++){
        /* Only non-repeat values */
        if(a[i] != num)
            printf("%d ", a[i]);
        num = a[i];
    }
}
```
Part 2 (20 Points)

Write the following function:

Void print_location(int a[], int n)

Its parameters are: an array of positive integers a, and its size n. The function prints the new index of every element in the array after it's sorted from largest to smallest.

If a number appears in the array several times, then the first appearance in the array will be the first appearance in the sorted array, etc.

Spaces and line breaks in the printed text don't matter.

Assume the number of elements n is smaller than N, an externally defined constant in #define.

Example: for n=6 and the array a:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

The function will print 6 numbers: 2 4 1 5 3 0

Because if the array was sorted, the order of the numbers would be the one in the bottom array – 6 would've been in cell 2, 3 would've been in cell 4, etc. therefore the first number printed is the new location of a[0], which is index 2. The second number printed is the new location of a[1] which is index 4, etc.

void print_location(int a[], int n) {
    int i, swapped = 1;
    int ind_arr[N];
    for (i = 0 ; i < n ; i++){
        ind_arr[i] = i;
    }
    for (i = n ; i > 1 ; i--){
        if (!bubbleExtra(arr,ind_arr,i)) break;
    }
    print_reverse_permutation(ind_arr,n);
}

void print_array(int arr[], int n){
    int i;
    for (i = 0 ; i < n ; i++){
        printf("%d",arr[i]);
        if (i < n-1) printf(" ");
    }
    printf("\n");
}

rest of the functions:
void print_reverse_permutation(int perm[], int n){
    int a_rev[N];
    int i;
    for (i = 0 ; i < n ; i++){
        a_rev[perm[i]] = i;
    }
    print_array(a_rev,n);
}

void swap (int* pA, int* pB){
    int temp = *pA;
    *pA = *pB;
    *pB = temp;
}

int bubbleExtra(int arr[],int indexArr[], int n){
    int i, swapped = 0;
    for (i = 0 ; i < n-1 ; i++){  
        if (arr[i] < arr[i+1]){          
            swap(&arr[i],&arr[i+1]);
            swap(&indexArr[i],&indexArr[i+1]);
            swapped = 1;
        }
    }
    return swapped;
}
Question 3 (30 Points)
Part 1 (15 Points)
Write the following function:
Int words(char s[])
Its parameter is a legal string s (ends with '\0'). The function returns the number of words in the string.
A word consists of only lowercase and uppercase characters. Between words there will be one or more spaces.
Note that there could be spaces in the beginning and the end of the string.
Examples:
1. For the string "Hello" the function will return 1.
2. For the string "Hello world" the function will return 2.
3. For the string " Hello world " the function will return 2.
4. For the string " Hello world " the function will return 2.

Recursion is not allowed at all in this part.

```c
int words(char s[]){
    int n = 0;
    do
        n += (*s!='\0' && *s!=' ' && (*(s+1)==' ' || *(s+1)=='\0'));
    while (*(s++));
    return(n);
}
```
Part 2 (15 Points)
Write the following recursive function:
```c
int rec_words(char s[])
```
Its parameter is a legal string `s` (ends with `\0`). The function returns the number of words in the string.

A word consists of only lowercase and uppercase characters. Between words there will be one or more spaces.

Note that there could be spaces in the beginning and the end of the string.

Examples:
1. For the string "Hello" the function will return 1.
2. For the string "Hello world" the function will return 2.
3. For the string "Hello world" the function will return 2.
4. For the string "Hello world" the function will return 0.

Loops are not allowed at all in this part.

```c
int rec_words(char s[]){
    /* ***********   1   *************** */
    if(s[0] == 0) return 0;
    return rec_words(s+1) + (s[0] != ' ' && (s[1] == ' ' || s[1] == 0));
    /* ************   2   *************** */
    if(s[0] == 0) return 0;
    if(s[0] != ' ' && s[1] == ' ')
        return 1 + rec_words(s+1);
    if(s[0] != ' ' && s[1] == 0)
        return 1 + rec_words(s+1);
    return rec_words(s+1);
    } /* ************   3   *************** */

    int result;
    if(s[0] == 0) return 0;
    result = rec_words(s+1);
    if(s[0] != ' ' && (s[1] == ' ' || s[1] == 0))
        result++;
    return result;
    */
    /* **************   4   ************** */
} /* *************** */
```