DALHOUSIE UNIVERSITY
Inspiring Minds

CSCI 2132 — Software Development

Assignment 7

Due: *Friday, Nov 30, 2018 by midnight*
Worth: 65 marks (= 10 + 15 + 10 + 15 + 15)
Instructor: Vlado Keselj, CS bldg 432, 902-494-2893, vlado@dnlp.ca

Assignment Instructions:

Solutions to this assignment must be submitted through SVN, in a similar way as for the previous assignment, unless specified differently by a question. The solutions to practicum questions must also be submitted via the practicum web site.

1) (10 marks) Briefly describe the `malloc` function, how it works, and its time efficiency. Describe briefly the `free` function, how it works, and its time efficiency.

Record and submit your answer via SVN in the file: *CSID*/`a7/a7q1.txt` where *CSID* is your CSID and it is your main SVN directory for the course.

2) (15 marks) Submit the answer to this question as a C file named `a7q2.c`, which includes definitions of the required functions below. The file must be submitted via SVN in the pathname: *CSID*/`a7/a7q2.c` where *CSID* is your CSID and it is your main SVN directory for the course.

The file must be valid C code that can be compiled and tested with other code.

Let us assume that the following structure is used to create a linked list:

```c
struct node {
  int val;
  struct node *next;
};
```

a) (5 marks) Write a recursive function `list_sum_r` with the following prototype:

```c
int list_sum_r(struct node *head);
```

which returns 0 if the parameter `head` is NULL, and otherwise it returns the sum of all elements of the list that starts with the `head` pointer.

b) (5 marks) Write a non-recursive version of the function in a), called `list_sum_i` with the same prototype:

```
int list_sum_i(struct node *head);
```

and the same result.

c) (5 marks) Write a function `list_reverse` with the following prototype:

```
struct node * list_reverse(struct node *head);
```

that reverses the list and returns the pointer to the new head of the list.

3) (10 marks) Make sure that you complete the Lab 9 as required. There is a set of files that need to be submitted in this lab in your GitLab project directory "`git/csci2132/lab9`".

4) (15 marks) Your task in this question is to write a C program named `I.c` according to the specifications below. You will submit the program using SVN in the pathname *CSID*/`a7/I.c` where *CSID* is your CSID and the name of your main SVN directory for the course. You will also need to submit the program as a part of the Practicum 5. The Practicum 5 link is available at the course web site.

Problem description: You should be familiar with the program sort that can sort lines of a textual file in various ways using the keys. A simple sort will just sort lines alphabetically, or to be more precise in the ASCII order using the characters in the line and using the lexicographic order.

In some situations, we may want to keep several lines together and we do not want to break them in sorting. For example we can indicate that a line continues into the next line by using the backslash character at the very end of the line. Your task is to write a sorting program that sorts lines in such way that the continued lines are not broken. The continued lines should be sorted and compared to each other as long strings using the function `strcmp`.

Since there is not specified limit on the number of lines or the length of one line, it is recommended that you use `malloc` and `realloc` functions to allocate for each continued line exactly as much memory as required. You can read all continued lines into a linked list, then sort the list, and print them from the list.

Input: The standard input includes an arbitrary text. There are no specified limitations to the number of of lines or the length of lines.

Output: Your program must keep continued lines in a group, where any line continues to the next line if there is the backslash character at the very end of the line, and the program must printed the continued lines in a sorted order. Of course, the continued line can be mixed with single lines that are not part of any group.

You can find more details and sample input and output of the problem at the practicum site.

a) [10 marks] Functionality: To satisfy this requirement, your program must satisfy the program requirements as specified and pass the practicum testing.

b) [5 marks] Implementation: To receive these marks, the program must be implemented in the following way, and it will be marked by a marker. You are required to write a head comment that includes your name, date of the program, the course and assignment number, and a short description of the program. The program should follow the organization presented in the class.

The program should use an appropriate indentation. Each block should have a consistent indentation, which can be any number of spaces you choose, between 2 and 8. You can choose a reasonable brace placement style (for example, see `https://en.wikipedia.org/wiki/Indentation_style`). K&R style would be recommended, but any other known style is acceptable as well. Your program lines should not be longer than 80 characters.

The program should use allocation functions (such as `malloc` and `realloc`,) to be flexible about line length and allocate the appropriate amount of memory. It is preferable that you use linked list and merge sort, but it is acceptable if you use array-based sorting, as long as you do not use more memory than minimally required by the program.

When allocating memory, you should start with allocation of certain memory size, and whenever you need more memory, you should use realloc to double the memory size. Once you know the exact memory size that you need, you should again use realloc to shrink the block to the exactly needed memory size.

5) (15 marks) Your task in this question is to write a C program named `J.c` according to the specifications below. You will submit the program using SVN in the pathname *CSID*/`a7/J.c` where *CSID* is your CSID and the name of your main SVN directory for the course. You will also need to submit the program as a part of the Practicum 5. The Practicum 5 link is available at the course web site.

Problem description: We covered in class generation of all permutations of numbers 1, 2, ...n. However, as we noticed, the generated permutations were not lexicographically ordered. For example, for $n = 4$ the generated permutations were:

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 3 2
...
```

while the lexicographically ordered permutations would be:

```
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
...
```

If you are not familiar with lexicographic order, we can define it in the following way: We compare two sequences of numbers (or letters, or other comparable elements), from the start and keep doing it while the sequence elements are equal. When we come across the first elements that are different, we decide that sequences compare in the same way as those two elements. In the above example, when we compare sequences (1,4,3,2) and (1,4,2,3), we see that the sequences differ first time at the third position, and the sequence with 2 at the third position should come before the sequence with 3 at third position; i.e., the order should be (1,4,2,3) and then (1,4,3,2).

You should write a program that generates permutations in the lexicographically ordered way. The program must have two modes of operation, in one the program must produce all permutations, and in other it must produce only the permutation that comes at certain position in the permutations list.

Method: You should base your method on the algorithm covered in class. If you remember the algorithm (one version) from Lecture 19 was:

```
1: IF k == n-1 THEN print A
2: ELSE
3:    Permute(A, k+1, n)
4:    FOR i = k+1 TO n-1 DO
5:       swap A[k] with A[i]
6:       Permute(A, k+1, n)
7:       swap A[k] with A[i]    /* swap back */
```

In order to get lexicographic order, in the step 5, you should not just swap $A[k]$ and $A[i]$, but you need to rotate all elements from $A[k]$ to $A[i]$, by moving $A[k+1]$ into $A[k]$, $A[k+2]$ into $A[k+1]$, and so on, until you move $A[i]$ into $A[i-1]$, and finally move the old value of $A[k]$ into $A[i]$. Similarily in the step 7 you need to reverse this rotation.

Input:

Each line of input contains two integers m and n. The input ends with integers 0 and 0. Other than the final two numbers, n is always the positive integer. If the first number m is 0, you program must print all permutations of numbers 1, 2, ..., n in the lexicographic order. If the number m is positive, then your program must print only the m'th permutation

of numbers 1, 2, ..., n in the lexicographic order. If the number m is so large that you run out of permutation, your program should not print anything.

Output: For each par of numbers m and n in the input, your program should first produce the line that looks as follows:
If n is 0, the line should be:
`End of output`
otherwise, if m is 0, the line should be:
`Permutations of` n`:`
otherwise, the line should be:
`Permutation of` n `number` m`:`

After that, if $m = 0$ and $n > 0$ the program must produce all permutations as specified, one permutation per line; or if $m > 0$ only the specified permutation, or no permutations at all if m is larger than the number of permutations, and if both m and n are zero, then the program should end.

You can find more details and sample input and output of the problem at the practicum site.

a) [10 marks] Functionality: To satisfy this requirement, your program must satisfy the program requirements as specified and pass the practicum testing.

b) [5 marks] Implementation: To receive these marks, the program must be implemented in the following way, and it will be marked by a marker. You are required to write a head comment that includes your name, date of the program, the course and assignment number, and a short description of the program. The program should follow the organization presented in the class.

The program should use an appropriate indentation. Each block should have a consistent indentation, which can be any number of spaces you choose, between 2 and 8. You can choose a reasonable brace placement style (for example, see
`https://en.wikipedia.org/wiki/Indentation_style`). K&R style would be recommended, but any other known style is acceptable as well. Your program lines should not be longer than 80 characters.

The program should use the above method, although alternative methods are also acceptable if they are correct and equally or more efficient. If the input requires that you print only one permutation, the program should not continue generating unnecessary permutations internally, but move to the next case.