DALHOUSIE UNIVERSITY
Inspiring Minds

Fall 2017 (Sep5-Dec5)
Faculty of Computer Science

CSCI 2132 — Software Development

Assignment 5

Due: *Monday, Oct 30, 2017 by 11:59 p.m.*
Worth: 75 marks
Instructor: Vlado Keselj, CS bldg 432, 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.

The first question refers to the Lab 6 and must be submitted in the SVN directory *CSID*`/lab6`, where *CSID* is your FCS userid.

The answers to all other questions must be submitted in your SVN directory: *CSID*`/a4` where *CSID* is your CS userid. Remember that you need to add to `svn` the directory as well as any files that you want submitted.

1) (15 marks) Make sure that you complete the Lab 6 as required. There are a set of files that need to be submitted in this lab in your SVN directory for the course named "*CSID*`/lab6`" where *CSID* is your CS userid.

You must make sure that the location of this directory is exactly as specified, and that the directory and all required files are submitted properly to SVN before the deadline.

The following files need to be submitted: `testprintf.c` (2 marks), `emacs-function` (2 marks), `testprintf2.c` (2 marks), `testscanf.c` (2 marks), and `phone.c` (7 marks).

2) (20 marks) Explain what the following C functions do (at a higher level of abstraction). Submit your answers in a plain-text file named `a5q2.txt`.

You can assume that in all cases the input argument `n` is always positive, i.e., $n > 0$. Briefly explain the function in examples a) and b) and show the effect by using an small example with $n = 4$.

a) (5 marks)

```
void f_a(int n, int a[n][n], int b[n][n]) {
```

```
    int i, j;
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
          b[j][n-1-i] = a[i][j];
}
```

b) (5 marks)

```
int f_b(int n, int a[n]) {
  int m;
  if (n==1)
    return a[0];
  m = f_b(n-1, a);
  if (m > a[n-1])
    return a[n-1]
  else
    return m;
}
```

c) (10 marks) Explain the function `f_c(n, a[])` where `n` is a positive integer and `a` is an array.

```
void f_c1(int i, int j, int a[]);

void f_c(int n, int a[n]) {
  f_c1(0, n-1, a);
}

void f_c1(int i, int j, int a[]) {
  int tmp;
  if (i >= j) return;
  tmp = a[i];
  a[i] = a[j];
  a[j] = tmp;
  f_c1(i+1, j-1, a);
}
```

In the part c), draw the stack activation records when function is executed for `ret = f_c(5, a);` where the array `a = { 1, 2, 3, 4, 5 }`.

Since you are to submit your answer as an plain-text file, use so-called ASCII graphics, which should look like the following, for **some other ("imaginary")** functions `g` and `h`:

```
+------------------+
| h(x=3,y=2)       |
+------------------+
| h(x=5,y=3)       |
+------------------+
| g(n=3, a={1,2,3}) |
| return a={0,0,0}  |
+------------------+
| main             |
+------------------+
```

In the above example, the function `g` has an array argument, and for it we show the initial value (`{1,2,3}`) and also the value at the return from the function (`{0,0,0}`).

3) (10 marks) Write a C program named `a5q3.c` which removes adjacent duplicates from a list of integers.

The program reads the standard input expecting a list of integers separated by whitespace. After reading each integer, it should print it on a separate line by itself, unless the integer is equal to the integer read before. This means that the program behaves in the same way as the Unix utility `uniq` but for integers instead of lines.

You **must not** use an array to store integers, but print them as soon as they are read to the standard output.

The sample input and output files are given below. They will also be made available on the course web site with the assignment, and in the `~prof2132/public` directory on bluenose.

Sample Input

```
1 4 5 4 5 4 4 4
4
4
4
-3
-3
2  1   1 1 0 0
```

Sample Output

```
1
4
5
```

```
4
5
4
-3
2
1
0
```

4) (20 marks) Write a C program named `a5q4.c` which reads standard input and reports at the end the counts and frequencies of all letters and the space character. The program must normalize the letters, i.e., must count as the same uppercase and lowercase letters. The count is the number of times a letter occured in text, while frequency is a fraction of time the letter occured divided by the total number of letters. (Be careful to avoid dividing by zero!)

Any characters between letters, i.e, between words, are counted as a simple space character, which is printed as an underscore ('_') character. You should also assume that there is a space character at the very beginning and the very end of the text.

If the input is empty or if it does not contain any letters, the output should be the same as only one space was processed.

The program output must be as in the sample output shown below. You should align the numbers as shown in the table, and the frequency is printed in percentages rounded to exactly four decimals. (The percentages mean that the actual fraction is multiplied with 100.)

Sample input and output are given below, provided on the web site, and in the directory `~prof2132/public` on bluenose.

Make sure that your program produces format as given in the sample output.

Sample Input:

```
i.e., must count as the same uppercase and lowercase letters.
The count is the number of times a letter occured in text, while
frequency is a fraction of time the letter occured divided by the
total number of letters.  (Be careful to avoid dividing by zero!)

Any characters between letters, i.e, between words, are counted as a
simple space character, which is printed as an underscore (‘\verb,_,’)
character.
```

Sample Output:

```
Letter occurences freq.
------------------------
   _       71    18.25 %
   a       24     6.17 %
   b        8     2.06 %
   c       21     5.40 %
   d       13     3.34 %
   e       53    13.62 %
   f        6     1.54 %
   g        1     0.26 %
   h       11     2.83 %
   i       19     4.88 %
   j        0     0.00 %
   k        0     0.00 %
   l       10     2.57 %
   m        7     1.80 %
   n       16     4.11 %
   o       16     4.11 %
   p        5     1.29 %
   q        1     0.26 %
   r       27     6.94 %
   s       19     4.88 %
   t       33     8.48 %
   u       12     3.08 %
   v        4     1.03 %
   w        6     1.54 %
   x        1     0.26 %
   y        4     1.03 %
   z        1     0.26 %
------------------------
```

5) (10 marks) Make sure that your SVN repository is clean. This means that you should have only the directories and files that are required to be saved there. If you have a strong reason to include some extra files, describe them in a README file and submit this README file as well.

If you would like to keep some extra files in the SVN repository for your own archive independently of labs and assignments, you can store them in a directory called `misc`. Otherwise, your SVN repository should only contain the directories named: `a2`, `a3`, `a4`, `a5` and so on; and `svnlab`, `lab4`, `lab6`, and similar.

One additional directory will be created by the instructor with the name `feedback` which may contain more information about your SVN repository and other feedback. You need to

run the command '`svn update`' from time to time to check this directory.

These SVN directories should also contain only the files required to be submitted. Have in mind that you can have additional files there as long as they are not submitted to the SVN repository.