Programming Design, Spring 2016
Homework 9

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Please upload one PDF file for Problem 1 and two CPP files for Problems 2 and 3 to PDOGS at http://pdogs.ntu.im/judge/. Each student must submit her/his individual work. No hard copy. No late submission. The due time of this homework is ***2:00 am, April 27, 2016*** (NOT April 25). Please answer in either English or Chinese.

Before you start, please read Sections 7.10 and 22.9–22.13 of the textbook.[1]

The TA who generates the testing data and grades this homework is ***Chien Huang***.

Problem 1

(20 points; 5 points each) Please answer the following questions:

(a) Recall that we may count the number of spaces in a given sentence:

```
char a[100] = {0};
while(cin.getline(a, 100))
{
  int i = 0;
  int spaceCount = 0;

  while(a[i] != '\0')
  {
    if(a[i] == ' ')
      spaceCount++;
    i++;
  }
  cout << spaceCount << "\n";
}
```

Modify this program to print out the number of sequences of consecutive spaces in a given sentence. For example, "`this  is a     book`" contains 8 spaces but only 3 sequences of consecutive spaces.

(b) Given a string which contains fewer than 100 characters, I would like to replace all the "ntu" to "NTU". For example, "I like ntu, and I like ntuim." should become "I like NTU, and I like NTUim." I wrote the following program:

```
char a[100] = {0};
cin >> a;
char* p = strstr(a, "ntu");
while(p != nullptr)
{
  strcpy(p, "NTU");
  p = strstr(p, "ntu");
}
cout << a << "\n";
```

However, it does not work. Please help me modify this program so that after the loop, `a` contains the desired string.

[1] The textbook is *C++ How to Program: Late Objects Version* by Deitel and Deitel, seventh edition.

(c) Consider the following program

```
#include <iostream>
#include <cstring>
using namespace std;

const int CNT = 4;
const int LEN = 10;
void sortName(char name[][LEN], const int CNT);

int main()
{
  char name[CNT][LEN] = {"John", "Mikasa", "Eren", "Armin"};
  sortName(name, CNT);
  for(int i = 0; i < CNT; i++)
    cout << name[i] << " ";
  return 0;
}
```

and the definition of `sortName`:

```
void sortName(char name[][LEN], const int CNT)
{
  for(int i = 0; i < CNT; i++)
  {
    for(int j = 0; j < CNT - i - 1; j++)
    {
      if(strcmp(name[j], name[j + 1]) > 0)
      {
        char temp[LEN] = {0};
        strcpy(temp, name[j]);
        strcpy(name[j], name[j + 1]);
        strcpy(name[j + 1], temp);
      }
    }
  }
}
```

Count the exact number of times that `strcpy` is invoked.

(d) Continue from Part (c). Consider another implementation of `sortName`:

```
void sortName(char* name[], const int CNT)
{
  for(int i = 0; i < CNT; i++)
  {
    for(int j = 0; j < CNT - i - 1; j++)
    {
      if(strcmp(name[j], name[j + 1]) > 0)
      {
        char* temp = name[j];
        name[j] = name[j + 1];
        name[j + 1] = temp;
      }
    }
  }
}
```

To invoke this implementation, we need to create a pointer array pointing to the names:

```
char name[CNT][LEN] = {"John", "Mikasa", "Eren", "Armin"};
char* ptr[CNT] = {0};
for(int i = 0; i < CNT; i++)
  ptr[i] = name[i];

sortName(ptr, CNT);

for(int i = 0; i < CNT; i++)
  cout << ptr[i] << " ";
```

Explain how the pointer swapping works. Is this implementation more time-efficient than that in Part (c)? Explain.

Problem 2

(40 points) A restaurant is open seven days a week. The numbers of waiters/waitresses needed each day are different. To facilitate our discussion, let's call waiters/waitresses agents. There are two types of agents: full-time and part-time. Each full-time agent works for five consecutive days and then have two days off. Each part-time agent works for one day in a week.[2] A full-time agent is paid F dollars per week, and a part-time agent is paid P dollars per day. While part-time agents are more flexible to schedule, they are more expensive, i.e., $F < 5P$. Given the number of agents needed for each day, the question is to determine a hiring and scheduling plan for full-time and part-time agent to minimize the total cost while satisfying all demands.

As an example,[3] suppose the demands are given in Table 1, where day 1 means Monday, day 2 means Tuesday, ..., and day 7 means Sunday. One feasible schedule is the following: 18 agents start to work on day 1, 6 on day 3, 3 on day 5, and 7 on day 6. Let x_i be the number of full-time agents starting to work on day i, $i = 1, ..., 7$, this schedule can be expressed as $x = (18, 0, 6, 0, 3, 7, 0)$. In total 34 agents are required. As we may see, all the demands are satisfied, though on some days there are some excess supply. Suppose the weekly wage for a full-time agent is \$10000, the total cost of this plan is \$340000.

Day	1	2	3	4	5	6	7
Demand	18	12	24	19	27	16	14

Table 1: An example demand distribution

Suppose that we may also hire part-time agents at a daily wage \$3000. Let y_i be the number of part-time agents working on day i, $i = 1, ..., 7$, and $y = (y_1, y_2, ..., y_7)$ be a schedule of part-time agents, then $y = (0, 0, 0, 0, 0, 2, 0)$ means hiring two part-time agents on day 6. This allows us to modify the plan for full-time agents to $x = (18, 0, 6, 0, 3, 5, 0)$, and the total cost becomes \$320000 + \$6000 = \$326000. In fact, if we hire seven and five part-time agents on days 6 and 7 and set $y = (0, 0, 0, 0, 0, 7, 5)$, we will only need 27 full-time agents with $x = (18, 0, 6, 0, 3, 0, 0)$. The total cost becomes \$270000 + \$36000 = \$306000.

While a polynomial-time algorithm exists for finding an optimal schedule, in this problem you are only required to implement a given algorithm, which is based on the algorithm given to you in Problem 3 of Homework 8. Recall that in that algorithm, we try to start our planning from each of the seven days in a week. The seven different starting days give us seven candidate solutions, and we still do that now. For each starting day, we run the algorithm given in Problem 2 of Homework 8. However, when we want to schedule any full-time agent to start working on a day, we check whether this is the

[2] It does not really matter whether it is one part-time agent working for two days or two part-time agent each working for one day. Therefore, let's say each part-time agent works for one day.

[3] This example is the same as the one in Problem 2 of Homework 8.

last iteration, i.e., whether we will satisfy all demands after this iteration. If no, we schedule full-time agents according to the algorithm given in Problem 2 of Homework 8; otherwise, we find an optimal way to fulfill the unsatisfied demands with both full-time and part-time agents. Note that if an iteration is the last iteration, there are at most five days having unsatisfied demands, and these five days must be consecutive. This makes finding an optimal way easy.

As an example, consider $D^0 = (18, 12, 24, 19, 27, 16, 14)$ as the initial demand distribution in Table 1. Let's start on day 1. We will first set $x_1 = D_1^0 = 18$, as day 1 is currently the first day that still needs officers. We then update D^0 to $D^1 = (0, 0, 6, 1, 9, 16, 14)$ to represent the unfulfilled demands. This requires us to set $x_3 = D_3^1 = 6$. We then have $D^2 = (0, 0, 0, 0, 3, 10, 8)$. By setting $x_5 = 3$, we have $D^3 = (0, 0, 0, 0, 0, 7, 5)$. Up to now, we followed the algorithm given in Problem 2 of Homework 8. However, when we see that setting $x_6 = 7$ (following that algorithm) will terminate the planning, we now we reach the last iteration. Now instead of setting $x_6 = 7$, we should find a best way to fulfill the unsatisfied demands. It turns out that $y = (0, 0, 0, 0, 0, 7, 5)$ is the best, and we do not need any more full-time agents. This results in $x = (18, 0, 6, 0, 3, 0, 0)$ and $y = (0, 0, 0, 0, 0, 7, 5)$ as a candidate solution (if the planning starts on day 1). We should continue to start our planning on the other six days, generate the other six candidate solutions, compare them, and find a best one. Note that part-time agents should be considered only in the last iteration. Also note that "a best way" of fulfilling unsatisfied demands in the last iterations depends on the relationship between F and P. For example, if $F = 10000$ but $P = 8000$, then $y = (0, 0, 0, 0, 0, 7, 5)$ is no longer a part of an optimal schedule.

Input/output formats

There are 15 input files. In each file, there are nine integers D_1, D_2, ..., D_7, F, and P. Two consecutive integers are separated by a white space. It is known that $0 \leq D_i \leq 10000$, $0 \leq F \leq 1000000$, $0 \leq P \leq 1000000$, and $F < 5P$. Given this input, your program should run the given algorithm to find a feasible schedule (x, y). The schedule should be printed out as 14 integers in two lines, x_1, x_2, ..., and x_7 in the first line and y_1, y_2, ..., and y_7 in the second line. Each two consecutive integers in one line should be separated by a white space. If there are multiple best ways in the last iteration, pick the one with the smallest number of full-time agents. Finally, if there are multiple optimal candidate solutions, pick the one whose starting day has the smallest index.

For example, for the input

```
18 12 24 19 27 16 14 10000 3000
```

the output should be

```
18 0 6 0 3 0 0
0 0 0 0 0 7 5
```

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are allowed to use only techniques covered so far. NO other techniques are allowed. Finally, you should write relevant comments for your codes.

Grading criteria

- 30 points will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct set of outputs gives you 2 points.
- 10 points will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.

Problem 3

(40 points) Given a paragraph, we may want to count the number of occurrences of some particular words. For example, if one types "attach", "attached", and/or "attachment" in an e-mail and then click on "send", the e-mail software should check whether there is an attached file.

In this problem, you will be given a paragraph with multiple words separated by white spaces and punctuation marks. You will also be given a list of keywords to be searched within the given paragraph. To make the task easier, we only look for perfect matches, i.e., a keyword is considered existing in the paragraph if there is a word in the paragraph matching the keyword exactly. For example, "key" and "keyword" do not make a perfect match. Of course, punctuation marks are not considered as part of a word. For example, "game-theoretic" is considered two words, "game" and "theoretic". Following this rule, "ntu" and "www.ntu.edu.tw" make a perfect match between the two "ntu"s. Finally, the matching is case-insensitive. Therefore, "NTU" and "ntu" are considered as the same words and thus give a perfect match.

Hint. This homework is easy if you appropriately use `tolower`, `strlen`, `strcmp`, and `strtok`. The example program on p. 52 of the slides about `strtok` is extremely useful.

Input/output formats

There are 15 input files. In each file, there are two lines of characters. These characters include English letters (capital and lowercase), commas, periods, exclamation marks, question marks, hyphens, white spaces, and newline characters. Newline characters only exist at the end of lines. The first line contains the paragraph for you to search for keywords in. Two words are separated by a white space or a punctuation mark. The second line contains several keywords, separated by white spaces. In the first line, there are at most 10000 characters and 1000 words. In the second line, there are at most 100 characters and 10 keywords. Each word or keyword contains at most 50 characters. There is no punctuation mark in the second line.

Given the input, your program should list the numbers of occurrences of these keywords in the order as they appear in the input. For example, for the input

```
This is an apple. Is that a book? I am writing a time-efficient algorithm.
a is time
```

the output should be

```
2 2 1
```

What should be in your source file

Your .cpp source file should contain C++ codes that will both read testing data and complete the above task. For this problem, you are allowed to use only techniques covered so far. NO other techniques are allowed. Finally, you should write relevant comments for your codes.

Grading criteria

- 30 points will be based on the correctness of your output. PDOGS will compile your program, feed testing data into your program, and check the correctness of your outputs. Each fully correct set of outputs gives you 2 points.
- 10 points will be based on how you write your program, including the logic and format. Please try to write a robust, efficient, and easy-to-read program.