CS29003 ALGORITHMS LABORATORY
Assignment No: 8
Last Date of Submission: 30–September–2015

Mr. Avarice owns a lathe machine. He receives n tasks to be carried out by the machine. The i-th task has a load T_i (a positive integer) which stands for the time needed to perform the task. Mr. Avarice will carry out all of the n tasks he is assigned. So he will have to work for a total of $\sum T_i$ time. The work starts at time zero. Let F_i denote the time when the i-th task finishes. If the tasks $i_1, i_2, \ldots, i_k$ are carried out just before starting task i, then $F_i = T_{i_1} + T_{i_2} + \ldots + T_{i_k} + T_i$. Mr. Avarice gets a fixed profit per task, but has to pay a penalty P_i for each unit of time delay in finishing task i. So his goal is to minimize the total penalty, that is, to minimize the objective function defined by the sum $Obj = \sum P_i F_i$. The value of Obj depends upon the sequence in which the tasks are carried out. It is assumed that once a task starts running on the lathe machine, it must be allowed to proceed to completion. Your problem is to suggest optimal greedy algorithms to compute the sequence of the tasks, that minimizes Obj.

Part 1 In this part, assume that the given tasks can be carried out in any order. That is, $n!$ different sequences are possible. Write a function *optnocons*() to compute a best sequence, that is, a sequence that minimizes the objective function Obj. Your algorithm should run in $O(n \log n)$ time. You should print the sequence in which the tasks are selected in your optimal solution.

Part 2 In practice, tasks may be dependent on one another. For example, Mr. Avarice has to shape a cylinder to the desired size before he can cut threads on it, that is, the task of reshaping must be carried out earlier than the task of thread cutting. Assume that the tasks are organized into m chains. Each task must belong to some chain, and no two different chains may share a common task. The tasks in a chain must be carried out in the sequence in which they appear in the chain. Mr. Avarice is not forced to complete a chain in one shot, that is, after completing some tasks in a chain C, he may complete one or more tasks of other chains, and then come back to finish some of the remaining tasks of C, leave C again, and so on. Each *task* (not chain), once started, must be completed before work begins on another task. Use a table to store the chains. Each chain (an entry in the table) contains the following items:

Chain at index i: TaskCount$_i$, a list of TaskCount$_i$ (penalty, load) pairs

The chain is identified by its index i in the table. Use a linked list or a static/dynamic array to store the list of tasks in a chain. Write a function *buildchains*() to put the n input tasks to m chains. The value m, and the chain indices to which the tasks will go are supplied by the user. Append new tasks at the ends of chains.

Part 3 Write an $O(n^2)$-time function *optchain*() for minimizing Obj under the constraints imposed by the chains introduced in Part 2. In order to select the next task(s) for Mr. Avarice, your function selects a chain and a (non-empty) prefix of the chain such that this choice is locally optimal. All the tasks in this prefix are then carried out one by one. If the prefix covers the entire chain, then we are done with the chain. Otherwise, the chain is replaced by a new chain (a suffix of the old chain) starting immediately after the last task included in the prefix. Repeat until there are no chains remaining. You may use additional items in chain-table entries if you need them to solve this part.

Write a main() function to do the following.

- Read n (the number of tasks) and (P_i, T_i) for the n tasks from the user.
- Run the function of Part 1 to compute (and report) the minimum value achieved by Obj.
- Read from the user m (the number of chains). Run the function of Part 2 to send the n tasks to m chains as specified by the user. Print the chains.
- Run the function of Part 3 to obtain the minimum value achieved by Obj under the chain constraints.

Submit a single C/C++ source file solving all the parts. Do not use global/static variables/arrays.

Sample Output

```
        n = 25

        56 33 19  1 38  3 24 14  3  7 32 11 12 79 27  5 17 44 46 20
        59 14 15 71 74 51 96 79 78 51 37 32 82 53 30 18 55 53 29 57
        57 58 65 68 37 91 70 54 33 17

    +++ Optimization without constraints

        ...

        Selecting Task  0 (56, 33)
        Selecting Task 17 (30, 18)

        ...

        Minimized objective function = ...

        m = 5

        Moving tasks to chains:
         0  1  2  3  4  3  3  1  0  1  3  3  0  0  2  0  0  0  3  4
         1  0  3  2  0

        Chain  0:  56  17  74  96  37  82  30  65  33
                   33  44  51  79  32  53  18  68  17
        Chain  1:  19  27  46  57
                    1   5  20  58
        Chain  2:  38  78  70
                    3  51  54
        Chain  3:  24  32  12  59  15  55  37
                   14  11  79  14  71  53  91
        Chain  4:   3  29
                    7  57

    +++ Optimization satisfying chain constraints

        ...

        Selecting Task  0 of Chain  3 (24, 14)
        Selecting Task  1 of Chain  3 (32, 11)
        Selecting Task  0 of Chain  0 (56, 33)
        Selecting Task  1 of Chain  2 (78, 51)
        Selecting Task  2 of Chain  2 (70, 54)
        Selecting Task  1 of Chain  0 (17, 44)
        Selecting Task  2 of Chain  0 (74, 51)

        ...

        Minimized objective function = 419754
```