Functions, Arrays, and Pointers

Object Oriented Programming - Programming Coursework #7 (22 pts)

Due date : June 24, 2020 5pm

Please note that the submission deadline is Wednesday June 24.

1 Task

The objective of this coursework is to gain a deeper insight into defining and using functions, arrays, and pointers. The task is to implement three programs. The first program (Program1) will solve the 0-1 Knapsack problem (10 pts): The program will read values from an input data file ('Program1Input.txt') providing the weight capacity of a bag, a set of items each associated with a weight and a value, and it will generate an output file ('Program1Output.txt') listing the selected items to be stored in the bag (see Sec. 2.1 for details). The second program (Program2) calculates the smallest palindrome integer (reads the same backward as forward) for an input integer (6 pts). It will read an integer from a data file ('Program2Input.txt') and generate an output file containing two integer values ('Program2Output.txt'). The third program (Program3) will calculate the inverse of a matrix (6 pts). Program3 will read a matrix from an input file ('Program3Input.txt') and generate an output file ('Program3Output.txt') containing the inverse of the input matrix.

2 Program instructions and specifications

All submitted codes must satisfy the execution requirements and compilation requirements described in Secs. 3.2 and 3.3, respectively. If the submitted codes do not meet these two sets of requirements, the corresponding codes will get 0 point. There are additional, source code requirements (see Sec. 3.1 for details).

2.1 Program1: Knapsack problem

Suppose that we are given a set H of N items, each having two attributes, a weight and a value, and a bag with a weight capacity. Our goal is to pick items in H to store in the bag such that the total value of the selected items is maximized while the corresponding total weight is smaller than or equal to the weight capacity of the bag. This is an instance of a mathematical optimization: If we denote the weight capacity of the bag, and the weight and value of the i-th item in H as W, w_i and v_i, respectively, the problem is to determine a subset $T \subset \{1, \dots, N\}$ that maximizes

$$\mathcal{V}(T) = \sum_{i \in T} v_i,$$

subject to

$$\mathcal{W}(T) = \sum_{i \in T} w_i \leq W.$$

Note that in this problem, an integer $i \in \{1, \dots, N\}$ can appear only once in T, i.e., one cannot put the same item multiple times in the bag.

Naïvely assessing all 2^N possible subsets T of $\{1, \dots, N\}$ is computationally infeasible even for a moderately large set H. A computationally more affordable approach is given as follows: Suppose that V is a matrix of size $(N+1)\times(W+1)$ such that the entry $V(i,s)$ at row i and column s stores the maximum total value of any subset of items in $\{1,\dots,i\}$ of size at most s. In this case $V(N,W)$ is the maximum value $\mathcal{V}(T^*)$ with T^* being the desired maximizer of $\mathcal{V}$. If we set

$$\begin{aligned} V(0,s) &= 0, && \text{for } 0\leq s\leq W \\ V(i,s) &= -\infty, && \text{for } s<0, \end{aligned} \tag{1}$$

then in the general case $(1\leq i\leq N, 0\leq s\leq W)$, $V(i,s)$ can be calculated by

$$V(i,s)=\max(V(i-1,s), v_i+V(i-1,s-w_i)). \tag{2}$$

The process in Eqs. 1–2 does not give the optimal subset T^* but only gives the maximum possible value $\mathcal{V}(T^*)$. However, extending this process to recored T^* is straightforward.

Input file: 'Program1Input.txt' to be provided by the CW assessor. This file contains integer numbers: The first line is an integer representing the bag capacity W while the second line corresponds to the number of total items N. From the third line, each line contains three integers separated by a space: In the $(i+2)$-th line, the first, second, and third integers respectively represent i, v_i, and w_i.

Output file: 'Program1Output.txt' to be generated by Program1. This should contain integer numbers: The first line corresponds to the optimum value $\mathcal{V}(T^*)$ of the solution T^* while the second line stores the number of elements $|T^*|$ of T^*. The third line lists $|T^*|$ elements of T^* each separated by a space. When there are multiple solutions, e.g., two item subsets, say T^* and T'^* achieve the same maximum total values, providing any one of them is sufficient.

Please note that in this CW the output file names do not contain *StudentID*, e.g., the output file name of Program1 is 'Program1Output.txt' not 'Program1Output*StudentID*.txt'. If the submitted codes do not observe this file name convention, the maximum possible points will be reduced to 50% of the total points.

2.2 Program2: Palindrome number

The reverse and add function starts with an integer, reverses its digits, and adds the reverse to the original. If the sum is not a palindrome, it repeats this procedure until it does. For example, if we start with 195 as the initial number, we get 9339 as the resulting palindrome after the fourth addition:

```
195 + 591 = 786
786 + 687 = 1473
1473 + 3741 = 5214
5214 + 4125 = 9339
```

This method leads to palindromes in a few steps for almost all of the integers. But there are exceptions. 196 is the first number for which no palindrome has been found. It has never been proven, however, that no such palindrome exists. The task is to write a program that takes a given number and gives the resulting palindrome (if one exists) and the number of iterations/additions it took to find it.

Input file: 'Program2Input.txt' to be provided by the CW assessor. 'Program2Input.txt' contains an integer.

Output file: 'Program2Output.txt' to be generated by Program2. This file contains two integers: The first number represents the smallest palindrome for the input provided in 'Program2Input.txt' calculated by the process described above. The second integer represents the number of iterations.

2.3 Program3: Matrix inverse

The Gauss Jordan elimination provides a simple way to calculate the inverse of a matrix (See Prof. Gilbert Strang's excellent turorial on Gauss Jordan elimination at `https://math.mit.edu/~gs/linearalgebra/ila0205.pdf`). However, any method that calculates the matrix inverse can be applied.

Input file: 'Program3Input.txt' to be provided by the CW assessor. The first line of 'Program3Input.txt' contains two integers: The first and the second integers represent the number of rows and columns of the input matrix, namely M and N, respectively. From the second line, each line of 'Program3Input.txt' represents a row of the matrix: The $(i+1)$-th line represents the i-th row consisting of N real-valued numbers each separated by a space.

Output file: 'Program3Output.txt' to be generated by Program3. The format of 'Program3Output.txt' is the same as 'Program3Input.txt': The first line contains the size (# rows and columns) of the inverse matrix while from the second lines, each line corresponds to a row of the inverse matrix. The elements of the output inverse should be visualized to six digits after the decimal point (by rounding off).

Sample inputs and outputs.

'Program1Input.txt' file:

```
15
5
1 4 12
2 2 2
3 2 1
4 1 1
5 10 4
```

'Program1Output.txt' file:

```
15
4
2 3 4 5
```

'Program2Input.txt' file:

```
195
```

'Program2Output.txt' file:

```
9339 4
```

'Program3Input.txt' file:

```
4 4
-0.3658   -0.2369   -0.0205    0.5094
 0.9004    0.5310   -0.1088   -0.4479
-0.9311    0.5904    0.2926    0.3594
-0.1225   -0.6263    0.4187    0.3102
```

'Program3Output.txt' file (note that the numbers are displayed to six digits after the decimal point):

```
4 4
 1.019499 1.368750 -0.224951  0.562791
 0.442442 0.925592  0.756293 -0.266341
-1.154698 0.716513  0.897798  1.890589
 2.854488 1.442187  0.226313  0.356360
```

2.4 Assessment

Program1 will be assessed based on five different instances of 'Program1Input.txt'. This input file will be located at the same directory as the executable file of Program1. When executed with an instance of 'Program1Input.txt', Program1 will generate 'Program1Output.txt' file in the same directory. Similarly, Program2 will be assessed based on five different instances of 'Program2Input.txt' (in the same directory as the executable file of Program2) and it should generate the corresponding 'Program2Output.txt' file. Program3 will follow the same assessment scheme.

3 What to hand in for assessment

Please submit the source codes for Program1, Program2 and Program3, and a brief report (a single report for all programs) via Black Board. Comments in the codes and the report should be written in English. The report should provide code comments explaining the algorithms. If the submission does not include a report, the maximum possible points will be kept at 40% of the full points. Please format the submission as a single zip file 'StudentID_Name.zip' that includes the files of the source codes and the report,

e.g.,
20201111_KwangInKim.zip
- Program1.cpp
- Program2.cpp
- Program3.cpp
- Report.pdf

The submitted source codes should contain 'Program1.cpp', 'Program2.cpp', and 'Program3.cpp' files each including the id and name of the submitting student.

In "Program1.cpp"

```
//StudentId Name
#include <iostream>
Your code here...
```

3.1 Source code requirements

The submitted Program2 code should implement and call a function 'reverseandadd' that takes an integer and returns the integer obtained by reversing the digits in the input, and add the resulting number to the input.

```
int reverseandadd(int Input)
{
        float Output;

        // Your code here...

        return Output;
}
```

All submitted codes should implement the required operations from scratch: **Calling library functions that perform the required operations from any part of the submitted codes is not allowed**. If each submitted code does not meet the source code requirements, e.g.,. not properly defining and using 'reverseandadd' function in Program2, the maximum possible points for the respective program will be kept at 50% of the corresponding maximum points. Please note that this does not mean that such code will automatically get 50% of the maximum points: E.g., if Program2 uses external libraries that calculates palindrome, it will get 0 point.

3.2 Execution requirements

For assessing Program1, 'Program1Input.txt' will be prepared such that it the maximum number of items is less than or equal to 20. For Program2, the input in 'Program2Input.txt' will be prepared in the way that the answer can be calculated in less than 1000 iterations, and yield a palindrome that is not greater than 4294967295. For program3, the size of the input matrix will be smaller than or equal to 20×20. Once executed for these cases, each program should terminate within five seconds at UNI06 server.

3.3 Compilation requirements and instruction

Each submitted code needs to be compilable by g++4.8.5 and C++98 Standard compilers, and in the Linux environment. Specifically, they should be **compilable and executable in UNI06 server**.

Students can verify the correct compilation and execution of their codes by following the instructions provided in ‘howtocompile.pdf’ document accompanying the CW1 announcement.

3.4 Exception handling requirements

For this CW, your programs do not have to handle exceptions caused by incorrect input data types. For instance, for Program2, ‘Program2Input.txt’ will contain a single integer and for Program3, the first line of ‘Program2Input.txt’ will contain two identical integer values, i.e., the input matrix is square.