UNIVERSITY OF PARMA

Department of Engineering and Architecture
Degree course in Computer, Electronic and Communications Engineering
PRACTICAL TEST IN COMPUTER SCIENCE & PROGRAMMING LABORATORY
December 20th , 2024

Name: _____ Surname:_____ Matr:_____ Workstation: _____

Write a program using the C language (name the project with your student **<ID>**) that behaves as described below. The available time is 120 minutes. At the end of the time, the saved files on **U:** are going to be automatically collected. Additional documents, files… are available in **T:\Bertozzi**, it is recommended to use WordPad to read text files.

The "Knapsack Problem" is an optimization problem. Imagine having to fill a backpack of volume **B** with objects chosen from a set of **k** objects, each with volume $\mathbf{a_1, a_2, \ldots, a_k}$ such that $\mathbf{\Sigma_i a_i > B}$. Clearly, you won't be able to fit all k objects into the backpack, but only some combinations of them. The goal is to find which combination(s) of objects fills the backpack as much as possible. Unfortunately, there is no efficient algorithm to solve this problem, which in its general form requires 2^k attempts (NP-complete problem).

The only efficient way to fill it can be obtained if the *k* objects form a "superincreasing" sequence (defined below). In this case, the possible approach is to "put" the largest elements that fit into the backpack (*greedy algorithm*) until the space is exhausted or until all have been tried.

For example: let's consider having 5 objects with volumes 25, 51, 95, 190, and 384, and a backpack of 555 cm^3. Initially, we will try with the object of the largest volume (384), which fits. Then we consider the object with a volume of 190 cm^3, which cannot be inserted since only 555-384=171 cm^3 remain. Therefore, we examine the object with a volume of 95 cm^3, which can be inserted, and so on… Develop a program that:

1. Takes as input the name of a binary file (<number>.dat) which contains: in the first 4 bytes (an int) the volume of the backpack to be filled, in the next 4 bytes the number n of objects, and in the remaining n*4 bytes, the volume of the n objects ordered by volume from the smallest. Reads and stores the contents of this file in an appropriate data structure.
   Print the read data for debugging purposes.

2. Verifies that the read sequence is superincreasing, i.e., that each number in the ordered sequence is greater than the sum of the previous numbers. The program terminates if this condition is not met.

3. Uses the "greedy algorithm" discussed above to calculate which objects maximize the use of the backpack, prints their volumes, and prints the percentage of backpack utilization.

4. Modify point 1 to also handle files (u<number>.dat) that contain the volumes of the n objects not sorted.

It is mandatory to use dynamic memory allocation where it makes sense. Defining and using functions for the first 3 points allows for additional points.

The code should be developed following the proposed order. The correction stops at the first incorrectly implemented step.