UNIVERSITY OF ŽILINA
Faculty of Management Science and Informatics

SP from Algorithms and Data Structures 1 - First level

Assignment

Load the population data of the municipalities in Austria for the last 5 years into a sequential data structure of your choice. Thus, after loading, 1 sequential data structure will be loaded, which will contain the data of the municipalities with their five-year population data for males and females.

The input data can be found in the files **2020.csv**, **2021.csv**, **2022.csv**, **2023.csv** and **2024.csv**. The files are encoded in UTF-8. A description of the data in the file can be found below, a formatted view of the file along with the first 10 records can be found in the file header.pdf. You can edit the input file as you wish, but the required data must not be lost (i.e. no municipality or relevant column can "disappear", but you can convert the files to another format or remove unnecessary data in advance). If you modify the input data, describe this modification in the documentation.

For all municipalities, retrieve and record the following data:

- Name of the municipality (column *Commune (aggregation by NUTS-levels)*).
- Municipality code (the data is in the column next to the name of the municipality, the codes are according to the Statistical Office of Austria, e.g. Deutschkreutz has code 10801 - https://www.statistik.at/atlas/blick/?gemnr=10801).
- Population (record both male and female population) in each year (data in *male* and *female* columns in each file).

The existing library of data structures can be used to develop this level (and only this level in particular) (e.g. `std::vector`, `std::array`).

Program a universal algorithm to process those data from the data structure that satisfy the given predicate[1].

Processing in our case will be considered as inserting data into another sequential direct-access structure. Enable the populated sequential direct structure to be printed.

We will use predicates that return true if:

- `containsStr`: the name of the given municipality contains the specified string.
- `hasMaxResidents`: the total population in the specified year was less than or equal to the specified number.
- `hasMinResidents`: the total population in the specified year was greater than or equal to the specified number.

Thus, the first level can be concretized to: Program an algorithm that filters the municipalities whose {name includes the passed string}/{population in a given year was at most as large as the specified population}/{population in a given year was at least as large as the specified population} into another sequential data structure, which you then print out. Make the algorithm as universal as possible (it will be used later).

[1] A predicate is a function that takes a test object and returns a boolean value depending on whether the object passes the test or not.

Tips

- Encapsulate the algorithm into a separate object.
- Implement predicates (in our case `containsStr`, `hasMaxResidents`, `hasMinResidents`) in the form of lambda functions (preferred) or virtual methods.
- Pass a pair of iterators (in this case, `begin` and `end` from the given data structure) to the algorithm instead of the data structure itself. To avoid having to specify the exact type of iterator, use templates and standard-defined methods.
- You can modify the input file if you don't lose the required information. Then describe such modification in the documentation.

Evaluation

Demonstration of functionality (optional)

- 5 points.
- When demonstrating functionality, you must consider the diacritics (accents)!
- Takes place in the 6th week of the semester.
- Points are not dependent on producing documentation.
- Due to the originality check, points are conditional on uploading the SP to Moodle.

During defense

- It is not necessary to consider diacritics (accents) when defending.
- The data on municipalities are in a single data structure. Loading the data requires a single opening of each file and processing each row of the file at most once - max 3pts.
- The algorithm is in a dedicated separate object (i.e. not in a non-member method or member method of another object that calls it) - max 3pts.
- It is possible to change the predicate in the algorithm without modifying the code of the algorithm itself by
  - virtual method - max 2pts.
  - lambda function - max 3pts.
- The algorithm operates on data that is given in the form
  - of data structure (e.g. `std::vector`) - max 2pts.
  - of a pair of iterators - max 3pts.

Documentation

The documentation is submitted with the final version of the SP at the end of the semester. The documentation must be prepared according to the published requirements (document SP rules).

In addition to the mandatory parts, add the following parts to the documentation:

- **UML activity diagram** of the **universal algorithm** you have constructed – max 2pts.
- **Programmer's guide**, i.e. how to use your universally designed algorithm in the cases described here and how to proceed in case of its future extension with another predicate – max 1pt.
- Modifications to the input files if you have made any.