TNG033/2017: Dugga 1

Attention

This is an **individual** home dugga.

Books, notes, and internet material can be used.

If you find any open questions in some of the presented exercises then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the programs that clarify your assumptions/decisions, if any.

If you use Code::Blocks then follow the instructions given here. Remember that you should not ignore the compiler warnings, since they are usually an indications of serious problems in the code.

You are not allowed to post any information about the dugga in any website until the deadline expires.

Course goals

This is an assessment of whether you can use the concepts introduced in lectures 1 to 8 (e.g. pointers, dynamic memory allocations, classes, operator overloading).

Requirements for grade G

- Readable and well indented code.
- Use of good programming practices.
- Global variables cannot be used[1].
- Respect for submission instructions.
- Code copied from a web page must be clearly indicated through a commented line containing the web page address used as source, with exception for the material posted on the course website.
- Use of statements that are not part of the ISO C++ leads to automatically failing the dugga.
- Your programs must pass all test examples shown in this paper.
- Other criteria are listed in the feedback report document available from course website.

Note that there is no guarantee that your code is correct just because it passes all given test examples. Thus, you should always test your code with extra examples, you come up with.

Deadline

27 of November, 12:00.

[1] All variable must be declared inside a function of your program.

Submission instructions

1. Create **one** source (`.cpp`) file for the exercise. The file must be named with your LiU login (e.g. `ninek007.cpp`).
2. Write your name and personal number at the beginning of the source code file.
3. Submit only the source code file (`.cpp`) through Lisam, without compressing it (i.e. neither `.zip` nor `.rar` files are accepted).

Remember that you should deliver only the source code file. Moreover, answers to the dugga exercises sent by email are ignored.

It is the last submission made in Lisam that is graded. All other submissions you may have uploaded in Lisam are not considered.

Duggor solutions submitted not accordingly the submission instructions are ignored.

Questions

The aim of the dugga is that you solve the exercise without help of other people. However, we give you the possibility to send us questions about the problem in this dugga by email. To this end, email Aida Nordman (aida.vitoria@liu.se). Your emails must have the course code and your study programme in the subject (e.g. "`TNG033/MT2:` ...").

Only emails received from 12:00 to 20:00 on Friday will be answered. Emails received after 20:00 on Friday are not answered. The emails will be answered until Saturday at 12:00.

Be brief and concrete. Emails can be written either in Swedish or English.

Note that you should not send emails related to Lisam system.

User support for Lisam

Help and support for Lisam system is available at helpdesk@student.liu.se and by calling the phone number 013-28 58 98.

Login and password to access the course material

All course material, like lecture slides and code examples, are available through the course website (**`http://weber.itn.liu.se/~aidvi05/courses/12/index.html`**). However, the material is password protected. Use the following login and password to access the material.

login: **`TNG033`** password: **`TNG033ht13`**

Lycka till!!

Exercise

The aim of this exercise is to define a class named `Data_Vector` to represent a sequence of labeled data elements. An example of a `Data_Vector` `dv1` is shown below.

```
<Temperature:38, Height:2.7, Weight:50, Pressure:12.2>
```

Thus, a `Data_Vector` represents a sequence of values (`doubles`), called data elements. Each data element is associated with a label (`string`). For instance, the first data element of `dv1` has value `38` and its label is "`Temperature`".

The number of data elements in a `Data_Vector` can vary. For instance, there can be a `Data_Vector` with one data element, while another `Data_Vector` can have several hundreds of data elements.

In this exercise, assume that labels always start with an upper-case letter followed by lower-case letters, e.g. "`Temperature`".

The class `Data_Vector` should support the following functionality.

- A default constructor that creates an instance of the class with zero data elements.
- A constructor that creates an instance of the class, given an array of values (`doubles`), an array of labels (`strings`), and the number of values. An example is given below.

  ```
  string labels1[] = { "Temperature", "Height", "Weight", "Pressure" };
  double values1[] = { 38, 2.7, 50, 12.2 };

  Data_Vector dv1(values1, labels1, 4);
  ```

  For simplicity, assume that each label is unique (i.e. the array of labels does not contain repeated labels).
- A copy constructor.
- Destructor.
- A function `isEmpty` that tests whether the `Data_Vector` is empty, i.e. it has zero data elements.
- An assignment operator.
- Subscript operator, `operator[]`, that returns the data element associated with a given label. For instance, `dv1["Weight"]` returns the data element `50`.
- A stream insertion operator.
- An `operator+` such that `dv1 + dv2` "merges" two `Data_Vectors` `dv1` and `dv2`. Some examples of use of this operator are given in the appendix.
- The `operator+=` such that `dv1 += dv2;` is equivalent to `dv1 = dv1+dv2;` .

- Product of a `Data_Vector` `dv` with a constant `k` (`double`), `k*dv` or `dv*k`. For instance, `2.0 * dv1` (or `dv1 * 2.0`) returns the `Data_Vector`

  `<Temperature:76, Height:5.4, Weight:100, Pressure:24.4>`

  If `dv` is an empty `Data_Vector` then `k*dv` (or `dv*k`) returns an empty `Data_Vector`.

- A function `slice` such that given a list of labels, represented as a `string` of labels separated by white spaces, returns a `Data_Vector` with the data elements associated with the given labels. For instance, `dv1.slice("Height Pressure")` returns the `Data_Vector <Height:2.7, Pressure:12.2>`.

  Assume that the given list of labels does not contain repeated labels. Additionally, if a label, belonging to the given list of labels, does not occur in the labels of the `Data_Vector` then this label should be simply ignored.
  For example, `dv1.slice("Height Age Pressure")` returns the same result as `dv1.slice("Height Pressure")`, i.e. `<Height:2.7, Pressure:12.2>`, while `dv1.slice("Color")` returns an empty `Data_Vector`.

The class `Data_Vector` should be implemented by using dynamically allocated arrays. Your implementation cannot use containers defined in the C++ Standard Template Library (e.g. `std::vector`, `std::map`, `std::array`, `std::list`, `std::forward_list`, `std::set`, etc).

Design your class with care and follow the given specifications.

Add your code to the file `dugga1.cpp` (to be downloaded with this dugga). This file already contains a `main` function ready to test incrementally your code. Thus, you can comment away those tests corresponding to functions not yet implemented. The file `dugga1_out.txt` contains the expected output of the program.

Note that we cannot guarantee that your solution is correct just because it passes the given tests.

Finally, the source file you submit with your solution should contain the given `main` function without modifications. But, feel free to make any other tests to your code and remember to rename the file `dugga1.cpp` with your LiU id.

Appendix

Three examples that illustrate the use of `operator+` are shown below.

- **Example 1**: consider another `Data_Vector` `dv2`, as defined below.

```
string labels2[] = { "Temperature", "Height", "Weight", "Pressure" };
double values2[] = { 10, 2, 0.5, 0 };
Data_Vector dv2(values2, labels2, 4);
```

  Then, `dv1 + dv2` returns the `Data_Vector`

```
<Temperature:48, Height:4.7, Weight:50.5, Pressure:12.2>
```

- **Example 2**: if `dv0` is an empty `Data_Vector` then `dv1 + dv0` (or `dv0 + dv1`) returns `<Temperature:38, Height:2.7, Weight:50, Pressure:12.2>` (i.e. a copy of `dv1`).

- **Example 3**: consider another `Data_Vector` `dv3`, as defined below.

```
string labels3[] = { "Temperature", "Age", "Pressure", "Duration" };
double values3[] = { 10, 1.5, 0.5, 120 };
Data_Vector dv3(values3, labels3, 4);
```

  Then, `dv1 + dv3` returns the `Data_Vector`

```
<Temperature:48, Height:2.7, Weight:50,
 Pressure:12.7, Age:1.5, Duration:120>
```

  ,while `dv3 + dv1` returns the `Data_Vector`

```
<Temperature:48, Age: 1.5, Pressure:12.7,
 Duration: 120, Height:2.7, Weight:50>
```