Advanced Programming

SZTE Department of Software Engineering
2022

General instructions

- You only need to implement the classes, methods, and data members listed in the description, no extra points are given for anything else.
  - You may include auxiliary functions if they help you solve the problem.
- The quotation marks around strings are not part of the expected output
- There is an initial solution called task.cpp
  - For each task, place your solution after the `//WORK HERE!` comment
  - If you want to test the first task uncomment the `# define TEMPLATE_VECTOR_INTERVAL` line at the beginning of the file. Do this for each task.

Template class: IntervalVector (15 points)

Create an `IntervalVector` template class, which stores the number of `unsigned integer`s of an interval. It has two template arguments: MIN and MAX. These specify the interval in which the class stores the number of elements. Assume that MIN will always be less than or equal to MAX. Example: if we instantiate a variable like this: `IntervalVector<1,3> iv;`, initially it will store zero 1s, 2s, and 3s. Implement the following functions for the `IntervalVector` class:

- Make sure that the number of elements is initially 0!
- Implement the += operator. Its right-hand side argument is an `unsigned integer`. We want to store an extra one of this number in the `IntervalVector` object, if it falls within the closed interval [MIN,MAX].
  - Example: In the case of `IntervalVector<1,5> v; v+=5; v+=5;` two fives are stored in the vector.
  - If the input is incorrect (it is outside the interval), throw the value in the parameter (i.e. the wrong index) as an exception.
- Create an indexer operator for the class, which takes an `unsigned` value as a parameter and returns how many of this element are stored in the class. There should be two versions: a constant version and a version through which the stored value can be modified. If the parameter does not fall within the interval, throw the value in the parameter as an exception (i.e. the wrong index).
- Implement the output stream operator («). The format of the output is like this:

```
    <MIN>: <number of items>
    <MIN+1>: <number of items>
    ...
    <MAX>: <number of items>
```

  – Example:

```
        IntervalVector<1,5> v;
        v += 5;
        cout << v;
        //Output
        //1: 0
        //2: 0
        //3: 0
        //4: 0
        //5: 1
```

- Implement the + operator for two `IntervalVector` objects, which cover exactly the same interval. The result is an `IntervalVector` summing the number of elements of the two operands.

  – Example:

```
        IntervalVector<1,5> v1, v2;
        v1 += 5;
        v2 += 5;
        IntervalVector<1,5> v3 = v1 + v3;
        cout << v;
        //Output
        //1: 0
        //2: 0
        //3: 0
        //4: 0
        //5: 2
```

Template function: count_element (10 points)

Write an `unsigned count_element` template function. It should count how many times a given element occurs in an array. It should have three template arguments. The first template parameter specifies the type of elements stored by the array (T), the second argument specifies the length of the array (N), and the third argument specifies the type of element to search for (K). The default value for K should be T, i.e. the type of elements stored by the array. The function has two parameters: the first is a pointer pointing to T, i.e. this pointer points to an "N-sized array of elements of type T". The second parameter should be of type K. This parameter determines which element is counted. By default, the function counts the number of times the value in the second parameter occurs in the array and returns it. Let there be a specialization of the function where the template parameters are T=std::string, N=5, K=char. In this case, the total number of occurrences of the character received in the second function parameter should be counted in all the strings stored in the array.

```
int arr[5] = {1,3,5,2,3};
cout << count_element<int, 5>(arr, 3) << endl;

string arr2[5] = {"asd", "bsd", "asda", "cccc", "aaa"};
cout << count_element<string, 5, char>(arr2, 'a') << endl;
//Output:
//2
//6
```

Manipulator: anonym (7 points)

Write a manipulator named `anonym`. It anonymizes words starting with a capital letter in the input. Anonymization means overwriting the characters of words starting with a capital letter with * characters. A word is a string of characters with a space before and after it. The only exception is if the word is at the beginning of the input (no other character before it) or at the end of the input (no other character after it). No other case needs to be addressed. The manipulator should behave as follows: the first time it is used, anonymization is turned on, the next time it is used, anonymization is turned off, the third time it is used, anonymization is turned on again, and so on.

```
cout << anonym << "Simon Doyle is the murderer. " << anonym << "It was ↩
   investigated by Hercule Poirot." << endl;
cout << anonym;
cout << "his last known location: the Nile" << anonym << endl;
//Output:
//***** ***** is the murderer. It was investigated by Hercule Poirot.
//his last known location: the ****
```

Effector: „Exchange” (6 point)

Write an `Exchange` effector that takes an `unsigned` value and a `string` as its parameters. The effector converts euro to forint and vice versa. The number parameter represents the amount of money, the string represents the currency, it can be "EUR" or "HUF". The exchange rate of the euro should be 425. That is, for an input of 100 EUR, write `"42500 HUF"` to the output stream. The conversion from forint to euro should be done with integer division. That is, if the input is 800 HUF, the output should be `"1 EUR"`. If the string of the currency is not "HUF" or "EUR", the output written on the stream should be `"ERROR"`. The quotation marks are not part of the outputs.

```
cout << "Convert 100 EUR to HUF: " << Exchange(100, "EUR") << endl;
cout << "Convert 800 HUF to EUR: " << Exchange(800, "HUF") << endl;
//Output
//Convert 100 EUR to HUF: 42500 HUF
//Convert 800 HUF to EUR: 1 EUR
```

Traits (12 points)

For this task, see the initial solutions in task.cpp (after the first `#ifdef TRAITS_CAKE` line). There are two types of sugar (normal and brown sugar), two types of food size (full portion and half portion), and two types of person: dietary (Dietary) and non-dietary (Normal) are given. The tasks are as follows:

- Create two specializations for the `PersonTraits` template class. In both, the types `sugar_type` and `portion_size` should be publicly defined as follows:
  - For a `Normal` person, the `sugar_type` is `Sugar` and the `portion_size` is `FullPortion`.
  - For a `Dietary` person, the `sugar_type` is `BrownSugar` and the `portion_size` is `HalfPortion`.
- Create a template class called `PersonalizedCake`. Its first template argument is a type (let's call it `Person`) that has a public method called `get_name()`. This method returns a `string`. The second template argument of the `PersonalizedCake` is a type that publicly defines the types `sugar_type` and `portion_size`. The second template argument should have a default value which is "recommended cake composition" based on the first parameter.
  - Let the `PersonalizedCake` class have a public constructor with one parameter. The type of the parameter is `Person` (obtained in the first template argument). Store this parameter as a data member.
  - There should be two additional data members in the class that store the sugar type and the portion size.
  - Be able to write the class to `ostream`. The string returned by the `get_name` method of the stored `Person` data member should be written to the stream (no line breaks at the end)
  - Implement a method called `double calorie_intake()` which uses the sugar type's `energy` method and the portion size's `size` method to calculate the calorie intake (the multiplication of the two values).
  - Implement the `>` operator, which can compare two `PersonalizedCake` objects with arbitrary template parameters (not just identical ones). Whichever `PersonalizedCake` has the larger `calorie_intake()` is the larger of the two.