Indexing sequences

Submission deadline:	**2022-05-08 23:59:59**
Late submission with malus:	**2022-05-15 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**5.0000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	3 / 20 Free retries + 20 Penalized retries (-2 % penalty each retry)
Advices:	2 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to implement class template, the template will implement a generic index.

The index is implemented as generic class `CIndex`. The instance is initialized with a parameter - a sequence of elements. The sequence is to be indexed. The sequence may be of the form:

- `string` - a C or C++ string consisting of characters,
- `vector<T>` - a vector of values (elements of type T),
- `list<T>` - a list if values (elements of type T).

Once the instance is initialized and the input sequence is indexed, the instance may be used for searching. The search is given some sequence of elements (the data type matches the stored sequence). The result is a set of indices where the searched sequence was found in the indexed sequence.

To develop a flexible class, the matching of elements may be parameterized. The class will accept two generic parameters - the type of the indexed sequence and a comparator (second optional generic parameter). The searching does not have to compare the elements for an exact match. If a custom comparator is provided, the matching shall follow that comparator. For instance, the customized comparator may be used to implement case insensitive searching. The comparator will follow the guidelines used in STL: it is of the form of a function, functor, or a lambda function. The comparator accepts two parameters of type T (elements to compare), the result is `true` if the first element is smaller than the second element in the desired ordering. If no custom comparator is provided, operator < will be used.

The elements in the sequences are virtually unlimited. Examples are characters (char), integers (int), or strings (C++ string). In general, the element type T guarantees:

- copying (operator = and copy constructor),
- comparison by operator < ("less than"), or by the custom comparator,
- freeing (destructor),
- there may be further operations available in type T, however, your implementation cannot rely on them. Caution: default constructor, operator ==, operator !=, ... are not guaranteed.

Submit a source file with your `CIndex` template implementation. Moreover, add any further class (or classes) your implementation requires. The class must follow the public interface below. If there is a mismatch in the interface, the compilation will fail. You may extend the interface and add you auxiliary methods and member variables (both public and private, although private are preferred). The submitted file must include both declarations as well as implementation of the class (the methods may be implemented inline but do not have to). The submitted file shall not contain anything unnecessary. In particular, if the file contains `main`, your tests, or any #include definitions, please, keep them in a conditional compile block. Use the template below as a basis for your implementation. If the preprocessor definitions are preserved, the file maybe submitted to Progtest.

Your implementation must use STL, the compiler supports almost all C++ 17 extensions.

There are mandatory and bonus tests in the homework. An implementation using naive searching algorithm passes mandatory tests, however, it does not pass speed tests. To pass the bonus tests, an improved algorithm must be used. You may assume:

- Once indexed, the sequence is searched many times. Therefore, it make sense to spend some time preprocessing the sequence, the searching may be faster. The searching is called approx 100 times per instance.
- The fully generic implementation (generic type, generic comparator) is required in the mandatory tests. The speed tests use `string` type and default comparator. If a specialized template with these generic parameters is provided, the speed may be several orders of magnitude higher.
- KMP algorithm may be used to speed up string searching, suffix array or DAWG may be used to improve the indexing.

Examples and basic tests are included in the attached file.

The solution of this homework cannot be used for code review.

Sample data: Download

Reference

- **Evaluator: computer**
  - Program compiled
  - Test 'Zakladni test s parametry podle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test nahodnymi daty': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.244 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test nahodnymi daty + mem dbg': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.712 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Rychlost #1': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 13.355 s (limit: 20.000 s)
    - Bonus test - success, evaluation: 120.00 %
  - Test 'Rychlost #2': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 1.064 s (limit: 20.000 s)
    - Bonus test - success, evaluation: 120.00 %
  - Overall ratio: 144.00 % (= 1.00 * 1.00 * 1.00 * 1.20 * 1.20)
- Total percent: 144.00 %
- Early submission bonus: 0.50
- Total points: 1.44 * (5.00 + 0.50) = 7.92

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**10**	--	--	--
Lines of code:	**103**	**10.30 ± 5.93**	**23**	`CIndex::search`
Cyclomatic complexity:	**27**	**2.70 ± 2.10**	**8**	`CIndex::search`

3 **2022-05-08 21:08:20** Download

Submission status:	Evaluated
Evaluation:	5.0000

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.272 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test + mem dbg': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 2.359 s (limit: 10.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Speed #1': Abnormal program termination (Time limit exceeded)

        - Cumulative test time exceeded, killed after:: 20.005 s (limit: 20.000 s)
        - Bonus test - failed, evaluation: No bonus awarded
    - Test 'Speed #2': Not tested
        - Bonus test - failed, evaluation: No bonus awarded
    - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00)
- Advices used: 1
- Penalty due to advices: None (1 <= 2 limit)
- Total percent: 100.00 %
- Total points: 1.00 * 5.00 = 5.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**6**	--	--	--
Lines of code:	**115**	**19.17 ± 26.09**	**70**	`main`
Cyclomatic complexity:	**19**	**3.17 ± 4.41**	**13**	`CIndex::search`

2	**2022-05-08 20:33:40**
Submission status:	Evaluated
Evaluation:	0.0000

- **Evaluator: computer**
    - ☑ Compile in 'basic' mode failed.

```
In file included from testbed.cpp:252:
tested: In instantiation of ‘std::set<long unsigned int> student_865cd57ff2f5b4ed5e41f1bd60096c73::CIndex<T_, C_>::search(const T_&) const [with T_ =
testbed.cpp:496:33:   required from ‘void compareResults(int&, int&, teacher_865cd57ff2f5b4ed5e41f1bd60096c73::CIndex<T_, C_>&, student_865cd57ff2f5b
testbed.cpp:638:45:   required from here
tested:43:27: error: no match for ‘operator==’ (operand types are ‘const CMinType’ and ‘const CMinType’)
   43 |                 if (*ptr1 == *ptr3) {
      |                     ~~~~~~^~~~~~~
In file included from /usr/include/c++/10/bits/stl_algobase.h:64,
                 from /usr/include/c++/10/bits/specfun.h:45,
                 from /usr/include/c++/10/cmath:1927,
                 from testbed.cpp:6:
/usr/include/c++/10/bits/stl_pai

<skipping 161874 B>

 std::variant<_Types ...>&)’
 1195 |   _VARIANT_RELATION_FUNCTION_TEMPLATE(!=, not_equal)
      |   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/c++/10/variant:1195:3: note:   template argument deduction/substitution failed:
In file included from testbed.cpp:252:
tested:47:39: note:   ‘const CMinType’ is not derived from ‘const std::variant<_Types ...>’
   47 |                             if (*ptr2 != *ptr3) {
      |                                 ~~~~~~^~~~~~~
In file included from testbed.cpp:21:
/usr/include/c++/10/variant:1233:18: note: candidate: ‘constexpr bool std::operator!=(std::monostate, std::monostate)’
 1233 |   constexpr bool operator!=(monostate, monostate) noexcept { return false; }
      |                  ^~~~~~~~
/usr/include/c++/10/variant:1233:29: note:   no known conversion for argument 1 from ‘const CMinType’ to ‘std::monostate’
 1233 |   constexpr bool operator!=(monostate, monostate) noexcept { return false; }
      |                             ^~~~~~~~~
```

- Advices used: 1
- Penalty due to advices: None (1 <= 2 limit)
- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00

1	**2022-05-08 19:46:50**
Submission status:	Evaluated
Evaluation:	0.0000

- **Evaluator: computer**
    - ☑ Compile in 'basic' mode failed.

```
In file included from testbed.cpp:252:
tested: In instantiation of ‘std::set<long unsigned int> student_f0571864b1cc543e22ca1e14652d90a4::CIndex<T_, C_>::search(const T_&) const [with T_ =
testbed.cpp:496:33:   required from ‘void compareResults(int&, int&, teacher_f0571864b1cc543e22ca1e14652d90a4::CIndex<T_, C_>&, student_f0571864b1cc5
testbed.cpp:638:45:   required from here
tested:48:27: error: no match for ‘operator==’ (operand types are ‘const CMinType’ and ‘const CMinType’)
   48 |                 if (*ptr1 == *ptr3) {
      |                     ~~~~~~^~~~~~~
In file included from /usr/include/c++/10/bits/stl_algobase.h:64,
                 from /usr/include/c++/10/bits/specfun.h:45,
                 from /usr/include/c++/10/cmath:1927,
                 from testbed.cpp:6:
/usr/include/c++/10/bits/stl_pai

<skipping 161874 B>

 std::variant<_Types ...>&)’
 1195 |   _VARIANT_RELATION_FUNCTION_TEMPLATE(!=, not_equal)
      |   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
/usr/include/c++/10/variant:1195:3: note:   template argument deduction/substitution failed:
In file included from testbed.cpp:252:
tested:52:39: note:   ‘const CMinType’ is not derived from ‘const std::variant<_Types ...>’
   52 |                             if (*ptr2 != *ptr3) {
      |                                 ~~~~~~^~~~~~~
In file included from testbed.cpp:21:
/usr/include/c++/10/variant:1233:18: note: candidate: ‘constexpr bool std::operator!=(std::monostate, std::monostate)’
 1233 |   constexpr bool operator!=(monostate, monostate) noexcept { return false; }
      |                  ^~~~~~~~
/usr/include/c++/10/variant:1233:29: note:   no known conversion for argument 1 from ‘const CMinType’ to ‘std::monostate’
 1233 |   constexpr bool operator!=(monostate, monostate) noexcept { return false; }
      |                             ^~~~~~~~~
```

- Total percent: 0.00 %
- Total points: 0.00 * 5.00 = 0.00