DNS #1

Submission deadline:	**2021-04-25 23:59:59**
Late submission with malus:	**2021-06-30 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**3.0000**
Max. assessment:	**3.0000** (Without bonus points)
Submissions:	4 / 20 Free retries + 20 Penalized retries (-2 % penalty each retry)
Advices:	1 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop classes to support DNS server backend.

DNS is a network service which supports name-to-address translation. For instance DNS name `progtest.fit.cvut.cz` is (as of now) translated to IP address `147.32.232.142`. This translation is due to DNS record named `progtest` of type `A` (IPv4 address) in zone `fit.cvut.cz`. DNS is a heterogeneous system, there is support for other translations (AAAA for IPv6 address, MX for mail server location, CNAME for name aliases, ...). Next, DNS allows one name with many different record types (e.g. there may exist both A and AAAA record for a name, such as in the case of www.fit.cvut.cz), moreover, there may exist several records of the same type for a name. For example, there may exist many A records for a name. Clients choose arbitrary record from the list thus the workload is distributed among the servers. For example google.cz uses this technique. On the other hand, 100% identical DNS records cannot be duplicated.

Real DNS is a complex distributed and redundant system. Our implementation is a simplification where the implemented classes represent the DNS records and a container object. Next, we will limit our implementation to A, AAAA, and MX records.

Thus the task is to implement classes `CRecA`, `CRecAAAA`, and `CRecMX`. These classes will represent the corresponding DNS records. Next, there is to be implemented class `CZone`. The class will serve as a container of the above DNS records. Apart from these classes, you may (and you will have to) develop your further auxiliary classes. Finally, there are some already developed classes to help you: `CIPv4` and `CIPv6` to represent IP addresses. The later two classes are present in the progtest environment and their lightweight version is delivered in the attached archive (you may use them to develop and test your implementation).

CRecA

Class `CRecA` represents a record of type A - a translation of a name into an IPv4 address. The interface is:

constructor
: initializes a new instance based on the name and IPv4 address parameters,

`Name()`
: method returns the name component,

`Type()`
: method return the type (A in this case).

output operator
: displays the record in the output stream (see the attached archive for details), the output does not include a newline,

other
: you may add further methods to the class interface as needed.

CRecAAAA

Class `CRecAAAA` represents a record of type AAAA - a translation of a name into an IPv6 address. The interface is:

constructor
: initializes a new instance based on the name and IPv6 address parameters,

`Name()`
: method returns the name component,

`Type()`
: method return the type (AAAA in this case).

output operator
: displays the record in the output stream (see the attached archive for details), the output does not include a newline,

other
: you may add further methods to the class interface as needed.

CRecMX

Class `CRecMX` represents a record of type MX - an information on mail server location. The interface is:

constructor
: initializes a new instance based on the name, mail server name and its priority,

`Name()`
: method returns the name component,

`Type()`
: method return the type (AAAA in this case).

output operator
: displays the record in the output stream (see the attached archive for details), the output does not include a newline,

other
: you may add further methods to the class interface as needed.

CZone

Class `CZone` is a container of DNS records. The interface is:

constructor
: initializes a new empty instance, with zone name set,

`Add()`
: method adds a record (A/AAAA/MX) into the zone. The record is appended to the end of the list of existing records. The method returns `true` for success or `false` for failure (an identical record already exists in the zone).

`Del()`
: method deletes a record from the list. The ordering of other records in the zone is not modified. The method returns `true` for success or `false` for failure (identical record does not exist in the zone).

`Search()`
: method searches for record(s) with given name. The returned records must support the following operations:

- output operator - the returned records are displayed in the output stream (see samples for details). The ordering of the records must follow the order the records were added to the zone,
- method `Count` to return the number of records selected,
- index operator to access individual records, i.e. the results of the index operator must accept methods `Type`, `Name`, and output operator from the interface of `CRecA`/`CRecAAAA`/`CRecMX`. Moreover, if the index in the index operator is outside the bounds, `std::out_of_range` must be thrown.

output operator
: the zone is displayed into the stream, the exact format is shown in sample runs. The order of records must follow the order the records were added to the zone,

copying
: zone objects must correctly handle copying/operator=.

other
: you may add further methods to the class interface as needed.

Analyze the problem well before coding. Focus on the object design, do not duplicate code. Correct solution requires polymorphic classes. The testing environment tests whether the classes are polymorphic or not. Moreover, it tests basic design properties of the classes. A solution without polymorphic classes will not compile. Do not use RTTI (`typeid` will not work, header `typeinfo` is missing, it is an intention). Try not to use `dynamic_cast` (or at least, do not use it often). If the object design is clumsy, there will be many branches in the code, the code will be lengthy, confusing, and error-prone.

Submit a source file with the implementation of the required classes (and your auxiliary classes). Download the attached archive and use it as a basis of your development. The archive consists of `test.cpp` (place your implementation here) and `ipaddress.h` (the implementation of auxiliary classes already present in the testing environment). The attached file `test.cpp` is intended to be submitted to Progtest. You just need to add your classes (outside of the conditional compilation blocks), and keep existing conditional compilation in place.

There is no need to use any special algorithm to pass the mandatory tests. Your implementation will need fast searching to pass the bonus test (many searches, many records in the zone).

Advice:

- You can use `dynamic_cast` in your code, indeed, it may be useful to implement comparisons (reference uses `dynamic_cast` exactly there). On the other hand, do not use `dynamic_cast` to implement `CZone`. The idea behind is simple - you do not want to update the existing code when a new record type is introduced. If `dynamic_cast` is used in `CZone::Add` or `CZone::Del`, then each newly introduced record type requires updates in these methods. If `dynamic_cast` is used in the classes implementing the various records and if these classes are well-designed, then the `dynamic_cast` will only cast to the type of its own class (e.g., the implementation of `CRecA` will use `dynamic_cast<CRecA&>`). Thus the existing implementation remains unchanged when a new class (for a new record type) is added.
- A solution of this problem **cannot** be used for code review. However, you may complete your solution to pass the extended problem (DNS #2). A solution of the extended problem may then be used for code review.

Sample data:		**Download**
Submit:	**Choose File** **No file selected**	**Submit**

Reference

- **Evaluator: computer**
  - Program compiled
  - **0** Test 'Zakladni test podle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.001 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test navrhu trid': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 4.999 s)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test nahodnymi hodnotami': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.220 s (limit: 4.999 s)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test kopirujiciho konstruktoru': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.022 s (limit: 4.779 s)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test operatoru=': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.026 s (limit: 4.757 s)
    - Mandatory test success, evaluation: 100.00 %
  - **0** Test 'Test nahodnymi daty + kontrola prace s pameti': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.017 s (limit: 5.000 s)

        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Test rychlosti': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 1.020 s (limit: 5.000 s)
        - Bonus test - success, evaluation: 120.00 %
    - Overall ratio: 120.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.20)
- Total percent: 120.00 %
- Early submission bonus: 0.30
- Total points: 1.20 * (3.00 + 0.30) = 3.96

		Total	Average	Maximum	Function name
SW metrics:	Functions:	**22**	--	--	--
	Lines of code:	**172**	**7.82 ± 3.70**	**18**	`CZone::operator=`
	Cyclomatic complexity:	**41**	**1.86 ± 1.10**	**4**	`CZone::operator=`

4	**2021-04-25 13:00:08**	Download
Submission status:	Evaluated	
Evaluation:	3.0000	

- **Evaluator: computer**
    - Program compiled
    - **0** Test 'Basic test with sample commands': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 0.001 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Class design test': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 0.000 s (limit: 4.999 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Random test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.519 s (limit: 4.999 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Copy constructor test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.170 s (limit: 4.480 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'operator= test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.139 s (limit: 4.310 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Random test + mem test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.023 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - **0** Test 'Speed test': Abnormal program termination (Time limit exceeded)
        - Cumulative test time exceeded, killed after:: 5.007 s (limit: 5.000 s)
        - Bonus test - failed, evaluation: No bonus awarded
    - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 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 * 3.00 = 3.00

		Total	Average	Maximum	Function name
SW metrics:	Functions:	**34**	--	--	--
	Lines of code:	**375**	**11.03 ± 31.28**	**190**	`CRecA::CRecAAAA::CRecMX::main`
	Cyclomatic complexity:	**55**	**1.62 ± 1.03**	**5**	`CRecA::CRecAAAA::CRecMX::CRecMX::operator==`

3	**2021-04-25 12:57:57**	Download
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
    - Compile in 'basic' mode failed. **[Unlock advice (655 B)]**
- Advices used: 1
- Penalty due to advices: None (1 <= 2 limit)
- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00

2	**2021-04-25 12:50:36**	Download
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
    - ☐ Compile in 'basic' mode failed.
- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00

1	**2021-04-25 12:31:21**	Download
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
    - Compile in 'basic' mode failed. **[Unlock advice (2069 B)]**
- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00