Data types I

Submission deadline:	**2022-05-01 23:59:59**
Late submission with malus:	**2022-05-15 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:	4 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to develop classes that implement data types in a C/C++ compiler.

A compiler has to keep track of the types used in the program being compiled. Some data types are built-in (e.g., int and double), while other are programmer-defined (e.g., enums and structs). The classes needed in this homework are designed to describe the four aforementioned data types and support basic operations with the types.

The program consists of the following classes:

CDataTypeInt

The class implements built-in type `int`. The interface is:

default constructor
: initializes the instance,

getSize()
: the method returns size of the type (4 in the case of int),

operator ==
: compares two types, returns true if they are identical (both are ints),

operator !=
: compares two types, returns true, if they are not identical,

operator <<
: displays type declaration in the given stream.

CDataTypeDouble

The class implements built-in type `double`. The interface is:

default constructor
: initializes the instance,

getSize()
: the method returns size of the type (8 in the case of double),

operator ==
: compares two types, returns true if they are identical (both are doubles),

operator !=
: compares two types, returns true, if they are not identical,

operator <<
: displays type declaration in the given stream.

CDataTypeEnum

The class represents an enumeration. The interface is:

default constructor
: initializes the instance,

getSize()
: the method returns size of the type (4 in the case of enums),

add()
: the method adds a new enumeration value to the list. An exception is thrown if the same value is already included in the list (see sample runs),

operator ==
: compares two types, returns true if they are identical (both are enumerations, moreover, the lists of values are identical, the values are listed in the same order),

operator !=
: compares two types, returns true, if they are not identical,

operator <<
: displays type declaration in the given stream. Caution, enum values must be printed in the order they were added.

CDataTypeStruct

The class represents a structure. The interface is:

default constructor

initializes the instance,

getSize()
: the method returns size of the type (based on the size of the fields),

addField(name, type)
: the method adds a new field to the end of the field list. A field is described by its name and type (int/double/enum). If the name collides with another existing name in the field list, the method shall throw an exception (see sample runs),

field(name)
: the method is used to access the type of the given field. If the name does not refer to an existing field, the method throws an exception (see sample runs). The field is accessed in read-only mode,

operator ==
: compares two types, returns true if they are identical (both are structures, both have the same number of fields, the datatypes of corresponding fields are the same, HOWEVER, field names do not have to match),

operator !=
: compares two types, returns true, if they are not identical,

operator <<
: displays type declaration in the given stream. The fields are listed in the order they were added via `addField`.

Submit a source file with the implementation of `CDataTypeInt`, `CDataTypeDouble`, `CDataTypeEnum`, and `CDataTypeStruct`. The classes must implement the above interface, moreover, you may add your own method to simplify the code.

Notes:

- Both STL and C++ string is available. However, the problem needs a good analysis before the coding. Use polymorphism when designing the classes. Indeed, the classes will not be accepted if they are not polymorphic.
- Type declarations produced by operator << are not tested for an exact match. Instead, whitespace is ignored in the comparison. The attached example uses function `whitespaceMatch` for the comparison, you are expected to implement the function.
- Method `getSize` should return the size using `size_t` data type.
- Code sample is available in the attached archive.
- A solution of this (simplified) problem **cannot** be used for code review. However, a correct solution of the extended problem (Data types II) is eligible for code review.
- **Update 2022-04-09:** several methods accept strings as parameters. Implement your interface such that it accepts both C-strings (`const char *`) and C++ strings (`std::string`).

Sample data: **Download**

Reference

- **Evaluator: computer**
  - Program compiled
  - Test 'Zakladni test podle ukazky': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test navrhu trid': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test nahodnymi hodnotami': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.278 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Test nahodnymi daty + mem debugger': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.413 s (limit: 5.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00)
- Total percent: 100.00 %
- Early submission bonus: 0.30
- Total points: 1.00 * (3.00 + 0.30) = 3.30

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**24**	--	--	--
Lines of code:	**138**	**5.75 ± 2.57**	**14**	`toText`
Cyclomatic complexity:	**39**	**1.63 ± 0.90**	**4**	`CDataTypeStruct::operator==`

4 **2022-04-26 21:58:59** **Download**

Submission status: Evaluated
Evaluation: 3.0000

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample commands': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 5.000 s)

        - Mandatory test success, evaluation: 100.00 %
    - Test 'Class design test': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 0.000 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Test 'Random test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.236 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Test 'Random test + mem debugger': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.371 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00)
- Total percent: 100.00 %
- Total points: 1.00 * 3.00 = 3.00

		Total	Average	Maximum	Function name
SW metrics:	Functions:	**27**	--	--	--
	Lines of code:	**345**	**12.78 ± 34.66**	**188**	`main`
	Cyclomatic complexity:	**59**	**2.19 ± 1.47**	**5**	`operator`

3	**2022-04-26 21:11:06**	**Download**
Submission status:	Evaluated	
Evaluation:	2.1000	

- **Evaluator: computer**
    - Program compiled
    - Test 'Basic test with sample commands': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 0.000 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Test 'Class design test': success
        - result: 100.00 %, required: 100.00 %
        - Total run time: 0.000 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Test 'Random test': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.207 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Test 'Random test + mem debugger': success
        - result: 100.00 %, required: 50.00 %
        - Total run time: 0.444 s (limit: 5.000 s)
        - Mandatory test success, evaluation: 100.00 %
    - Detected total of 12751 leaked memory blocks (30 % penalty).
    - Overall ratio: 70.00 % (= (1.00 * 1.00 * 1.00 * 1.00) * 0.7)
- Total percent: 70.00 %
- Total points: 0.70 * 3.00 = 2.10

		Total	Average	Maximum	Function name
SW metrics:	Functions:	**29**	--	--	--
	Lines of code:	**346**	**11.93 ± 33.58**	**188**	`main`
	Cyclomatic complexity:	**60**	**2.07 ± 1.46**	**5**	`operator`

2	**2022-04-26 20:26:09**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
    - Program compiled
    - Test 'Basic test with sample commands': failed
        - result: 60.87 %, required: 100.00 %
        - Total run time: 0.000 s (limit: 5.000 s)
        - Mandatory test failed, evaluation: 0.00 %
        - ☑ Failed (invalid output)

          ```
          op << mismatch
          ```

          ☑ Output data **[56 B / 56 B]**

```
struct
{
int;
enum
{
NEW
FIXED
BROKEN
DEAD,
};
double;
}
```

☑ Reference **[110 B / 110 B]**

```
struct
{
  int m_Length;
  enum
  {
    NEW,
    FIXED,
    BROKEN,
    DEAD
  } m_Status;
  double m_Ratio;
}
```

- ☑ Failed (invalid output)

```
 op << mismatch
```

☑ Output data **[57 B / 57 B]**

```
struct
{
int;
enum
{
NEW
FIXED
BROKEN
READY,
};
double;
}
```

☑ Reference **[111 B / 111 B]**

```
struct
{
  int m_Length;
  enum
  {
    NEW,
    FIXED,
    BROKEN,
    READY
  } m_Status;
  double m_Ratio;
}
```

- ☑ Failed (invalid output)

```
 op << mismatch
```

☑ Output data **[56 B / 56 B]**

```
struct
{
int;
enum
{
NEW
FIXED
BROKEN
DEAD,
};
```

```
double;
}
```

☑ Reference **[109 B / 109 B]**

```
struct
{
  int m_First;
  enum
  {
    NEW,
    FIXED,
    BROKEN,
    DEAD
  } m_Second;
  double m_Third;
}
```

- ☑ Failed (invalid output)

```
 op << mismatch
```

☑ Output data **[53 B / 53 B]**

```
struct
{
int;
enum
{
NEW
FIXED
BROKEN
DEAD,
};
int;
}
```

☑ Reference **[107 B / 107 B]**

```
struct
{
  int m_Length;
  enum
  {
    NEW,
    FIXED,
    BROKEN,
    DEAD
  } m_Status;
  int m_Ratio;
}
```

- Failed (invalid output)
- Failed (invalid output)
- Failed (invalid output)
- Failed (invalid output)

- Detected total of 12573 leaked memory blocks (30 % penalty).
- Overall ratio: 0.00 %

- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**29**	--	--	--
Lines of code:	**343**	**11.83 ± 33.61**	**188**	`main`
Cyclomatic complexity:	**60**	**2.07 ± 1.46**	**5**	`operator`

1	**2022-04-26 20:21:37**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample commands': failed
    - result: 60.87 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 5.000 s)
    - Mandatory test failed, evaluation: 0.00 %

        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
        - Failed (invalid output)
    - Detected total of 12774 leaked memory blocks (30 % penalty).
    - Overall ratio: 0.00 %
- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**26**	--	--	--
Lines of code:	**334**	**12.85 ± 35.36**	**188**	`main`
Cyclomatic complexity:	**57**	**2.19 ± 1.49**	**5**	`operator`