Homework Assignment

Class:	CS202	**Semester:**	Fall 2018
Assignment type:	Homework assignment	**Due date:**	11/9/18
Assignment topic:	Operator overloading	**Assignment no.**	4
Delivery:	WebCampus – cpp files and txt file		

Goal
Practice the operator overloading for the user defined class

Input to the program
Input is internal: code in **`main()`** function is used to test the functionality

Procedure for the implementation
Develop the **`smartArray`** class. There are several operations that can be done on this array. Implement:

- Adding two **`smartArray`** objects: (elements on the same positions are added)
- Subtracting two **`smartArray`** objects (elements on the same positions are subtracted)
- Multiplying two **`smartArray`** objects (elements on the same positions are multiplied)
- Dividing two **`smartArray`** objects (elements on the same positions are divided)
- Merging two **`smartArray`** objects (elements from array **`A`** and **`B`** are put to the array **`C`**)

General remarks
- Keep all your testing code in submitted **`cpp`** files
- For all the problems, ensure/add the proper memory allocation/deallocation (all instructions about memory are not necessarily mentioned in the instruction).
- For all the problems, please use **`valgrind`** tool to confirm the proper memory management. Use the command:

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes
--num-callers=20 --track-fds=yes ./01.o
```

where `01.o` is the name of tested binary file

Stage I. Implementing base class structure with basic operators

1. Develop a class `smartArray` with the following members: (5p)

```
public:
  int *elements        // dynamic array, memory is allocated in the
                       // constructor. Use *this to access size member.
  int length()         // returns the length of the array
  smartArray()         // default constructor, sets size to 0
  smartArray(int size)     // constructor, allocates the memory for
                           // *elements, if size==0, then no memory allocation
                           // is done
  ~smartArray()        // destructor
private:
   int size            // number of the elements in the array
```

2. Overload << and >> operators (15p)

- overload operator **<<** to print the array contents in the following format:
  `[33,66,23,63,75,23]` (example)
- overload operator **>>** to read the array elements from the keyboard. Format:
  `enter element 1:`
  `enter element 2:`
  etc.

3. Implement relational operators (10p)

- overload == operator. Two **`smartArray`** objects are the same (== returns true) when they have the same sizes and the same elements on the same positions.
- overload **!=** operator, use call to == operator and invert the logic.

4. Implement the array resize (20p)

- implement the public member function **`void resize(int newsize)`** that will resize the **`*elements`** array.
- Steps that need to be taken:
  - Allocate memory of the new array of size **`newsize`**
  - Copy elements from old **`*elements`** array
    - If the size of the new array is bigger than the current **`*elements`** array, then fill the additional elements with 0 value
    - If the size of the new array is smaller than the current **`*elements`** array, then copy only **`newsize`** elements from the current **`*elements`** array
    - Delete the memory currently assigned to **`*elements`**
    - Remap **`*elements`** pointer to the new array

5. Overload [] operator (10p)

- Overload **`[]`** operator, so you can access **`*elements`** through **`[]`** operator, instead of through **`*elements`** member.
  Example:

```
smartArray myArray(100);
myArray.elements[5]      // access through *elements
myArray[5]               // access through [] operator
```

6. Copy constructor and = operator (20p):

Because the **`smartArray`** contains dynamic array as a member, overloading **`+,-,*,/,&`** requires implementing copy constructor and overloading **`=`** operator.

a) Implement copy constructor (10p)
b) Implement **`=`** operator (10p)

7. Implement concatenation (20p)

- Overload the **`&`** operator to concatenate two arrays
- If the size of **`smartArray A`** is 10 and the size of **`smartArray B`** is 20, then the resulting **`smartArray C`** has the size of 30 and contains elements from both **`A`** and **`B`**.

8. Implement the following overloads: (20p)

- overload operator `+` so elements in two arrays are added. E.g.
  ```
  newArray[0]=a[0]+b[0]
  newArray[1]=a[1]+b[1]
  ```
  etc.
- overload operator `-` so elements in two arrays are subtracted. E.g.
  ```
  newArray[0]=a[0]-b[0]
  newArray[1]=a[1]-b[1]
  ```
- overload operator `*` so elements in two arrays are multiplied. E.g.
  ```
  newArray[0]=a[0]*b[0]
  newArray[1]=a[1]*b[1]
  ```
- overload operator `/` so elements in two arrays are divided. E.g.
  ```
  newArray[0]=a[0]/b[0]
  newArray[1]=a[1]/b[1]
  ```

For operators `+ - * /` consider also `smartArray` objects `A` and `B` of different sizes. The resulting array must have the size of the longer of `smartArray` objects `A` and `B`. Supply the missing elements of the shorter array with `0`s for `+` and `-`, and `1`s for `*` and `/`.

Simple version: implement these operators only for arrays of the same length (-15p)

Test for stage 1:

- Declare `smartArray` object `s1` of size 15, fill with values using `>>` operator
- Declare `smartArray` object `s2` of size 10, fill with values using `>>` operator
- Declare `smartArray` object `s3` using default constructor. Do: `s3 = s1+s2` and print `s3` using `<<` operator
- Declare `smartArray` object `s4` using default constructor. Do: `s4 = s1 - s2` and print `s4` using `<<` operator
- Declare `smartArray` object `s5` using default constructor. Do: s5 `= s1 * s2` and print `s5` using `<<` operator
- Declare `smartArray` object `s6` using default constructor. Do: `s6 = s1 / s2` and print `s6` using `<<` operator
- Test if `s1!=s2`, output the result using `cout`
- Resize `s2` to 15, fill the element 11-15 using `[]` operator
- Declare `smartArray` object `s7` using default constructor. Do: `s7 = s1 & s2` and print `s7` using `<<` operator

Stage II. Implementing the smartArray class as a template

1. Implement the class from stage 1. as the class template (25p)
Thus the `smartArray` will not handle just integers, but any numerical type requested.

Test for stage 2:

- Declare `smartArray` object `s8` of size 5 and type `double` (use class template). Directly access `*elements` member to fill values (use `for` loop and `cin`)
- Write the contents of the `smartArray` (use `for` loop, `cout`, and direct access to `*elements`)
- Declare `smartArray` object `s9` of size 6 and type `double` (use class template). Directly access `*elements` member to fill values (use `for` loop and `cin`)
- Write the contents of the `smartArray` (use `for` loop, `cout`, and direct access to `*elements`)

Stage III. Answering the question (5p)

Answer: for the class template, can you use operators defined in part 1? Why?

Submission:

For the stage I, test each operator separately (including `valgrind` test). If your operator is not working correctly, then remove the code for that operator. Please don't send operator functions that you know they aren't working properly.

Include the following elements in your submission: (`rid` = your rebel id)

Problem	Element	File
Stage I	Code of your program (for stage 1)	rid_1.cpp file
Stage II	Code of your program (for problem 2)	rid_2.cpp file
Stage III	text file with the answer to the question	rid_3.txt file
	Summary of the submission	
	Summary: 2 cpp files and 1 txt file, submit them to the WebCampus (add all the files as the single submission). Remember about proper names of the files!	