UNLV UNIVERSITY OF NEVADA, LAS VEGAS

Homework Assignment

Class:	CS202	**Semester:**	Fall 2019
Assignment type:	Homework assignment	**Due date:**	10/15/19
Assignment topic:	Pointers	**Assignment no.**	4
Delivery:	WebCampus – cpp files		

Goal

Practice the pointers and dynamic memory allocation

Input to the program

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

Procedure for the implementation

- There are four problems in this assignment.
- Each problem is separate and must be implemented in a separate cpp file

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

Problem 1. Simple pointers usage (20%)

1. Write a program, that will be reading double values into the dynamic array
   - Ask user how many numbers user wants to store in the array, store this number in ***asize***
   - declare two pointers of double type: **p_min*, **p_max*
   - read numbers in the loop, that has the following steps:
     - read a number into array
     - update **p_min* and **p_max*, to point to minimum and maximum elements in the array. If either **p_min* or **p_max* are updated, then write: "new minimum value at address ***newaddress***, now pointing to value: ***value****"*
     - display:
       "iteration number: " *number of current iteration*
       "read number: " *number that was read in current iteration*
       "current array elements: " *all array elements, separated by commas*
       "current minimum = " *value of double pointed by* *p_min "at address: " *address stored in* *p_min
       "current maximum = " *value of double pointed by* *p_max "at address: " *address stored in* *p_max
   - start numbering of iterations from 0

Example output for single iteration: (assuming, that already read 5 elements: 7, 15, 2, 3 and 11):

```
Iteration number: 5
Please enter element number 5: 19
Updating p_max to address 0x69583867, now pointing to value: 19
Read number: 19
Current array elements: 7,15,2,3,11,19
Current minimum = 2 at address 0x695834
Current maximum = 19 at address 0x69583867
```

Problem 2. Accessing classes with pointers (25%)

Steps to implement the program: (add other steps if necessary).

- Declare class _Student_, with the following public members: _name_(_string_), _id_(_int_), _gpa_(_double_)
- Create dynamic array ___sArr___ of _Student_ objects, size of the array is entered by from keyboard (like in problem 1)
- Declare pointer _*current_ of type _Student_
- Read all the student data from the keyboard. Access the array element using _*current_ pointer (e.g. before writing to the array element, assign this element's address to _*current_ and access the element through _*current_ pointer only).
- Declare _Student_ object named _*BestGpaOfSemester_ using ___new___ operator
- Declare _Student_ object named _*LowestGpaOfSemester_ using ___new___ operator
- Iterate over the ___sArr___ and find the student with the highest _gpa_. **Copy** his data into _*BestGpaOfSemester_ object (i.e. when values in ___sArr___ change, the value in * ___BestGpaOfSemester___ does not change).
- Iterate over the ___sArr___ and find the student with the lowest _gpa_. **Copy** his data into _*LowestGpaOfSemester_ object (i.e. when values in ___sArr___ change, the value in * ___LowestGpaOfSemester___ does not change).
- Print the whole content of the ___sArr___, one record per line. Use [] to access elements, include address of the array and address of each element.
- Delete the ___sArr___ array
- Print the *___BestGpaOfSemester___ student (include address of the object)
- Print the *___LowestGpaOfSemester___ student (include address of the object)

Example output:

```
Please input the size of the array: 4
Enter student 1 name: _____
Enter student 1 id: _______
Enter student 1 gpa: ______
Enter student 2 name: _____
Enter student 2 id: _______
Enter student 2 gpa: ______
Enter student 3 name: _____
Enter student 3 id: _______
Enter student 3 gpa: ______
Enter student 4 name: _____
Enter student 4 id: _______
Enter student 4 gpa: ______

Student list: (array address: 0xd96cc5a0)
Student 1: John Smith, 2000838696, 3.66 (at address 0xd96cc5a0)
Student 2: Anna White, 1000436353, 3.89 (at address 0xd96cc600)
Student 3: Paul Doe, 2000222423, 3.00   (at address 0xd96cc660)
Student 4: Mary Green, 2000281323, 3.80 (at address 0xd96cc6c0)
```

```
Array deleted.

Student with highest GPA:
Anna White, 1000436353, 3.89      (at address 0xd96cc6e2)

Student with lowest GPA:
Paul Doe, 2000222423, 3.00        (at address 0xd96cc6f6)
```

Problem 3. Pointers arithmetic (35%)

Steps to implement the program:

- Ask the user, about the *int* array size, store it in ***asize*** variable
- Create the dynamic array of integers ***DArr***, with the size of *asize*
- Fill the array: *n-th element receives the value n*(n+1)*
- Print the whole array, separate elements with commas
- Declare pointer _*current_, point it to the first element of the array
- Create infinite loop. During each loop iteration, do the following:
  - Ask the user, which **relative** element user wants to access. This means, that you are asking user: how many positions to the left or to the right of the current element, is the element you want to jump to located?
  - Each time user enters location outside of the range, set _*current_ to the first element of the array

    Example: (_*current_ points to the first element of the array)
    - Array contents: 0,2,6,12,20,30,42,56,72,90,110
    - User enters: **5**
    - Program prints (in one line): address of the ***DArr***, address of _*current_ element, address of the last element
    - Program prints the element of index 5: i.e. 6th element to the right from the first element in the array: **30**, program prints: "current value=30, current position: 5/9" (x/y: x=position of the current element (index of the element) – **calculate using array address and *current address**, y = *asize*)
    - User enters: -2
    - Program prints (in one line): address of the ***DArr***, address of _*current_ element, address of the last element
    - Program prints: "current value=12, current position: 3/9"
    - User enters: **30**
    - Program prints (in one line): address of the ***DArr***, address of _*current_ element, address of the last element
    - Program prints: *Position set to 0.*
  - Note that you have to update the pointer _*current_ each time user enters the value

    Quit your program, when user enters -1111
- Note, that you **must** use the addresses to calculate the integer position of _*current_ ("current position", also denoted as *x* above). This is "pointer arithmetic". You can't have separate variable, where you store the position. Use *addr_1* – *addr_2* to calculate the offset.

Sample output:

```
enter array size: 10
0,2,6,12,20,30,42,56,72,90,
enter relative pos: 5
DArr=0x1810010 current=0x1810024 last=0x1810038
current value=30, current position=5/9
enter relative pos: -2
DArr=0x1810010 current=0x1810016 last=0x1810038
current value=12, current position=3/9
enter relative pos: 30
Position set to 0.
```

current pos = 0

new pos = 0 + 5 = 5

current pos = 5

new pos = 5 - 2 = 3

current pos = 3

current pos = 0

Problem 4. Pointers in classes (20%)

Write a program:

- Declare a class *DArr*, with the following members:

  public:
  - double *arr // pointer to the array
  - int size // size of the array
  - void displayElements() // displays all elements of the array, uses * operator
  - double* getMax() // returns the pointer to maximum number from the array
  - DArr() // constructor (see description below)
  - ~DArr() // destructor (see description below)

Constructor *DArr*():

- Asks user for the size of the array
- Dynamically creates the array ***arr*** of the size provided
- Prints "Constructor: allocating X bytes of the memory" (calculate X knowing the size of *double* and number of elements in the array)
- Fills the array with random *double* values, from the range 0..1000. Include fraction part in the randomization process.
- For calculations, use *sizeof()* function to determine the size of single element

Destructor ~*DArr*():

- Prints: "Destructor: freeing X bytes of the memory" (calculate X knowing the size of *double* and number of elements in the array)
- Deallocates the memory allocated by constructor

- The *displayElements*() function **must** access elements using *() operator, do not use [] operator.
- The *getMax*() function **must** access elements using *() operator, do not use [] operator.

Test:

- Create object d1 of *DArr*
- Display all the elements of *DArr* using *displayElements*() function
- Get the maximum element of *DArr* using *getMax*() function

Test 2:

- In *main()* function, create an array of *DArr* objects (prompt user for its size)
- Display all elements from the array, for each element of the array display all the double values, along with max double

Sample output:

```
Enter the array size: 20
Constructor: allocating 160 bytes of memory
array elements:
311.23,751.60,16.11,224.53,813.89,497.33,199.64,832.93,846.21,195.90,429.22,5
08.34,684.29,30.56,574.12,116.54,997.55,157.89,667.63,614.14
max element: 997.55 at the address 0x0398493
Destructor: freeing 160 bytes of memory
```

Submission:

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

Problem	Element	File
1	Code of your program (for problem 1)	rid_1.cpp file
2	Code of your program (for problem 2)	rid_2.cpp file
3	Code of your program (for problem 3)	rid_3.cpp file
4	Code of your program (for problem 4)	rid_4.cpp file
	Summary of the submission	
	Summary: 4 cpp files, submit them to the WebCampus (add all the files as the single submission). Remember about proper names of the files!	