UNLV UNIVERSITY OF NEVADA, LAS VEGAS

Homework Assignment

Class:	CS202	**Semester:**	Fall 2018
Assignment type:	Homework assignment	**Due date:**	11/30/18
Assignment topic:	Stacks, Queues	**Assignment no.**	6
Delivery:	WebCampus – cpp files and txt file		

Goal

Practice the use of linked lists and stacks

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 I. Recursion (20p)

1. Write a recursive function, *power*, that takes as parameters two integers *x* and *y* such that *x* is nonzero and returns x^y. For $y \geq 0$:

$$power(x,y) = \begin{cases} 1 & if\ y = 0 \\ x & if\ y = 1 \\ x * power(x, y-1) & if\ y > 1 \end{cases}$$

If $y < 0$:

$$power(x,-y) = \frac{1}{power(x,y)}$$

Prompt user for *x* and *y*. If user enters non-integers, then catch the exception and prompt again.

Problem II. Stack class template (30p)

Implement template class for the stack. The stack is of the size of *n* elements, and *n* is given as parameter to the constructor. Implement the following class *myStack*:

```
public:
  void push(<Type>) // puts the integer element onto the stack
  <Type> pop()      // retrieves the element from the top of the
stack
  void disp()       // prints the entire stack
private:
  int stackPointer  // points to the top free spot in the stack.
  <Type> *elements
```

Use a dynamic array. Throw exceptions when:

- **pop** function was used when stack is empty
- **push** function was used when stack is full

Catch exceptions in **main()** function, rethrow in **pop()** and **push()**

Prepare a menu (see sample output). Clear screen at the beginning of each iteration (before printing the menu). To clear the screen on *bobby* (this might not work on other systems) use the following:

```
#include <stdlib.h>
… // some code...
system("clear");
```

Sample output/operation:

```
Stack:
Menu:
1. push element
2. pop element
3. exit
Enter: 1
Enter value: 5

Stack: 5
Menu:
1. push element
2. pop element
3. exit
Enter: 1
Enter value: 7

Stack: 5,7
Menu:
1. push element
2. pop element
3. exit
Enter: 1
Enter value: 3
```

```
Stack: 5,7,3
Menu:
1. push element
2. pop element
3. exit
Enter: 1
Enter value: 11

Stack: 5,7,3,11
Menu:
1. push element
2. pop element
3. exit
Enter: 2

Stack: 5,7,3
Menu:
1. push element
2. pop element
3. exit
Enter: 2

Stack: 5,7
Menu:
1. push element
2. pop element
3. exit
Enter: 2
Popped element: 7

Stack: 5
Menu:
1. push element
2. pop element
3. exit
Enter: 2

Stack: 5
Menu:
1. push element
2. pop element
3. exit
Enter: 2
can't pop from empty stack
error operating the stack at position 0

Stack:
Menu:
1. push element
2. pop element
3. exit
Enter: 3
```

Problem III. Linked Lists (20p)

Use single-linked list, forward-created. Each node describes a record info for a car. Car can be added at any position and any car can be removed.

Node:

```
int id
string make
int price
int year
Car *next   // link to the next element
```

In **`main()`** function, write a menu with the following options:

`1. add car` - add new node at specified position. Automatically assign new id.
`2. remove car` - remove node, prompt for id of the car to remove
`5. exit`

- Maintain a variable, where you store id numbers, so each newly added car will automatically receive new sequential id that was not assigned to any car before.
- Write the list of cars during each loop execution.
- Provide proper deletion of the memory (both when option 2 is used and when option 5 is used)
- add logic that car can be added only on proper position. If the position exceeds the size of the list, then add the element at the end of the list.

Sample output/operation

```
CAR MANAGEMENT
Car List:
----------
Options:
1. Add car
2. Remove car
5. Exit
Enter: 1
Enter position: 1
Enter make: Ford
Enter price: 5000
Enter year: 2011


CAR MANAGEMENT
Car List:
100   Ford        5000        2011
----------
Options:
1. Add car
2. Remove car
5. Exit
Enter: 1
```

```
Enter position: 2
Enter make: GMC
Enter price: 4500
Enter year: 2010
CAR MANAGEMENT
Car List:
100   Ford        5000        2011
101   GMC         4500        2010
----------
Options:
1. Add car
2. Remove car
5. Exit
Enter: 1
Enter position: 2
Enter make: Toyota
Enter price: 7000
Enter year: 2013


CAR MANAGEMENT
Car List:
100   Ford        5000        2011
102   Toyota      7000        2013
101   GMC         4500        2010
----------
Options:
1. Add car
2. Remove car
5. Exit
Enter: 2
Enter id of car to remove: 101

CAR MANAGEMENT
Car List:
100   Ford        5000        2011
102   Toyota      7000        2013
----------
Options:
1. Add car
2. Remove car
5. Exit
Enter: 5
```

Problem IV. Doubly linked list (30p)

Use the list from problem III as a base and extend it to the doubly linked list. Each node describes a record for a car. Node contains the following car info:

```
int id
string make
int price
int year
```

Extend the class for car element to be the doubly linked list.

Menu:
1. add car - add new node to the end of the list. Automatically assign new id.
2. remove car - remove node, prompt for id of the car to remove
3. print list (sort by newest added first)
4. print list (sort by oldest added first)
5. exit

Clear screen at the beginning of each iteration (before printing the menu).

Submission:

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

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