Lab Sheet 2 - Stack

"It is better to have 100 functions operate
on one data structure than 10 functions
on 10 data >structures."
- *Alan Perlis*

1. Write a program to implement a stack to store *integer* numbers. Name of the class must be `Stack_Int`. Use *integer array* to represent the stack. The *size* of the stack can be fixed in advance as part of the class definition.

   Data types:

Name of data type	**Description**
`size_type`	to represent the size of the stack. Use unsigned int.
`value_type`	to represent the type of the data stored on the stack. Use signed int.

   Data members:

Name of data type	**Description**
`data`	Array for holding the elements. The type of array is `value_type` and its size if `stack_size`.
`stack_size`	Declare it as a constant static member of the class.
`top_of_stack`	To hold the current top of the stack. A value of -1 represents that the stack is empty. Make it a signed integer type.

   Constructors:

Constructor	**Description**
`Stack_Int()`	To create an empty stack. The top_of_stack will have a value -1.

   Member functions:

Function name	**Description**
`bool empty() const;`	To check whether the stack is empty.
`size_type size() const;`	Returns the number of elements on the stack.
`value_type& top();` `const value_type& top() const;`	Returns a reference to the top element of the stack. Implement const and non-const versions.
`void push(const value_type& x) throw (overflow_exception);`	Add element to the top of the stack.
`void pop() throw (underflow_exception);`	Removes the element from the top of the stack, effectively reducing the size by one. The value of this element can be retrieved before being popped by calling top( ).
`friend ostream& operator<< (ostream& out, const Stack_Int& s);`	Overload the output operator (<<).

   Create the *exception classes* by the names `overflow_exception` and `underflow_exception` to denote the *overflow* and *underflow* conditions. The class `overflow_exception` should be derived from the *exception* class and `underflow_exception` should be derived from the `runtime_error` class (of `stdexcept` header file).

   ```
   class overflow_exception: public exception { … }
   class underflow_exception: public runtime_error { … }
   ```

   Override the `what()`function (`virtual const char* what() const throw()`) in both the above classes to denote the reason for the exception. You will have to create a private string data object in the `overflow_exception` class to hold the reason for the exception. But such a declaration is not required in `underflow_exception` class definition since `runtime_error` already has a string data object.

   Create files `Stack_Int.h` and `Stack_Int.cpp` files to hold the code for the stack. The `overflow_exception` and `underflow_exception` classes can also be defined in the `Stack_Int.h` header file along with the `Stack_Int` class definition. The `main` function can be in `main.cpp`. Make sure to include *header guards* for the header files (either using `#ifndef` or using `#pragma once`)

In the *driver* program (the `main` function), use the following technique to *populate* and *print* the stack.

a) `push <int>` to push an integer to the stack.
b) `pop` to pop an element
c) `print` to print the current elements of the stack.
d) `exit` to exit the program.

Any other command (other than `push`, `pop`, `print` and `exit`) should be reported as an error. Catch the *overflow* and *underflow exceptions* in the *main* function and report the errors to the user of the stack class.

Sample run for the program:

```
push 10
push 20
print
+-----+
|   20|<-- Top
+-----+
|   10|
+-----+

push 30
print
+-----+
|   30|<-- Top
+-----+
|   20|
+-----+
|   10|
+-----+

pop
print
+-----+
|   20|<-- Top
+-----+
|   10|
+-----+
pop
pop
print
STACK EMPTY

pop
Stack Underflow
push 10
push 20
push 30
push 40
push 50
push 60
push 70
push 80
push 90
push 100
push 110
Stack Overflow

pus
Wrong command
popp
Wrong command
printt
Wrong command
exit
```

2. Convert the above class to a *template* stack class that can store any kind of data types and not just integers. The name of the class should be `Stack`. The interface of the class remains the same.

   Create a stack of `strings` in the `main` function.

   How do you compile your program containing template classes after dividing it between *header* and *cpp* files? What compilation model would you follow? Does your compiler support the `export` keyword or do you plan to `include` *cpp* files in header files?

   How do you create friend functions in a template class?

   Sample run of program:

```
push Dog
push Cat
push Apple
print
+-----+
|Apple|<-- Top
+-----+
|  Cat|
+-----+
|  Dog|
+-----+

pop
print
+-----+
|  Cat|<-- Top
+-----+
|  Dog|
+-----+

exit
```