Lab 9

Objectives

The purpose of this lab is to reinforce stack and queue template class in C++.

Requirements

Implement a class named stack_pair that provides a pair of stacks. Make the class a template class. So, you will have two files: stack_pair.h and stack_pair.template, following the style of the text.

The basic idea is that two stacks can share a single static array. This may be advantageous if only one of the stacks will be in heavy use at any one time.

- The class should have various methods to manipulate the stack:

  T pop_a()

  T pop_b()

  These methods pop the top of the respective stack and return the value. Use assert to cause the program to fail if an attempt is made to pop from an empty list.

- void push_a(T item)

  void push_b(T item)

  These methods push the value onto the respective stack. Use assert to cause the program to fail if an attempt is made to push to a full stack.

- size_t size_a()

  size_t size_b()

  These methods return the size of the respective stack.

- bool is_empty_a()

  bool is_empty_b()

  These methods return true if the respective stack is empty and return false if the stack is not empty.

- is_full_a()

  is_full_b()

  These methods return true if the respective stack has no more space for new data to be pushed. The methods return false if the respective stack has space available. Because of the implementation, these two methods will always return the same thing.

The implementation of the stack should be done with a single static array. Define the CAPACITY of that array as a static constant of size 30, to start with. Variables will be needed to keep track of the top of each stack, we'll call them top_a and top_b. For illustrations, we'll use the following diagram. The CAPACITY is only 16. Notice that top_a and top_b indicate where the next element will go when pushing.

201	202	203										104	103	102	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			top_a								top_b				

Popping an element from stack a returns 203 and leaves this state:

201	202											104	103	102	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		top_a									top_b				

Pushing 105 onto stack b leaves this state:

201	202										105	104	103	102	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		top_a								top_b					

Here both stacks are empty.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
top_a															top_b

Here the stacks are full. Notice that top_a and top_b have ‘passed by’ each other.

201	202	203	204	205	206	207	109	108	107	106	105	104	103	102	101
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
						top_b	top_a								

Testing

Test your work by writing four programs. Two of these will be copies of others. When you are instructed to check something, use an if-statement and, if the expected behavior fails, print a message and terminate the program using the exit function.

Write a program that will push a few values onto the a stack, then push a few values onto the b stack. The program should pop the values on the a stack and check that they are correct as they are popped. The program should pop the values on the b stack and check that they are correct as they are popped.

Copy the previous program and increase the number of values pushed on a and or b to get a total number of CAPACITY values pushed between the two stacks. The program should not fail and both stack-full functions should return true.

Copy the previous program and push one more value onto one of the stacks. The program should fail.

Write a program that will check that popping from an empty stack terminates the program.