

```
1 /*-----
2 Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
3 file: dstack.hpp
4
5 -----*/
6
7 /*-----
8 This file has class definition
9 -----*/
10
11 /*-----
12 Definition of routines of dstack class
13 -----*/
14
15 /*-----
16 Constructor
17 -----*/
18 template <typename T>
19 dstack<T>::dstack(int size) :_sp(0), _stack(size){
20     if (display()) {
21         cout << "in dstack constructor:" << endl;
22     }
23 }
24
25 /*-----
26 Destructor
27 -----*/
28 template <typename T>
29 dstack<T>::~~dstack() {
30     if (display()) {
31         cout << "In dstack Destructor " << endl;
32     }
33     _sp = 0;
34     //calls _stack array destructor here from 0 to _sp-1
35 }
36
37 /*-----
38 Get num elements of the stack
39 -----*/
40 template <typename T>
41 int dstack<T>::num_elements() const {
42     return _sp;
43 }
44
45 /*-----
46 Is stack empty
47 -----*/
48 template <typename T>
49 bool dstack<T>::isempty() const {
50     return _sp ? false : true;
51 }
52
53 /*-----
54 Is stack full
55 Our stack can never be full.
56 -----*/
57 template <typename T>
58 bool dstack<T>::isfull() const {
59     return false;
60 }
61
62 /*-----
63 Get the top of the stack.
64
65 Stack is the owner of the object. Note that the object is
66 returned by alias so that NO copy constructor is called.
```

```

67
68 The caller has two options:
69
70 T& obj1 = s.top() ;
71 T  obj2 = s.top() ;
72
73 In first case, if obj1 is changed, he is really changing the stored obj in s
74 In second case, if obj2 is changed he is changing the copied object.
75 It has no effect on the object that was in the stack. obj2 will die
76 at the end of its scope.
77 -----*/
78 template <typename T>
79 T& dstack<T>::top(){
80     if (isempty()) {
81         assert(0);
82     }
83     return (_stack[_sp - 1]);
84 }
85
86 /*-----
87 pop: Remove the top element from the stack.
88 NOTHING is returned.
89 This object still resides in darray which dies
90 when destructor of darray is called. But user
91 has no access to it.
92 -----*/
93 template <class T>
94 void dstack<T>::pop() {
95     if (isempty()) {
96         assert(0);
97     }
98     --_sp;
99 }
100
101 /*-----
102 pop: Push the element to the top of the stack
103 My stack is never full. I can always push to my stack
104
105 Note that b is copied into stack. The copied object
106 is the property of stack and not the user.
107 -----*/
108 template <class T>
109 void dstack<T>::push(const T& b) {
110     _stack[_sp++] = b;
111 }
112
113 /*-----
114 apply function pf to each element of the stack.
115 -----*/
116 template <class T>
117 void dstack<T>::for_each_element_of_stack_from_top_to_bottom(void(*pf) (T& c)) {
118     for (int i = _sp - 1; i >= 0; i--) {
119         pf(_stack[i]);
120     }
121 }
122
123 //EOF
124
125
126

```