

```
1 /*-----
2 Copyright (c) 2015 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 c, bool d) :_display(d), _sp(0), _stack(c,d){
20     if (display()) {
21         cout << "in dstack constructor:" << endl;
22     }
23 }
24
25 /*-----
26 Constructor
27 -----*/
28 template <typename T>
29 dstack<T>::dstack(bool d) :_display(d), _sp(0), _stack(50, d){
30     if (display()) {
31         cout << "in dstack constructor:" << endl;
32     }
33 }
34
35 /*-----
36 Destructor
37 -----*/
38 template <typename T>
39 dstack<T>::~~dstack() {
40     if (display()) {
41         cout << "In dstack Destructor " << endl;
42     }
43     _sp = 0;
44     //calls _stack array destructor here from 0 to _sp-1
45 }
46
47 /*-----
48 Get num elements of the stack
49 -----*/
50 template <typename T>
51 int dstack<T>::num_elements() const {
52     return _sp;
53 }
54
55 /*-----
56 Is stack empty
57 -----*/
58 template <typename T>
59 bool dstack<T>::isempty() const {
60     return _sp ? false : true;
61 }
62
63 /*-----
64 Is stack full
65 Our stack can never be full.
66 -----*/
```

```

67 template <typename T>
68 bool dstack<T>::isfull() const {
69     return false;
70 }
71
72 /*-----
73 Get the top of the stack.
74
75 Stack is the owner of the object. Note that the object is
76 returned by alias so that NO copy constructor is called.
77
78 The caller has two options:
79
80 T& obj1 = s.top() ;
81 T  obj2 = s.top() ;
82
83 In first case, if obj1 is changed, he is really changing the stored obj in s
84 In second case, if obj2 is changed he is changing the copied object.
85 It has no effect on the object that was in the stack. obj2 will die
86 at the end of its scope.
87 -----*/
88 template <typename T>
89 T& dstack<T>::top(){
90     if (isempty()) {
91         assert(0);
92     }
93     return (_stack[_sp - 1]);
94 }
95
96 /*-----
97 pop: Remove the top element from the stack.
98 NOTHING is returned.
99 This object still resides in darray which dies
100 when destructor of darray is called. But user
101 has no access to it.
102 -----*/
103 template <class T>
104 void dstack<T>::pop() {
105     if (isempty()) {
106         assert(0);
107     }
108     --_sp;
109 }
110
111 /*-----
112 pop: Push the element to the top of the stack
113 My stack is never full. I can always push to my stack
114
115 Note that b is copied into stack. The copied object
116 is the property of stack and not the user.
117 -----*/
118 template <class T>
119 void dstack<T>::push(const T& b) {
120     _stack[_sp++] = b;
121 }
122
123 /*-----
124 apply function pf to each element of the stack.
125 -----*/
126 template <class T>
127 void dstack<T>::for_each_element_of_stack_from_top_to_bottom(void(*pf) (T& c)) {
128     for (int i = _sp - 1; i >= 0; i--) {
129         pf(_stack[i]);
130     }
131 }
132

```

133 //EOF

134

135

136