```
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: dstack.hpp

-----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has class definition
-----------------------------------------------------------------*/

/*----------------------------------------------------------------
Definition of routines of dstack class
-----------------------------------------------------------------*/

/*----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
template <typename T>
dstack<T>::dstack(int size) :_sp(0), _stack(size){
  if (display()) {
    cout << "in dstack constructor:" << endl;
  }
}

/*----------------------------------------------------------------
Destructor
-----------------------------------------------------------------*/
template <typename T>
dstack<T>::~dstack() {
  if (display()) {
    cout << "In dstack Destructor " << endl;
  }
  _sp = 0;
  //calls _stack array destructor here from 0 to _sp-1
}

/*----------------------------------------------------------------
Get num elements of the stack
-----------------------------------------------------------------*/
template <typename T>
int dstack<T>::num_elements() const {
  return _sp;
}

/*----------------------------------------------------------------
Is stack empty
-----------------------------------------------------------------*/
template <typename T>
bool dstack<T>::isempty() const {
  return _sp ? false : true;
}

/*----------------------------------------------------------------
Is stack full
Our stack can never be full.
-----------------------------------------------------------------*/
template <typename T>
bool dstack<T>::isfull() const {
  return false;
}

/*----------------------------------------------------------------
Get the top of the stack.

Stack is the owner of the object. Note that the object is
returned by alias so that NO copy constructor is called.
```

```

The caller has two options:

T& obj1 = s.top() ;
T  obj2 = s.top() ;

In first case, if obj1 is changed, he is really changing the stored obj in s
In second case, if obj2 is changed he is changing the copied object.
It has no effect on the object that was in the stack. obj2 will die
at the end of its scope.
-----------------------------------------------------------------*/
template <typename T>
T& dstack<T>::top(){
  if (isempty()) {
    assert(0);
  }
  return (_stack[_sp - 1]);
}

/*-----------------------------------------------------------------
pop: Remove the top element from the stack.
NOTHING is returned.
This object still resides in darray which dies
when destructor of darray is called. But user
has no access to it.
-----------------------------------------------------------------*/
template <class T>
void dstack<T>::pop()  {
  if (isempty()) {
    assert(0);
  }
  --_sp;
}

/*-----------------------------------------------------------------
pop: Push the element to the top of the stack
My stack is never full. I can always push to my stack

Note that b is copied into stack. The copied object
is the property of stack and not the user.
-----------------------------------------------------------------*/
template <class T>
void dstack<T>::push(const T& b)  {
  _stack[_sp++] = b;
}

/*-----------------------------------------------------------------
apply function pf to each element of the stack.
-----------------------------------------------------------------*/
template <class T>
void dstack<T>::for_each_element_of_stack_from_top_to_bottom(void(*pf) (T& c))  {
  for (int i = _sp - 1; i >= 0; i--) {
    pf(_stack[i]);
  }
}

//EOF



```