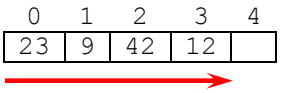
Drop-out Stack **Templated Container**

This project involves implementing a simple, semi-dynamic data structure: an array-based *drop-out stack*.

A drop-out-stack is simply a stack variant that modifies the `push()` operation so that the bottom element in the stack is dropped out (lost) if the stack is full. Applications of drop-out-stacks include history lists and undo lists in applications. Users might want to store all sorts of things as tasks in a drop-out stack. To accommodate that, we will require that the implementation use C++ templates.

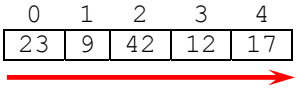
As you might expect, a drop-out-stack is often implemented so that there is a fixed, maximum capacity. That will be the case in this assignment, although the capacity will be specified as a parameter to a constructor. The underlying physical structure could be either a dynamically-allocated array or a linked list of nodes. In this assignment, you **must** use an array, allowing the top to "float" as elements are added and removed. You will never shift the contents of the array to perform either a `Push()` or a `Pop()`.

A drop-out-stack may exhibit a phenomenon similar to "queue creep" when held in an array. Suppose we have a drop-out-stack with a capacity of five, and have pushed four elements onto the stack:

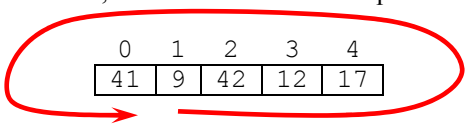


Here `Top` would be 4 (the index of the cell to be used on the next push operation), and the bottom element is at index 0.

Now suppose we push one more, filling the stack:



And now suppose we push another. Where does it go? In an array-based drop-out-stack, the new element will be pushed over the bottom element, using the array in a circular fashion:



Here the new element was 41, and now `Top` is 1 and the bottom element is at index 1.

Template Declarations

This assignment will be evaluated by a test harness. In order to accommodate that, we must constrain your implementation rather severely. In short, your templates must conform **exactly** to the declarations given later in this specification. The details of implementation are up to you, but if you modify or omit either data members or functions then your code will almost certainly not compile with the test harness.

You must also place the entire implementation of the template within a separate file, using the name shown in the template header file that follows.

Note the `DropOutStackT` template will declare a `friend` class as shown below:

```
#ifndef DROPOUTSTACKT_H
#define DROPOUTSTACKT_H
#include <iostream>
#include <iomanip>
#include <new>
using namespace std;

template <typename T> class DropOutStackT {

friend class Javert;

private:
   T*  Stk;                // pointer to stack array
   unsigned int Cap;       // capacity (dimension) of stack array
   unsigned int Top;       // index for next push
   unsigned int Size;      // # of elements stored

public:
   DropOutStackT(unsigned int Capacity = 0);
   DropOutStackT(const DropOutStackT<T>& Source);
   DropOutStackT<T>& operator=(const DropOutStackT<T>& RHS);

   bool Push(const T& Elem);          // insert Elem at Top
   bool Pop(T& Elem);                 // remove top-most element and return
   T* const Peek() const;             // provide access to top-most element
   void Clear();                      // reset stack to empty state

   bool isEmpty() const;              // check for empty stack
   //bool isFull()const;              // check for "full" stack
   unsigned int Capacity() const;     // report stack capacity

   void Display(ostream& Out) const;  // display stack contents

   ~DropOutStackT();                  // destroy stack
};

// import template implementation code:
#include "DropOutStackTImplementation.cpp"

#endif
```

Do NOT implement the isFull*() method

Design and implementation requirements

There are some explicit requirements, in addition to those on the *Coding Style* page of the course website:

- You must implement a C++ template (obviously), conforming to the given interface.
- Code from the C++ STL or any other non-approved library may **NOT** used in your implementation. Violation of this restriction will result in a score of **zero**!
- You must handle copy issues correctly. We will certainly test for this. Failures in your implementation will almost certainly result in program crashes and scores of zero.
- You may assume that any data type stored in your template will handle its own copy issues correctly (as it must).
- You may assume that any data type stored in your template will provide `operator<<()` and `operator==()`.
- You must properly allocate and deallocate memory, as needed.
- You must provide feedback to the client when an operation fails. The given prototypes of the member functions indicate where that is necessary. Under no circumstances should any of the template functions, other than `Display()`, write output.

The `Display()` function must write the contents of the drop-out-stack in the following format:

```
Capacity:  4
    0:  45
    3:  35
    2:  25
    1:  15
```

The capacity of the stack is reported, followed by the stored elements, listed from the top-most down, with a label indicating the array index at which each element is stored. If the stack is empty, write:

```
Capacity:  4
    Stack is empty
```

Challenge: When a `Pop()` operation is performed, the index `Top` must be moved backwards to the previous cell. If `Top` is zero, it must be reset to the largest valid index. This can obviously be done by treating that situation as a special case and using a selection statement. The challenge is this: implement `Pop()` so that the adjustment of `Top` does not require any special case logic. **Hint:** what is the additive inverse of 1 if you're doing circular arithmetic on the integers {0, 1, 2, …, n-1}? This challenge is **NOT** to be discussed on the CS web forums.

Testing Procedure for the DropOutStack

The following is an outline of a recommended testing procedure for your template code. This is essentially the testing strategy that will be employed by the Curator test harness. You should design your own testing strategy appropriately. Students who wish to compose test harness code and share it with others in the course may do so by sending the test harness code to their instructor for prior approval **before** posting to the CS web class forum.

I. Test deep copy logic
- create a stack and push some elements on it
- pass it to a function by value
- check contents of local stack against an accurate copy
- check contents of original stack against an accurate copy

Note: if there are ANY mismatches here, the test is aborted.

II. Test push/pop/peek/clear
- create a stack and push some elements on it *pop and check a few elements from the top down
- push the stack full, then push one more so it wraps *check the top element *pop and check a few elements
- pop it 'til it's empty and check the last element
- clear it
- push elements on until it wraps around again
- pop until it unwraps
- check contents

III. Test a big stack
- create a large stack
- push elements on until it wraps around
- check some elements
- pop until it wraps back
- check some elements
- clear it

Evaluation:

You should document your implementation in accordance with the *Code Style Page* on the course website. It is possible that your program will be evaluated for documentation and design, as well as for correctness of results. If so, your submission that achieved the highest score may be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

Note well: if you make two or more submissions that are tied for the highest score, the earliest of those will be graded. There will be absolutely no exceptions to this policy!

Moral: code and document to meet requirements from the beginning, rather than planning to retrofit documentation into a finished program.

Note that the evaluation of your project may depend substantially on the quality of your code and documentation.

What to turn in and how:

Submit the C++ `DropOutStackTImplementation.cpp` file containing your template implementation to the Curator System. Submit **only** the template implementation file. Submit **nothing** else. Instructions for submitting to the Curator are given in the *Student Guide* at the Curator website: http://www.cs.vt.edu/curator/. Be sure to follow those instructions carefully The submission URL is also posted at the Curator website. You will be allowed to submit your solution up to **six** times.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for the file:

```
//     On my honor:
//
//     - I have not discussed the C++ language code in my program with
//       anyone other than my instructor or the teaching assistants
//       assigned to this course.
//
//     - I have not used C++ language code obtained from another student,
//       or any other unauthorized source, either modified or unmodified.
//
//     - If any C++ language code or documentation used in my program
//       was obtained from another source, such as a text book or course
//       notes, that has been clearly noted with a proper citation in
//       the comments of my program.
//
//     - I have not designed this program in such a way as to defeat or
//       interfere with the normal operation of the Curator System.
//
//     - I have neither given nor received unauthorized aid in the
//       completion of this assignment.
//
//     <Student Name>  <Student PID>
```

Failure to include this pledge in a submission is a violation of the VT Honor Code.