CS 291/242: Object-Oriented Design and Programming

Programming Assignment 2
Due Wednesday, Feb 1st, 2006

The second assignment gives you an opportunity to use more advanced features of C++. In particular, a limitation of the `Array` class from the first assignment is that it only works on arrays of chars. To generalize this behavior, change the `Array` class to be a parameterized type by using C++ **templates**. Templates allow you to parameterize the class with the desired data type.

For this assignment, I would like everyone to use exception handling, now that we've had a chance to discuss it a bit more. Please let me know if you have any questions about how to use it in your program.

In addition, we'll now change the semantics of the array so that if you try to `set()` beyond the end of the array the array will grow automatically. We'll also add a `resize()` method that can be called to grow the array explicitly (I recommend using `resize()` in your implementation of `set()`.

Finally, I would like students taking the class as CS 291 to implement STL-like iterators for the Array. This is optional for students taking the class as CS 215.

The following is the new interface for the array.

```
// File Array.h

// This will be an array of <T>, i.e., it is a parameterized type.

template <class T>
class Array
{
public:
  // = Initialization and termination methods.

  // Define a "trait"
  typedef T TYPE;

  // Dynamically create an uninitialized array.  Throws
  // <std::bad_alloc> if allocation fails.
  Array (size_t size);

  // Dynamically initialize an array.  Throws
  // <std::bad_alloc> if allocation fails.
  Array (size_t size, const T &default_value);

  // The copy constructor (performs initialization).  Throws
  // <std::bad_alloc> if allocation fails.
  Array (const Array &s);

  // Assignment operator performs an assignment by making a copy of
  // the contents of parameter <s>, i.e., *this == s will return true.
  // Note that if the <max_size_> of <array_> is >= than <s.cur_size_>
  // we can copy it without reallocating.  However, if <max_size_> is
  // < <s.cur_size_> we must delete the <array_>, reallocate a new
  // <array_>, and then copy the contents of <s>.  Throws
  // <std::bad_alloc> if allocation fails.
  void operator= (const Array &s);

  // Clean up the array (e.g., delete dynamically allocated memory).
  ~Array (void);

  // = Set/get methods.

  // Set an item in the array at location index.  If <index> >
  // <s.cur_size_> then <resize()> the array so it's big enough.
  // Throws <std::bad_alloc> if resizing the array fails.
  void set (const T &new_item, size_t index);

  // Get an item in the array at location index.  Throws <std::out_of_range>
  // of index is not <in_range>, else returns 0.
  void get (T &item, size_t index) const throw (std::out_of_range);

  // Returns a reference to the <index> element in the <Array> without
  // checking for range errors or resizing the array.
  const T &operator[] (size_t index) const;

  // Set an item in the array at location index without
  // checking for range errors or resizing the array.
  T &operator[] (size_t index);

  // Returns the current size of the array.
  size_t size (void) const;

  // Compare this array with <s> for equality.  Returns true if the
  // size()'s of the two arrays are equal and all the elements from 0
  // .. size() are equal, else false.
  bool operator== (const Array<T> &s) const;

  // Compare this array with <s> for inequality such that <*this> !=
  // <s> is always the complement of the boolean return value of
  // <*this> == <s>.
  bool operator!= (const Array<T> &s) const;

  // Change the size of the array to be at least <new_size> elements.
  // Implementation note: the actual size of the array may be expanded
  // more (e.g., resize by a factor of two) to optimize future resizes.
  // Throws <std::bad_alloc> if allocation fails.
  void resize (size_t new_size);

  // = Iterator forward definitions (only for CS 291 students).
  typedef Array_Iterator<T> iterator;

  // Get an iterator that points to the beginning of the list
  iterator begin (void);

  // Get an iterator that points to the end
  iterator end (void);

private:
  // Returns true if <index> is within range, i.e., 0 <= <index> <
  // <cur_size_>, else returns false.
  int in_range (size_t index) const;

  // Add other helper methods you see fit here...

  // Maximum size of the array, i.e., the total number of <T> elements
  // in <array_>.
  size_t max_size_;

  // Current size of the array.  This starts out being == to
  // <max_size_>.  However, if we are assigned a smaller array, then
  // <cur_size_> will become less than <max_size_>.  The purpose of
  // keeping track of both sizes is to avoid reallocating memory if we
  // don't have to.
  size_t cur_size_;

  // Pointer to the array's storage buffer.
  T *array_;
};
```

Note that you'll have to change the definition of the methods in the Array.cpp file to utilize the template and exception syntax.

You can get the "shells" for the program from `www.cs.wustl.edu/~schmidt/cs215/assignment2`.

The Makefile, `main.cpp`, and `Array.h` files are written for you. All you need to do is edit the `Array.cpp` and `Array.i` files to add the methods that implement the Array ADT.

If you are enrolled in CS 215 please use the shells that are in the `ugrad` directory at the URL above. If you are enrolled in CS 291 please use the shells that are in the `grad` directory at the URL above.