Topics: Container ADT, class templates, command line arguments

Create a container ADT named CS2Container. It allows a user to store key/data pairs, and to retrieve those pairs sequentially, or directly by key. The user doesn't need to worry about where the data is or how it's stored.

UI

We give the user a struct which describes a data element in the container:

```
template <class T, class U>
struct element
{
	T key;
	U data;
};
```

The user instantiates a container by:

```
	CS2Container <key type, data type> containername (num);
```

where

- key type - the data type of the key field
- data type - the data type of the data field
- containername - the name of the container object
- num - the maximum number of elements in the container instance

Functions available to the user are:

- `bool push_back (element)`
- `bool retrieve_front (element)`
- `bool retrieve_back (element)`

Each push_back adds an element to the container. New data is always placed at the back. The “front” element is the element that was added to the container the longest time ago; the “back” element is the most recent element. A new element added via push_back becomes the new back element. push_back is the only way for a user to insert data into the container; there is no “push_front” function available. If there is no more room in the container, push_back returns a value of false; otherwise, it returns true to indicate successful insertion.

retrieve_front and retrieve_back retrieve copies of the data (both the key and data fields) from the front and back elements of the container, respectively. If data was retrieved, the functions return a value of true. If the container is empty, the return value should be false to indicate unsuccessful retrieval.

- `bool retrieve_by_key (element)`

The user provides the function with a local element object containing the key value to search for. retrieve_by_key performs a linear search through the elements of the container, looking for the key. If found, the function copies the data field from the matching container element into the data field in the element provided by the user, and the return value is set to true. If a matching element was not found, either because the key value was not present, or because the container is empty, the return value is set to false. Current position in the container is set to the element following the element containing the key.

- `int size ()`
- `int max_size ()`

Returns the current size of the container (number of elements), and the maximum size of the container, respectively.

- `bool start ()`

This function establishes current position at the first element in the container. If the container is empty, the start function returns a value of false. Otherwise, the return value is set to true.

- `bool get_next (element)`

get_next returns the element in the container (key and data fields) where current position is located. If there is no next element, because there are no more used elements in the container, get_next returns a value of false. If an element was retrieved successfully, the return value is true.

<u>Coding</u>

In a single header file you have:

- the element struct, templated
- the CS2Container class definition, similarly templated
- the CS2Container member functions, templated

Since everything in this list is templated, you'll have one header file containing everything related to the class. You'll need two type names, one for the key, and one for the data.

You'll have a main program which will test your class. The number of elements to use comes in as a command line argument. The argument is a three-character field, padded on the left with leading zeroes if necessary. Refer to the class discussion to code this.

data members:

p	pointer to a dynamically allocated array, of type element, and of size number of elements. Since p is a pointer to an array, you'll be able to refer to elements of the array by subscripting p.
number of elements	from the command line. This integer tells you the maximum size of the container.
number of used elements	starts at 0, and is incremented each time the user inserts a new element into the container. This integer tells you the size of the container at the present moment. number of used elements will never be greater than number of elements.
current position	the element number for the element where current position is established. A get_next call retrieves the element at current position. Before returning, get_next updates current position to "point" at the next element in sequence. retrieve_by_key sets current position at the element following the one retrieved.

member functions:

constructor	dynamically allocate an array of n elements. Set number of used elements to zero.
destructor	delete allocated memory.
push_back, size, max_size	described above.
retrieve_front and retrieve_back	the element provided by the user should be passed by reference. Data is given to the user by moving it from the container into the element's key and data fields.

retrieve_by_key	performs a linear search of the container, and returns the element with the specified key. Stops when an element with the specified key is found, or when the end of the used portion of the container is reached. The element object needs to be passed by reference. If the element is found, retrieve_by_key sets current position at the next element, so that a subsequent series of get_next calls retrieve elements sequentially from there.
get_next	returns false if the user tries to get_next past the number of elements currently used in the container. In that sense, returning "false" is similar to the file system returning EOF when an attempt is made to read past end of file.

As good as all this may sound, and as convenient as it may seem for the user, there are some obvious shortfalls with the CS2Container container:

- Once inserted, there is no way for the user to remove an element.
- The user cannot alter the sequence of elements after they're added.
- There is no expansion of the container. If the initial size is too small, too bad.
- If the initial size is too big, too bad, we waste memory.
- The method used for retrieval by key (simple linear search) is inefficient.
- The container does not order the elements in ascending or descending sequence. There might be advantages to maintaining an ordered list, but we're ignoring them.

The supplied main expects your command line argument to be set to 005.