CSE 241 Assigment #5 - Extendable Array Spring 2015

Due date: Apr. 27 20:00

Extendable Integer Array

Write ExtendableArray, an array of integers class. It supports the following functionality:

1. Conceptually, an ExtendableArray is an infinite length array initialized to all zeros.
2. A default ExtendableArray constructor should allocate memory space to hold 2 integers.
3. It should be expandable. Assigning to an element of the array that hasn't been allocated yet should transparently reallocate the memory space. I.e. operator[] may need more memory space than currently allocaed, which results in memory reallocation. Note that your object should not allocate memory space more than necessary.
4. Write operator[] so that it returns an ElementRef (see ExtendableArray.h provided).
5. If the index to operator[] used on the right side of an assignment operation is outside the currently allocated space, the operation returns 0 without reallocating more memory space (e.g., having int x = a[10000000000]; shouldn't blow out your memory).

Header Files

You are not allowed to make changes to the provided ExtendableArray.h. Your class should be implemented in ExtendableArray.cpp. Provided source files are located in /home/cse241/assign5.

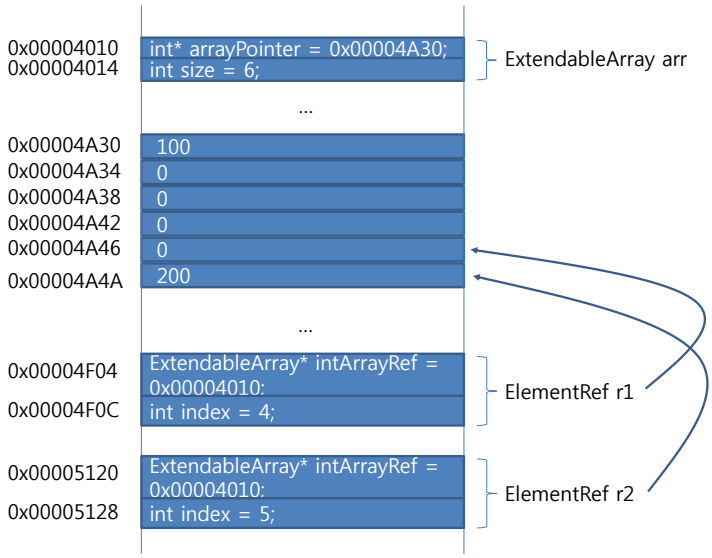
ExtendableArray.h

```
//
// Expandable integer array class
//
#ifndef EARRAY_H
#define EARRAY_H

#include <ostream>
using namespace std;

class ExtendableArray;

class ElementRef
{
private:
   ExtendableArray *intArrayRef;  //pointer to the array
   int index;                     // the index to the element
public:
   ElementRef( ExtendableArray& theArray, int i );
   ElementRef( const ElementRef& other );  // copy constructor
   ~ElementRef();

   ElementRef& operator=( const ElementRef& rhs );
   ElementRef& operator=( int val );

   operator int() const;
};


class ExtendableArray
{
```

```
private:
   int *arrayPointer;   // integer array pointer
   int size;            // the size of the array
public:
   ExtendableArray();   // allocates memory space for 2 integers
   ExtendableArray( const ExtendableArray& other );  // copy constructor
   ~ExtendableArray();  // destructor

   ExtendableArray& operator=( const ExtendableArray& rhs );

   ElementRef operator[]( int i );

   friend class ElementRef;  // ElementRef class can access my private members

   friend ostream& operator<< (ostream& o, const ExtendableArray& t){
       for(int i=0;i<t.size;i++){
           o << t.arrayPointer[i] << " ";
       }
       return o;
   }
};

#endif // EARRAY_H
```

Note: Reference

The following figure shows the relationship of the two classes.



Note that "reference" is the representation of storage, i.e., a variable name, used by a compiler. When we use a reference on the left side of `=`, for example, `myRef = 1;` 1 will be stored in the storage referenced by `myRef`. This is because the variable `myRef` references a memory space. We call this L-value of a variable (or reference). If you use the reference on the right side of =, for example, `int var = myRef;` the value stored in the memory space referenced by `myRef` will be copied to the memory space referenced by `var`. We call the value stored in a reference variable "R-value".

Suppose you have an ExtendableArray objet `arr`. Since `arr[1]` can be used for either L-value or R-value, `ExtendableArray::operator[]` must return a reference of the integer. However as your `arrayPointer` is pointing to a sequence of integers that do not have "references", we use `ElementRef` class object that pretends to be a reference of an integer in the array.

For example, suppose `arr[1]` returns a `ElementRef` class object. If the `ElementRef` object of `arr[1]` is used on the right side of the `operator=()`, the ElementRef class object should return an integer value so that its R-value can be used. If the `arr[1]` is used on the left side of `operator=()`, the right side integer value should be stored in the memory space referenced by `ElementRef` object.

Q:Why ExtendableArray::operator[] returns ElementRef?

A:
Suppose ExtendableArray::operator[] is returning a reference to an integer (int&). Then how would you increase the size of the dynamic array? The size of the array should be increased if the ElementRef is used as L-value. If ElementRef is used as R-value, you should return 0 if the array subscript is out of bound. Suppose the size of memory pointed by arr.arrayPointer is smaller than 1000. If so,

```
arr[10000] = 1;
```

the above code should allocate more memory spaces since the array subscript is larger than the available memory size.

```
cout << arr[10000] << endl;
```

but the above line should not reallocate memory as it is used as R-value, but it should print out 0.

If operator[] is returning a reference to an integer (int&), how would you detect whether the reference is used as R-value or L-value? This is why we need a class ElementRef.

Test Driver

The file driver1.cpp will contain the function main(). The purpose of this file is to make various calls to the functions in your classes to test them as thoroughly as possible. How you choose to use this file is entirely up to you. Keep in mind that any functions included in this file are for your own basic testing purposes only. You should continually make changes to this file so that it tests your code while you are making chnages to your class.

driver1.cpp

```
#include "ExtendableArray.h"
#include <iostream>
using namespace std;

void stuff_20(ExtendableArray arr)
{
  for (int i=0; i < 20; i++) {
    arr[ i ] = i;
  }

  cout << arr << endl; // 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
}

int main() {
  ExtendableArray a1;
  for (int i=0;i<20;i++)
    a1[i] = i;
  cout << a1 << endl; //0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

  if(1){
      ExtendableArray a2;
      for (int i=0;i<10;i++)
        a2[i+5] = a1[i];
      cout << a2 << endl; //0 0 0 0 0 0 1 2 3 4 5 6 7 8 9

      a1 = a2;
```

```
      for (int i=0;i<10;i++)
        a2[i] = i;

      cout << a1 << endl; //0 0 0 0 0 0 1 2 3 4 5 6 7 8 9
      cout << a2 << endl;  //0 1 2 3 4 5 6 7 8 9 5 6 7 8 9
  }

  cout << a1 << endl; //0 0 0 0 0 0 1 2 3 4 5 6 7 8 9

  ExtendableArray a3;

  a3[0] = 1;

  cout << a3 << endl; // 1 0

  stuff_20(a3);

  cout << a3 << endl; // 1 0

  cout << a3[2147483647] << endl; // 0

  return 0;
}
```

ExtendableArray.h and driver1.cpp files are in /home/cse241/assign5 directory. You should copy the two files into your current directory using the following command.

```
$ cp /home/cse241/assign5/*  .
```

After copying them, you should create ExtendableArray.cpp file. As ExtendableArray.h contains only the prototype declaration of class member functions, you must implement the member function body in ExtendableArray.cpp. Note that ExtendableArray.h must be included in ExtendableArray.cpp and driver1.cpp so that both cpp files know the classes and their member functions.

In order to compile the two cpp files, run g++ compiler with the list of cpp file names.

```
$ g++ driver1.cpp ExtendableArray.cpp
```

Submitting Your Code

You should submit ExtendableArray.cpp file only. Please do not submit the executable file or header file of your program. Turn in your project using the "oopsubmit" command as follows:

```
$ oopsubmit assign5 ExtendableArray.cpp
```

If you do not follow this submission guideline, you will lose 10 points out of 100.

You should also submit a hard copy of your code to TA. Your report must have a cover page with your student ID and name. In the report, your code must be well commented to explain your class. 10 points will be given for the report. Also, the sample input and the output of your implementation must be included in the report.