

[Assignment 5 & 20152008, KUSDavletov Ernar]

[OOP]Assign5

Student ID: 20152008

Name: KUSDavletov Ernar

Date: 2017.05.21

1. Descriptions

ExtendableArray is an array of integers class, which supports some functionalities. ExtendableArray is an infinite length array initialized to all zeros. In our case, default ExtendableArray constructor allocate memory space to hold 2 integers. It is expandable and consider the cases when operator located on the right side and on the left side. So, if we assign to the element of the array that hasn't been allocated yet, it transparently reallocates the memory space. 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.

2. Code

```
//Kusdavletov Ernar, 20152008
#include "ExtendableArray.h"
#include <iostream>

using namespace std;

ElementRef::ElementRef(ExtendableArray& theArray, int i){ //Constructor of ElementRef
    index = i; //assigning the input values to ElementRef object (index and intArrayRef)
    intArrayRef = &theArray;
}

ElementRef::ElementRef(const ElementRef& other){ //Copy constructor of ElementRef
    index = other.index; //assigning the values of one ElementRef object to another
    intArrayRef = other.intArrayRef;
}

ElementRef::~ElementRef(){} //Destructor should be empty

ElementRef& ElementRef::operator=(const ElementRef& rhs){ //operator overloading when ElementRef assigned
    if (index >= intArrayRef->size){ //if index is bigger than array size, expand array
        ExtendableArray newArray; //creating temporary variable
        newArray.size = index + 1; //assigning the size
        newArray.arrayPointer = new int [newArray.size]; //creating arrayPointer with size (index + 1)
        for (int i = 0; i < newArray.size; i++){
            if (i < intArrayRef->size){ //if arrayPointer previously existed, then copy it
                newArray.arrayPointer[i] = intArrayRef->arrayPointer[i];
            }
            else{ //if not, assign 0 value
                newArray.arrayPointer[i] = 0;
            }
        }
        //assigning newArray to intArrayRef
        *intArrayRef = newArray;
        intArrayRef->size = newArray.size;
    }
    if (rhs.index >= rhs.intArrayRef->size){ //if rhs index is out of range assign 0 value
        intArrayRef->arrayPointer[index] = 0;
    }
}
```

```

    else{ //else assign rhs arrayPointer value at index
        intArrayRef->arrayPointer[index] = rhs.intArrayRef->arrayPointer[rhs.index];
    }
    return *this;
}

ElementRef& ElementRef::operator=(int val){ //operator overloading when value assigned
    if (index >= intArrayRef->size){ //if index is bigger than array size, expand array
        ExtendableArray newArray; //creating temporary variable
        newArray.size = index + 1; //assigning the size
        newArray.arrayPointer = new int [newArray.size]; //creating arrayPointer with size (index + 1)
        for (int i = 0; i < newArray.size; i++){
            if (i < intArrayRef->size){
                newArray.arrayPointer[i] = intArrayRef->arrayPointer[i]; //if arrayPointer previously existed, then copy it
            }
            else{ //if not, assign 0 value
                newArray.arrayPointer[i] = 0;
            }
        }
        //assigning newArray to intArrayRef
        *intArrayRef = newArray;
        intArrayRef->size = newArray.size;
    }
    intArrayRef->arrayPointer[index] = val; //assigning integer value
    return *this;
}

ElementRef::operator int() const{ //int operator
    if (index < intArrayRef->size){ //if index smaller than size then return arrayPointer[index]
        return intArrayRef->arrayPointer[index];
    }
    else{ //else if it is out of range return 0 value
        return 0;
    }
}

ExtendableArray::ExtendableArray(){ //ExtendableArray default constructor which allocates memory to hold 2 integers
    size = 2;
    arrayPointer = new int[size]; //allocating memory
    for (int i = 0; i < size; i++)
        arrayPointer[i] = 0; //assigning 0 values
}

ExtendableArray::ExtendableArray(const ExtendableArray& other){ //Copy constructor of ExtendableArray
    size = other.size; //making sizes equal
    arrayPointer = new int[size]; //allocating memory
    for (int i = 0; i < size; i++)
        arrayPointer[i] = other.arrayPointer[i]; //assigning values
}

ExtendableArray::~ExtendableArray(){ //Destructor
    delete [] arrayPointer;
}

```

```

ExtendableArray& ExtendableArray::operator=(const ExtendableArray& rhs){
// = operator overloading when ExtendableArray object assigned
    if (this == &rhs) //if they are same just return *this
        return *this;
    delete [] arrayPointer; //deleting the arrayPointer memory
    size = rhs.size; //assigning the size
    arrayPointer = new int[size]; //allocating the memory of size
    for (int i = 0; i < size; i++)
        arrayPointer[i] = rhs.arrayPointer[i]; //assigning the values
    return *this;
}

ElementRef ExtendableArray::operator[](int i){ //[ ] operator overloading
    return ElementRef{*this, i}; //return the constructor of ElementRef with values (*this and i)
}

```

3. Sample input

First sample input

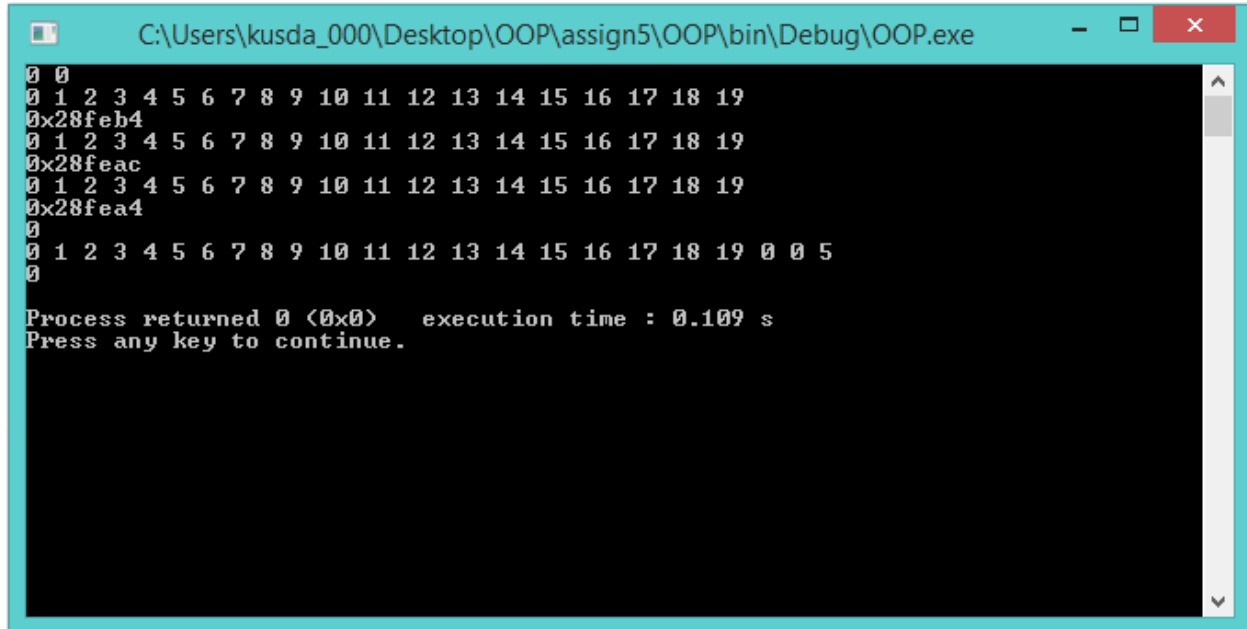
```
int main() {
    ExtendableArray a1;
    cout << a1 << endl; //0 0
    for (int i = 0; i < 20; i++)
        a1[i] = i;
    cout << a1 << endl; //0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
    cout << &a1 << endl;
    ExtendableArray a2(a1);
    cout << a2 << endl; //0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
    cout << &a2 << endl;
    ExtendableArray a3;
    a3 = a1;
    cout << a3 << endl; //0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
    cout << &a3 << endl;
    cout << a3[1000000000] << endl; // 0
    a1[22] = 5;
    cout << a1 << endl; //0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 0 5
    a1[10] = a2[1000000000];
    cout << a1[10] << endl; //0
    return 0;
}
```

Second sample input

```
int main() {
    ExtendableArray a1;
    cout << a1 << endl; //0 0
    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
    ElementRef b1(a1, 2);
    cout << b1 << endl; //2
    cout << &b1 << endl;
    ElementRef b2(b1);
    cout << b2 << endl; //2
    cout << &b2 << endl;
    ElementRef b3(a1, 5);
    cout << b3 << endl; //5
    cout << &b3 << endl;
    b3 = b1;
    cout << b3 << endl; //2
    cout << &b3 << endl;
    b3 = 777;
    cout << b3 << endl; //777
    return 0;
}
```

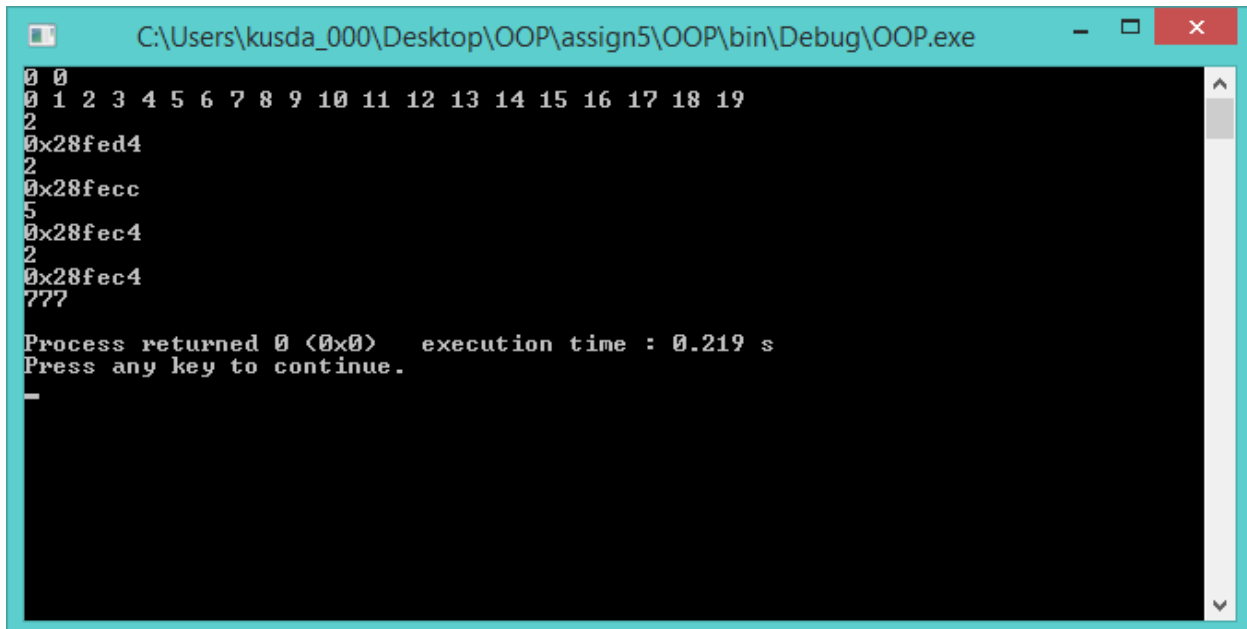
4. Sample output

First sample output



```
C:\Users\kusda_000\Desktop\OOP\assign5\OOP\bin\Debug\OOP.exe
0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0x28feb4
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0x28feac
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0x28fea4
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 0 5
0
Process returned 0 (0x0)   execution time : 0.109 s
Press any key to continue.
```

Second sample output



```
C:\Users\kusda_000\Desktop\OOP\assign5\OOP\bin\Debug\OOP.exe
0 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
2
0x28fed4
2
0x28fecc
5
0x28fec4
2
0x28fec4
777
Process returned 0 (0x0)   execution time : 0.219 s
Press any key to continue.
```