A Dynamic Array Class
Programming Workshop 2 (CSCI 1061U)

Faisal Qureshi
http://faculty.uoit.ca/qureshi
University of Ontario Institute of Technology

Topics

- `new` and `delete` for dynamic memory allocation and deallocation
- [] operators: _overloading [] operators

Task 1

Example 1

Use the `new` and `delete` operators to allocate an integer variable.

```cpp
int main()
{
    int i;       // allocates an integer
    int *pi = &i;  // pointer to an int

    cout << "Value of i = " << i << endl;
    cout << "Value of i = " << *pi << endl;
    cout << "Address of i = " << &i << endl;
    cout << "Address of i = " << pi << endl;

    int *pi2 = new int;  // Allocates memory to store an integer
                         // This memory is allocated on the heap
    cout << "Value of memory at pi2 = " << *pi2 << endl;
    cout << "Address of the allocated memory = " << pi2 << endl;

    delete pi2;  // Important to deallocate any/all allocated memories

    return 0;
}
```

Example 2

Use the `new` and `delete` operators to allocate a dynamic array.

```cpp
int main()
{
    int* arr = new double[10];  // Allocates a double array with 10 slots
```


```cpp
for (int i=0; i<10; ++i) {
    cout << arr[i] << endl;
}

delete [] arr; // Deleting the allocated array. Notice the extra []
    // after delete. This is important when dealing
    // with dynamically allocated arrays.
return 0;
}

Task 3

Create an **Arr** class that implements a dynamically allocated array for storing double values.

**Using Arr class**

For example, we will be able to use this array as follows:

Arr a(3); // Allocate an Arr instance that stores three double values

a[0] = 1; // Setting values
a[1] = 2.3;
a[2] = 3.43;

// No deallocation is needed!

**Using built-in C++ arrays**

Notice how the above will be accomplished using built-in C++ arrays.

double *a = new double[3]; // Allocating an array for storing three double values

a[0] = 1; // Setting values
a[1] = 2.3;
a[2] = 3.43;

delete [] a; // Don't forget to deallocated this array

**Code**

The following code implements the **Arr** class

```cpp
#include <iostream>
#include <cassert>
using namespace std;

class Arr
{
    protected:
        double * _data; // always a good practice to have members (data)
        // hidden (either protected or private)
        int _n; // size of the array, so we know how big it is
```
public:
Arr(int n) : _n(n) // save the passed n, which indicates the size of the array
{
    _data = new double[_n]; // allocate an array of size n (or _n)
}

Arr(const Arr& a)
{
    _n = a._n; // save the size of "this" equal to that of instance "a"
    _data = new double[_n]; // allocate "this" (_n) slots
    for (int i=0; i<_n; ++i) // copy values
    {
        _data[i] = a._data[i];
    }
}

~Arr()
{
    delete [] _data;
}

double& operator[](int i)
{
    return _data[i];
}

const double& operator[](int i) const
{
    return _data[i];
}

};

int main()
{
    Arr a(5); // a is an instance of type Arr
    // Arr must have a constructor that takes an int
    // this instance "a" has 5 slots for storing
    // double values.
    a[1] = 2.3;
    cout << a[1] << endl;

    Arr b(a); // b is an instance of Arr
    // Arr must have a constructor that takes an instance of type
    // Arr (why, because a is an instance of Arr)
    // Eureka! this object Arr must have a copy constructor
    // instance b is an exact copy of a

    return 0;
}