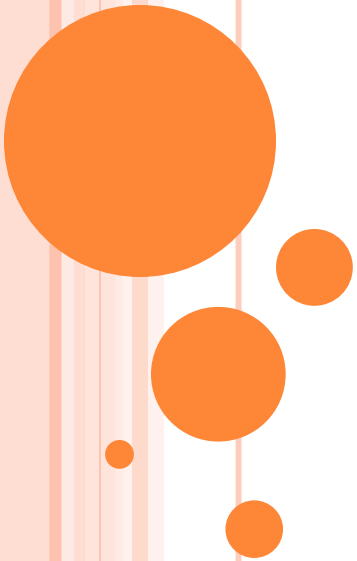

Dynamic Class Members

CS A150 - C++ Programming 1

Dynamic Variables

- We have already seen how pointers can create dynamic variables:

```
int *p = new int;
int *a = new int[];
```

- We can certainly use dynamic data in a class, but we need to make sure we also delete the data.

Deleting Dynamic Member Variables

- *Dynamically*-allocated variables
  - Do not go away until "deleted"
- **Destructor**
  - Automatically called when object is out-of-scope
  - Default version removes *only* ordinary variables, *not* dynamic variables
- If **pointers** are private member data
  - **Destructor** will de-allocate them when object is destroyed

A Pointer as a Class Member

- The project given as an example, **DArray**, creates objects that contain:
  - A **pointer** to an **int** (it will point to an **array** of **int**)
  - An **int** that stores the **capacity** of the array
  - An **int** that stores the **number of elements** in the array

```
class Darray
{
        ...
private:
        int *a;
        int capacity;
        int numOfElements;
};
```

A Pointer as a Class Member (cont.)

- The **constructor** will initialize
  - The capacity to a **default** length
  - The array to the default capacity
  - The number of elements to zero

```
Darray::DArray
{
        capacity = 50;
        a = new [capacity];
        numOfElements = 0;
}
```

A Pointer as a Class Member (cont.)

- The **destructor** will
  - Delete the array
  - Null the pointer

```
Darray::~DArray
{
        delete [] a;
        a = NULL;
}
```

A Pointer as a Class Member (cont.)

- The function **addElement** will
  - Add an element to the array
  - Increment the number of elements

```
void Darray::addElement(int newElement)
{
        a[numOfElements] = newElement;
        ++numOfElements;
}
```

  - Of course, this function should also check whether the array is full; we will take care of that in the actual project.

A Pointer as a Class Member (cont.)

- The function **compareArrays** will
  - Compare the calling object with a given object, both of type DArray
  - Return a bool value that indicates whether the two objects have the same elements in the same order.
  - Let's look at the function declaration:

```
bool compareArrays
        (const Darray& otherArray) const;
```

Is this function going to modify the **member variables** of the **calling object**?
No. Therefore, we should tag this **function** as **const**.

We are passing an **object**; therefore we need to **pass it by reference**.
Should we also use a const modifier?
We are ***not*** modifying this parameter; therefore, we need to **pass it as a const**.

A Pointer as a Class Member (cont.)

- **Implementation** of the function **compareArrays**
  - Let's look at the function definition:

```
{
    if (numOfElements != otherArray.numOfElements)
        return false;  //no need to go further

    while (idx < numOfElements)
    {
        if (a[idx] == otherArray.a[idx];
            ++idx;
        else
            return false;
    }
    return true;
}
```

Note that when calling **a[idx]** we need to specify which object we are referring to if it is not the calling object.

Example 2

- Project: DArray class

Dynamic Class Members (end)