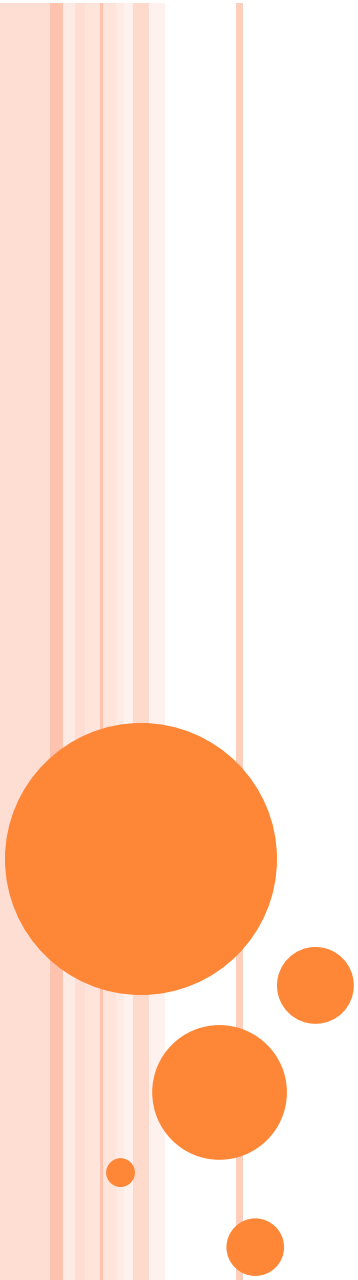


More on Overloading

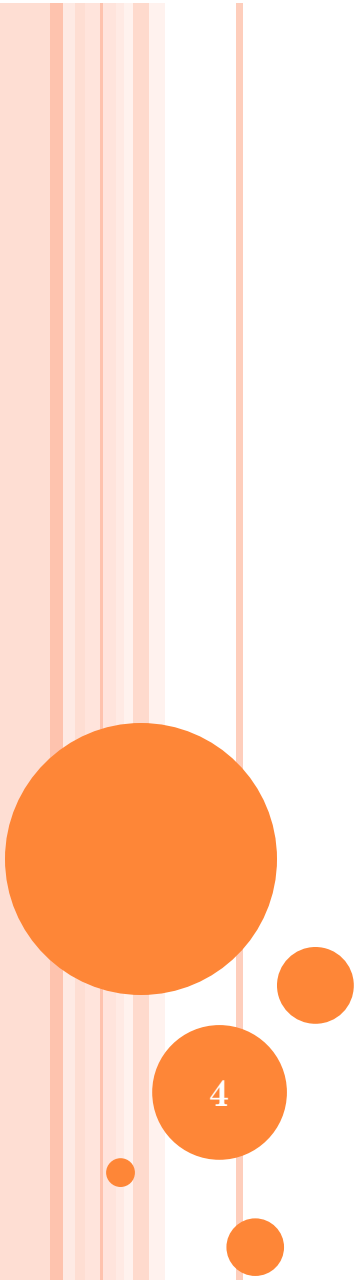
CS A250 – C++ Programming II

Operator Overloading

- We have seen how to overload operators as
  - **member** functions
  - **non-member** functions
  - **friend** functions
- **Operator overloading** (known also as **syntactic sugar**) improves code **readability**

Operator Overloading

- We will look at **overloading operators** that require a somewhat more complex implementation:
  - **Increment** and **decrement** operators
  - **Subscript** operator



Overloading Prefix and Postfix

OVERLOADING ++ AND --

- The **prefix** and **postfix** versions of the **increment** and **decrement** operators can all be overloaded.
  - Each overloaded operator has a distinct **signature**, so that the compiler is able to determine whether it is a **prefix** of a **postfix**.

The Prefix Operator

- With the class **Pair** in mind, we want to be able to use a function call such as:

```
Pair p(4,7);
++p;   // prefix
```

that will change **p** to (5, 8).

The Prefix Operator (cont.)

- Overloading the **prefix increment** (or **decrement**) **operator** as a **member function**:

```
Pair& operator++(); //declaration

Pair& Pair::operator++() //definition
{
   ++first;
   ++second;
   return *this;
}
```

The Prefix Operator (cont.)

- Overloading the **prefix increment** (or **decrement**) **operator** as a **member function**:

```
Pair& operator++(); //declaration

Pair& Pair::operator++() //definition
{
    ++first;
    ++second;
    return *this;
}
```

"**this**" is a **pointer** to the object.

The Prefix Operator (cont.)

- Overloading the **prefix increment** (or **decrement**) **operator** as a **member function**:

```
Pair& operator++(); //declaration

Pair& Pair::operator++() //definition
{
    ++first;
    ++second;
    return *this;
}
```

"**this**" is a **pointer** to the object.

We return what "**this**" is pointing to (that is, the **object**).

The Prefix Operator (cont.)

- Overloading the **prefix increment** (or **decrement**) **operator** as a **member function**:

```
Pair& operator++(); //declaration

Pair& Pair::operator++() //definition
{
    ++first;
    ++second;
    return *this;
}
```

"**this**" is a pointer to the object.

We return what "**this**" is pointing to (that is, the **object**).

Because we have **Pair &** (return by **reference**), only the **address of the object** will be returned.

The Postfix Operator

- With the class **Pair** in mind, we want to be able to use a function call such as:

```
Pair p(5,7);
p++;   // postfix
```

that will change **p** to (6, 8).

The Postfix Operator (cont.)

- Overloading the **postfix increment** (or **decrement**) **operator** as a **member function**:
  - Need to let the compiler know it is a ***postfix***
  - We add a "dummy parameter"

The Postfix Increment Operator (cont.)

```
Pair operator++( int ); //declaration

Pair Pair::operator++( int ) //definition
{
   Pair temp = *this; //make a copy of the obj

   ++first;              // modify the obj
   ++second;
   return temp;          //return the copy
}
```

The Postfix Increment Operator (cont.)

```
Pair operator++( int ); //declaration

Pair Pair::operator++( int ) //definition
{
    Pair temp = *this; //make a copy of the obj

    ++first;                // modify the obj
    ++second;
    return temp;
}
```

Why are we returning by **value** instead of reference (&)?

The Postfix Increment Operator (cont.)

```
Pair operator++( int ); //declaration

Pair Pair::operator++( int ) //definition
{
    Pair temp = *this; //make a copy of the obj

    ++first;
    ++second;
    return temp;
}
```

Why are we returning by **value** instead of reference (&)?

Because at the end of the function, **temp** will be destroyed, but by returning it by **value**, a **copy** will be created for the function that called **operator++.**

Note on Efficiency

- The additional object that is created by the **postfix increment** (or **decrement**) operator can result in a **significant performance problem** (*especially* in a loop)

- You should use the **postfix increment** (or **decrement**) operator only when the logic of the program requires post-incrementing (or post-decrementing).

EXAMPLE

- Project: Pair class

Overloading the Subscript Operator

THE SUBSCRIPT OPERATOR [ ]

- We can extract elements from an array with the **subscript operator [ ]**

```
int arr[ ] = { 10, 11, 12, 13};
cout << arr[2]; //will print 12
```

- The **subscript operator [ ]** does **not** work with objects of a class

```
DArray myArray;
//insert a few elements
cout << myArray[2]; //error!
```

OVERLOADING [ ]

- We can overload the **subscript operator [ ]** to return an element of an object of the **DArray** class at a specific index

```
//Declaration
int& operator[](int idx) const;

//Definition
int& DArray::operator[](int idx) const
{
  return a[idx];
}
```

OVERLOADING [ ] (CONT.)

- Why return by reference?

```
//Declaration
int& operator[](int idx) const;
```

- To be able to modify the value

```
DArray myArray (20);
myArray[0] = 3;
myArray[1] = 6;
```

More on Overloading (end)