



2007-02-15

Lecture #6

(Material on Test 1)

Today's Outline

'this' operator

Default Constructor

Destructor

Copy Constructor

private & public functions / members

static data members

Overloaded << and >> and = (and friend functions)

Overloaded operators (ex: +, -, <, >, etc.)

'this' operator

How does the this operator actual work?

Example:

```
Student john(“John”, “Doe”) ;  
john.RegisterForClass(“cs251”) ;
```

(program actually does something like this)

```
Student.Student(john, “John”, “Doe”) ;  
Student.RegisterForClass(john, “cs251”) ;
```

Example Class code:

```
// constructor  
Student(Student & this, string fName, string lName){ ... }  
// member function  
void RegisterForClass(Student & this, string class) { ... }
```

Default Constructor

Default Constructor:

```
Student( void ) { ... }
```

Example usage of Default Constructor:

```
Student john ; // default constructor called
```

```
Student * justin ; // NO constructor called, just a ptr created
```

```
justin = new Student() ; // default constructor called
```

Destructor

```
int main( int argc, char * argv[])  
{  
    Student john ; // default constructor called  
  
    return EXIT_SUCCESS ; // destructor called  
}
```

```
int main(int argc, char * argv[])  
{  
    Student * justin ; // no constructor, just a ptr created  
    justin = new Student( ) ; // default constructor called  
    delete justin ; // destructor called here  
    return EXIT_SUCCESS ; /// no destructor called here  
}
```

Copy Constructor

Class code:

```
// notice const ... because the “copied” student isn't changed  
Student(const Student & astudent) { ... }
```

Example usage:

```
int main( int argc, char * argv[] )  
{  
    Student john(“John”, “Doe”) ;  
    Student johncopy(john) ; // copy constructor called here  
    Student * johncopy2 ;  
    johncopy2 = new Student(john) ; // copy constructor  
  
    return EXIT_SUCCESS ;  
}
```

private & public functions / members

So far, in every example ... I've shown data members as “private” and member functions as “public” ... but that's not always the case!!!

```
class Colors{  
    public:  
        const int Red ;  
        const int Blue ;  
        const int Green ;  
        ....  
};
```

Usage:

```
Colors paint ;  
int mycolor = paint.Red ;  
int yourcolor = paint.Green ;
```

private & public functions / members

```
class Student{  
    public:  
        ...  
        double ComputeGrade( void ) ;  
    private:  
        double AdjustGradeForCurve( double ) ;  
        ...  
};
```

In this example ... we don't want the user to know about the curve, or how to adjust the grade for it, etc. ... but we would like to make our code more readable by putting the curve code into a separate function. Therefore, this is a good place for a Private member function.

static data members

There is a better, smarter way to create the color class we did before

```
class Colors{  
    public:  
        const static int Red = 1;  
        const static int Blue = 2;  
        const static int Green = 3 ;  
  
        ...  
};
```

Usage:

```
int mycolor = Colors.Green ;  
...  
if(mycolor == Colors.Red) { ... }
```

How does this look in memory?

Overloaded << and >> and = (and friend functions)

What is a friend function?

- A friend function is a way to cheat the system
- Friend functions are allowed **FULL** access to **PRIVATE** data members

```
class Student{  
    friend void TwinStudents(const Student, Student &) ;  
    private:  
        string fName ;  
        string lName ;  
        ...  
    public:  
        ....  
}; // notice below there is NO Student:: cause it's NOT a member  
    void TwinStudents(const Student a, Student & b) { ... }
```

Overloaded << and >> and = (and friend functions)

How can I make the following code work?

```
Student newguy ;  
cin >> newguy ;  
  
class Student{  
    friend ostream & operator<<(ostream &, Student &) ;  
    ...  
}  
ostream & operator<<(ostream & in, Student & s){  
    in >> s.fName ;  
    in >> s.lName ;  
    return in ;  
} // Question, why do we return in?
```

Overloaded << and >> and = (and friend functions)

How can I make the following code work?

```
Student john("John", "Doe");  
cout << john << endl;
```

```
class Student{  
    friend ostream & operator<<(ostream &, const Student &);  
    ...  
}  
ostream & operator<<(ostream & out, const Student & s){  
    out << s.fName << " , " << s.lName ;  
    return out ;  
} // Question, why do we return out?
```

Shallow Copy vs Deep Copy

```
class Student{
    public:
        string fName ;
        string lName ;
        double * grades ;
        int NumberOfGrades;
    private:
        ....
};

Student johncopy ;
Student john ;
...
johncopy = john ; // Shallow Copy! This is bad!
```

Overloaded =

How can I make the following code work?

```
johncopy = john ;
```

```
class Student{  
    public:  
        Student & operator=(const Student &) ;  
}  
Student & Student::operator=(const Student & other) {  
    this->fName = other.fName ;  
    this->lName = other.lName ;  
    this->grades = new double[other.NumberOfGrades] ;  
    this->NumberOfGrades = other.NumberOfGrades ;  
    for(int i = 0 ; i < this->NumberOfGrades; i++)  
        this->grades[i] = other.grades[i] ;  
    return this ;  
} // Question, why do we return this?
```

Overloaded operators (ex: +, -, <, >, etc.)

How can I make the following code work?

```
Book novel1 ;  
Book novel2 ;  
Book combined = novel1 + novel2 ;
```

```
class Book{  
    private:  
        string bookText ;  
    public:  
        Book & operator+(const Book &) ;  
}  
Book & Book::operator+(const Book & other) {  
    this->bookText = this->bookText + other.bookText ;  
    return this ;  
} // Question, why do we return this?
```

Overloaded operators (ex: +, -, <, >, etc.)

How can I make the following code work?

```
MyString str1 ;  
MyString str2 ;  
MyString str3 = str1 - str2 ;
```

```
class MyString{  
    private:  
        string str ;  
    public:  
        MyString & operator-(const MyString &) ;  
}  
MyString & MyString::operator-(const MyString & removeme) {  
    this->str = this->str.replace(removeme, "");  
    return this ;  
} // Question, why do we return this?
```

Overloaded operators (ex: +, -, <, >, etc.)

How can I make the following code work?

```
Student x , y ;  
if(x == y)
```

```
class Student{  
    public:  
        bool operator==(const Student &) ;  
}  
bool Student::operator==(const Student & other) {  
    if(this->fName == other.fName &&  
        this->lName == other.lName)  
        return true ;  
    else  
        return false ;  
} // note, this == might not be good? Better would be SSN or ID#
```

Overloaded operators (ex: +, -, <, >, etc.)

How can I make the following code work?

```
Student x , y ;  
if(x < y)
```

```
class Student{  
    public:  
        bool operator<(const Student &) ;  
}  
bool Student::operator<(const Student & other) {  
    if(this->age < other.age)  
        return true ;  
    else  
        return false ;  
}
```

How to look at C++ operators

```
Student john ;  
Student johncopy ;  
Student jane ;
```

```
john = johncopy ;
```

```
if(john < jane)  
    jane = john + johncopy ;
```

IS ACTUALLY INTERPRETED AS

```
john.operator=(johncopy) ;  
if(john.operator<(jane))  
    jane.operator=(john.operator+(johncopy)) ;
```

Reminder: Homework 2 due 02/20/2007 (next Tuesday) at Midnight

Reminder: Next Thursday we review for Exam 1

Exam1 = Tuesday 02/27/2007 in class