Lecture 4 - CS202

1. Copy Constructor slides
2. Topic #6 slides - Operator Overloading
   - what is it?
   - why it is important?
   - where is it used?
   - how to accomplish it
3. Example of Operator Overloading

Announcements:

What is operator overloading?

- using operators with class objects

```
cin >> variable;
```

bit shifting operator!

with operator overloading
We can have >> extract data from the input stream when used with an object of type istream &

- ability to use operators (instead of all named functions) when working with objects of a class

```
movie_list += video;   ← instead of "append"

cin >> video;  → instead of "input"
cout << video;  ← instead of "display"
```

Why is it important?

- It is actually operator overloading that allows us to have our class types seem like REAL data types, as if they were built into the language.

- Without them we couldn't

a) read cin >> anything
b) write cout << anything
c) compare "String class types"
if (string1 == string2)
↑ operator overloading

- Think about what a data type would be like if there are NO operators ?!

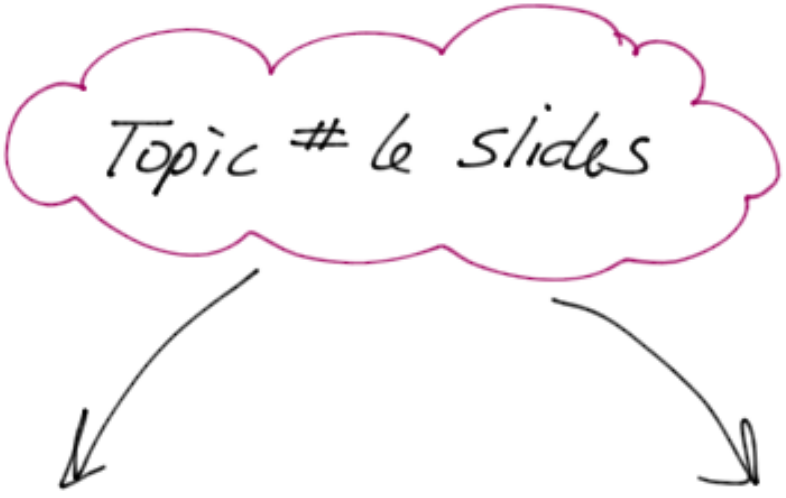
- Allows us to correct the implementation of the = operator when our classes manage dynamic memory (memberwise copy defaults --- which will result in Seg faults!)

Where is it used?

1. In classes when we want clients to use objects in the same way they might use built-in types.
2. When performing data abstraction
3. If we desire our class to be used with templates (where the same function or class can be used with ANY data type. If the operators used by a template function or class are not overloaded then your class can't be used with that code).

But... this only works IF you follow very precise rules about how the operators behave.

How is it accomplished?



As non member functions

- For all operators where the first operand is NOT an object of your class

---

returntype function(arg list);

- returntype ↑ residual value
- function ↑ operator+
- arg list ↑ 1 arg for unary, 2 args for binary operators

As Member Functions

- = [] += -= etc.
- when the first operand is Always a class type (of a class you control.

---

returntype function(arg list);

- returntype ↑ residual value
- function ↑ operator=
- arg list ↑ no args for unary operator, 1 arg for binary op.

Best Approach

Understand how the operators behave with the built in data types

```
int a, b, c;
```

```
b + c = a;
```

(crossed out)

```
a = b + c;
```

operand 1 operand 2

temp

residual value is an RVALUE
(the client program does not control the memory where the result is)

```
a = some_value;
```

operand 1 operand 2 (can be RVALUE or LVALUE)

LVALUE
(client program DOES control the memory where the result is placed)

Applying this to Operator Overloading

```
list a, b, c;

a = b + c;
```

merge or append

copy over

LVALUE EXPRESSION

RVALUE Expression

As Member Functions

```
list operator+ (const list&) const;
```

- `list`: RVALUE — Residual value (causes copy constructor to be invoked)
- `(const list&)`: Allows 2nd operand to be a constant — 2nd operand (object c)
- `const`: Allows 1st operand to be a constant

```
list & operator=(const list &);
```

- `list &`: LVALUE
- `(const list &)`: Allows us to copy FROM a constant

As Non-Member

```
list operator+ (const list &, const list &);
```

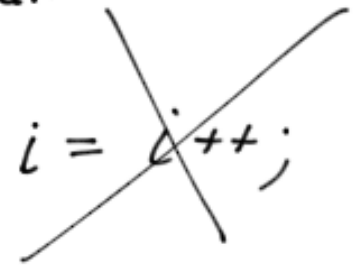
op1 op2

both can be constant objects

Trivia - Did you know?

1. `i = i` "self assignment"

2. `i = ++i;` ← why?



3. `i = f(i);`

4. `function( i, ++i );`
   `function ( i, i++);` ?

---

5. `if (this == &object)`
   pointer to the current object

6. `if (*this == object)` ?

RVALUE VS LVALUE

1. ++i — Lvalue
2. i++ — Rvalue
3. a[i] — Lvalue
4. a + b — Rvalue
5. c = d — Lvalue
6. a += b; — Lvalue

```
a = a + b;
a += b;
```

1. who controls the memory for the result
2. Can it be placed on the left side of an assignment operation?

Constant Member Functions

* any member function that does NOT modify any data members should be specified as a constant member function!

* the only member functions you can call through CONSTANT OBJECTS are constant member functions.

```
void function (const list & object)
{
    object.function();
```

constant object!

must be a constant member function!

---

* "display" should be a constant member function!

Member function Concatenation (chaining)

1. cout << "bluck" << variable;

MUST return an ostream & otherwise the next << operator will (a) syntax error
(b) bit shift
(c) use a different overloaded implementation

* Be careful - All Return Types must match expected data type for the operator's residual value.

* what about VOID ?

RVALUES and Copy Constructors

Topic 6 slides 41-43

```
string operator+ (const string &s1,
                  const string & s2)
{
    string temp
    temp.len = s1.len + s2.len +1 ;
    temp.str = new char [temp.len ];
    strcpy (temp.str, s1);
    Strcat( temp.str, s2);
    return temp;
}
```