PIC 10A 2B

TA: Bumsu Kim

UCLA

Today…

- Classes
  - Implicit and Explicit Parameters
  - Constructors and Overloading of Member Functions
- Examples with the **Matrix** Class

UCLA

C++ Classes

Concepts, Examples, and Exercise Problems

Classes (Review)

- Access Specifiers
  - Private – Members (data and methods) cannot be accessed **outside** the class
    - Inside the class interface, everything is accessible!
  - Public – Members can be accessed everywhere (e.g. in the **main** function)

- Constructor
  - A special member function used to initialize objects of its class type
  - Called only once, when an object of a certain class is created (defined)

- Accessing Members
  - Outside the class, use the dot "." operator to access the **public** data members/methods

- Accessors and Mutators
  - Any method that can modify the class object is called a mutator
  - An accessor provides the access to the protected (private) members
    - should be marked as `const`!

UCLA

Classes – Implicit and Explicit Parameters

- Consider a function "`length()`" of the `string` class

```
string str = "PIC 10A";
str.length();
```

- It doesn't get any *explicit* parameters, but it has an *implicit* parameter
  - `str1.length()`, `str2.length()`, `str3.length()` can return different values
  - The class object upon which the function called is called the *implicit* parameter

- Recall the "add" function in the **Matrix** class
  - at(i, j) is a call upon the implicit parameter
  - e.g. A.add(B) → it's A.at(i,j)

```
Matrix Matrix::add(const Matrix& B) const {
        if (isEqualDim(B)) {
                Matrix C(num_rows, num_cols);
                for (int i = 0; i < num_rows; ++i) {
                        for (int j = 0; j < num_cols; ++j) {
                                C.at(i, j) = at(i, j) + B.at(i, j);
                        }
                }
        return C;
        }
        else {
                cout << "Error: Dimensions must match!" << endl;
                return Matrix{};
        }
}
```

UCLA

Classes – Constructor Initializer List

- Always use the ***constructor initializer list*** for initialization

```
class B {
public:
    string name;
    double salary;
    int age;
    void b() const;
    B();
};
```

```
B(string _name, double _salary, int _age)
     : name(_name), salary(_salary), age(_age) {}
```

```
int main() {
    B b_object;
    B John("John Doe", 60000, 25);
}
```

The constructor's **body** is empty in this case

UCLA

Classes – Constructor Initializer List

- Default values can be set using the initializer list

```
class B {
public:
    string name;
    double salary;
    int age;
    void b() const;
    B();
};
```

```
B() : name("default_name"), salary(0.0), age(21) {}
```

The constructor's **body** is empty in this case

```
int main() {
    B b_object;
    B John("John Doe", 60000, 25);
}
```

UCLA

Classes – Constructor Initializer List

- Or, you can further do this:

```
class B {
public:
    string name;
    double salary;
    int age;
    void b() const;
    B(string _name = "default_name", double salary = 0.0, int age = 21);
};
```

with

```
B(string _name, double _salary, int _age)
      : name(_name), salary(_salary), age(_age) {}
```

This can replace the default constructor and works for both

```
int main() {
    B b_object;
    B John("John Doe", 60000, 25);
}
```

UCLA

Function Overloading

- Writing functions with the same name, but with different parameters is called "function overloading"

- Here the length function is overloaded on `string`, `vector<int>`, and `vector<double>`

```
// Function Overloading
int length(const string& str) {
        return str.length();
}

int length(const vector<int> v) {
        return v.size();
}

int length(const vector<double> v) {
        return v.size();
}
// ***********************************************
int main() {
        vector<int> vint{ 1, 2, 3, 10, 20, 30 };
        vector<double> vdouble{ -1.0, 0.0, 1.0 };
        string str("PIC 10A");

        cout << "length of vint = " << length(vint) << endl;
        cout << "length of vdouble = " << length(vdouble) << endl;
        cout << "length of str = " << length(str) << endl;
        return 0;
}
```

Function Overloading

- Operators are also functions, the only difference is that they *can* be used with special forms (i.e. with symbols) instead of "function_name(parameters)"
- e.g.

```
string a = "a", b = "bb";
cout << "a+b = " << a + b << ", or " << operator+(a, b) << endl;
```

- Thus we can also overload the operators on the user-defined classes
  - Operators +, – (binary), and – (unary)
  - Operator<<
  - See the **Matrix** class example!

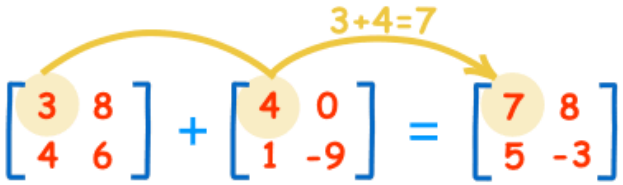
UCLA

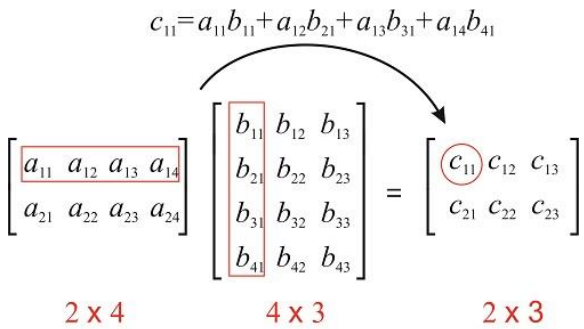
“Matrix” Class with Operator Overloading

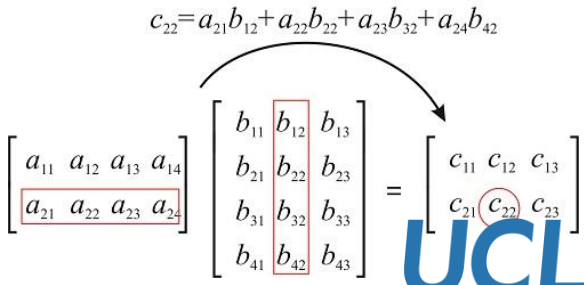
- We overload the operators + and – (binary), and operator – (unary) and operator<< for the output
- Compare with the previous version:

```
// Printing
cout << "A: " << A << endl;
// A.print();

// Arithmetic Operations (and printing)
cout << "A + C: " << A+C << endl;
// (A.add(C)).print();

cout << "A * B: " << A*B << endl;
// (A.multiply(B)).print();

cout << "1/4 * A * B: " << A*0.25*B << endl;
// (A.scalarMultiplication(0.25).multiply(B)).print();
```

$$\begin{array}{c|c} & \begin{array}{cccc} 1 & 2 & \cdots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \end{array}$$







HW6 Questions?

- Recall how we dealt with the 2-D vectors in the **Matrix** class
  - How did the double for loops work?
  - How did we access the elements (entries of the matrix)?

- If you understand how it works, HW6 becomes easy!

UCLA

Your Feedback is welcome

- Don’t hesitate to give a feedback on the discussion
- Use the link on my Github repo, or the link below:
  - https://forms.gle/erZj1iSgHNrHQuXk6

My Github repo on the web looks like:

code	Week2 Tu
LICENSE	Initial commit
README.md	Update README.md

README.md

PIC10A

PIC10A discussion 2B, UCLA for Fall 2022

Google form link for feedbacks: https://forms.gle/erZj1iSgHNrHQuXk6

Click this link

UCLA