**NAMESPACE**: organized code into logical groups & prevent same name collisions. A namespace is a scope for the entities that it encloses.  Scoping rules avoid identifier conflicts across different namespaces. // include system libraries + headers first // other libraries second // own libraries last

**ABSTRACTION**: display only essential info & hiding details. Generalize + reduction. Creation of objects & classes.

**ENCAPSULATION**: hides all implemented details within itself. Will not see data or logic that class object store within itself

**POLYMORPHISM**: msg can be displayed in many forms/diff behaviour in diff instances depending on type of data

**Operator overloading**: make operator behave differently according to situations

**Function overloading**: make single function name to perform diff tasks

**INHERITANCE**: one class taking properties/characteristics from another class. Inheritance relates classes that share the same structure and behaviour.

**MODULES:** highly cohesive & couples loosely to other modules. Store source code in header (.h) implantation (.cpp)

**COMPILATION PROCESS**: **preprocessor** (interprets all directives while creating a single unit )

**Compiler**( compiles each translation unit separately & creates a binary version of it)

**linker**(determines amount of static memory used by application/ assemble all binary units & create one complete executable)

**Guard** **is** added to header to prevent multiple inclusions of the header into a single compilation unit

Global – whole program // File – code within file

Function – length of function

Class- function of class

Block – code block

**DECLARATION**: associates identifier with type// tells compiler how to interpret entity’s identifier.

**DEFINITION:** gives meaning to identifier/executable/declaration.

**FUNCTION SIGNATURE**: identifier name, parameter type, order of parameter.

**REFERENCES**: must be initialized when created. Alias to variable/object. Don’t have memory/store address/not null. **Single precision: float f = 1.1 // float& a = f**

**MEMBER FUNCTIONS:** functions with association to an object//allows access & changes to object’s data.

1. **QUERIES**: report state of object. Adding const at end to make it query function to avoid changes. (Get function)
2. **MODIFIERS:** changes state of objects (set functions)
3. **SPECIAL**: functions to create, assign, destroy objects

**PUBLIC:** members accessible by anyone/client code. Struct default is public.

**PRIVATE:** only accessible by members of objects/within same class scope. Class default is private.

**EMPTY STATE**: default so that when receive invalid input, it will go to default response. Controls behaviour so that everytime we have same result.

**Empty state int = 0 // char = ‘\0’ pointer = nullptr**

Class Luxi{ **1) Default Constructor** // Luxi() 2) **Copy Constructor** // Luxi(const Luxi&)

3) **Copy assignment operator** //Luxi & operator =(const Luxi&) 4) **Deconstructor** // ~Luxi()

These are 4 members functions. Compiler will automatically compile if we don’t assign our own.

**CONSTRUCTOR**: called during creation of object. Have same name as class. Don’t have return type.

**COPY CONSTRUCTOR**: object of class returned by value. Passed to function by value as argument.

Any member function can access private members of its class.

**DECONSTRUCTORS**: objects go out of scope. Program ends. Delete operator called. Only one in a class. No return type, parameter/arguments. Avoid memory leak by using delete.

**CURRENT OBJECT**: “this” points to address of where object is stored. \*this will give actual object.

**Modular Programming**: organize all code for relative tasks& objects into separate files/header files under the name of those tasks/objects.

**C++ can be fully object oriented or not.**

**Float\* pNumbers = nullptr; pNumbers = new float[16];**

**//declaring it to single precision floating pt and initializing it to dynamically allocate array of 16**

**MEMBER OPERATORS**: internal inside class. Can modify state of left operand(Current object)

**HELPER OPERATOR**: external outside class(global). Don’t change operands.

1. Unary(one): +- 2. Binary(two): +-/x relational(== >< !=) logical(&& ||) 3. Ternary(Three) conditional

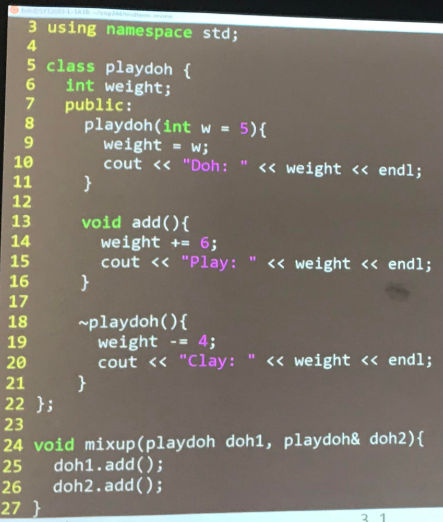
Operators that cannot be overloaded: scope :: member selection . pointer .\* condition ?:

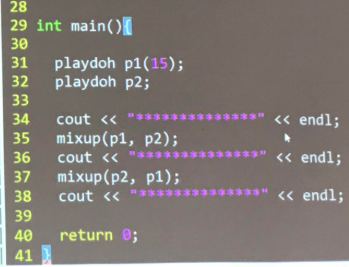
**Operator Overloading**: provide operator with special meaning for a data type. 2+2 // 2 is operand. + is operator

**HELPER FUNCTIONS**: assistive functions that support can existing class. Perform part of computation of another function. Exist outside of class. Global. Make code easier to read cuz can give them descriptions

**FREE HELPER:** global functions existing outside of class but has parameter of class type. No private access.

“friend” keyword allows private access // **auto:** determine type of variable x through value given from the right. Ex auto x=3 // useful cuz type determined by compiler, we don’t need to explicitly specify





**Playdoh doh1 is COPY. Playdoh& doh2 & is ORIGINAL.**

**Need to deconstruct copy each time leaves scope.**

**Ans: Doh: 15 Doh:5 \*\* Play: 21 Play:11 Clay:17 \*\***

**Play:17 Play: 21 Clay:13**

**\*\* Clay: 7 Clay: 17**

#include “luxi.h”

Using namespace std;

Namespace midterm{

*// class has default constructor sets 2 attributes to null (empty state)*

**Luxi::Luxi() {**

**One = nullptr;**

**Two = nullptr;**

**}**

// initialize attributes to the value of input parameter Dynamically allocate memory

**Luxi::Luxi(double first, double second) {**

**One = new double;** // one still a pointer

**Two = new double;** // two still a pointer

\***one = first;** //value of 1 assign to double first

**\*two = second;** //value of 2 assign to double second

**}**// deconstructor

**Luxi::~Luxi() {**

**Delete one;**

**Delete two;**

**}**

// query function that prints value onto screen

**Void Luxi::query() const{**

**Cout << one << two <<endl;**

**}**

// Modifier function

**Void Luxi::modifier(double first, double second){**

**If (one != nullptr || two != nullptr) {**

**One = new double;**

**Two = new double;**

**}**

**\*one = first;**

**\*two = second;**

**}**

//// conversion to double operator that returns final. If empty, return 0

**Luxi::operator Luxi() {**

**If (one == nullptr || two == nullptr){**

**Return 0;**

**}**

**Return (\*one + \*two);**

**}**

**}**

#include <iostream>

#ifndef LUXI\_H

#define LUXI\_H

Using namespace std;

Namespace midterm {

// *create class with 2 attributes of pointer double*

**Class Luxi {**

**Double\* one;**

**Double\* two;**

**Public:**

**Luxi();** // constructor

**Luxi(double, double);** //constructor w/2 parameter

**~Luxi();** //deconstructor

//class has a query that prints one and two on screen

**Void query() const;**

//class has a modifier that receives 2 attributes .

**Void modifier(double, double);**

// conversion to double operator that returns final. If empty, return 0

**Operator double();**

**};**

**}**

#endif

**#include <cstring> in order to use strcpy**

**// when setting string to empty state:**

**1) Use strcpy(variablename, ““) OR**

**2) Variablename[0] = ‘\0’**

**Example:**

Luxi = new char[strlen(pikachu) + 1];

Strcpy(luxi, pikachu)

// we are putting pikachu into luxi

**Static Memory:** amount of memory allocated for a program at load time// determined at compile time

**Dynamic Memory:** demand more memory from system during run time

**Dynamic Allocation**: variablename= new type

Ex: width = new double; 🡨 fundamental type

Ex: char\* name = new char[size]; 🡨 array // whatever char\* type is = new char. Ex)Luxi\* name = new Luxi[5];

**Deallocate:** delete[] name; // delete double;

Delete[]x 🡪 deletes array of data

Delete x 🡪 deletes a single char

**Const int SOMETHING\_MAX\_LEN = 30;** // goes after namespace sdds and before class name

Q: When does a variable shadow another variable

A: identifier declared with an inner scope shadows an identifier declared with a broader scope.

Ex: variable declared before function vs variable declared in for loop.

Q: compound type: struct & class. Fundamental type: int long char float double bool