Prevent multiple header file inclusion

/// At the top of a header file, this is used to make sure that this is only copied once, use macros
#pragma once ;
/// Is often a problem when a bunch of classes are used with shared parents etc

/// Alternate way
#ifndef TESTCLASS_H_
#define TESTCLASS_H_

#endif
/// If it is not defined it will use it. Else, won't redo

Inheritance

Two types of inheritance

- Implementation inheritance, where the actual code is transferred to the child class
- Interface inheritance where the method names are inherited

Public from parent to (Public, protected, private)
Can't pass on parent's private info
Protected: can be inherited by the next generation but is not accessible outside of the class (acts as private)
No inherited class can have more lax restrictions than the parent

- Public → protected or private
- Protect → protected or private
- Private → never inherited

Interface

Has only public pure virtual ("=0") methods and static methods
It may not have non-static data members
It need not have any constructors defined. If a constructor is provided it must take no arguments and it must be protected
If it is a subclass it may only be derived from classes that satisfy these conditions and are tagged with the Interface suffix

Constructors

Called in order of base then derived class
Derived class constructor must call the base class constructor

- Must initialize base classes members

If the derived constructor is omitted the default constructor calls the base class default constructor

Destructors operate in the reverse order of constructor. Kill subclass → Kill superclass

Data is created/initialized first, then constructor of object is run

Classes

Is a → inheritance (Generalization)

General → more specific

Tight coupling
Subclass points with arrow to base class
Denoted by a clear arrowhead

Has a → field within the class, will define an instance of that class

Composition and aggregation are types of association

Uses a (as a data member)→ aggregation

Some field is also a class, but the object can theoretically exist independently of the total object. Whole part relationship. Denoted by a clear diamond

If not independant → composition

Generalization → inheritance, extends another class

Leads to code with dependencies which is hard to change

Using → Dependency

A class is used as a parameter in a method
Low coupling, non bidirectional dependence
Denoted by a dotted line

Class implements one or more interfaces → Realization

Data is private by default
If you inherit, there is a possibility of functions with the same name
This is done by:

```
Inherited.class1.function();
```

OR

```
Inherited.class2.function();
```

Polymorphism (many forms)

Override the functions of the base class

Friend class: has access to the private members of another class
Pointers can be used to type cast

```
pi=3.14;
int& ir=pi;
*ir=3;
```

Pointers of derived classes can be used to call functions in the base class
Base class pointers cannot be used to call derived class functions which have added functionality not within the original class
Base class pointer can be used to call a class which is present in both classes
In this case: keyword virtual
No virtual = pointer class type function
virtual = the pointed to class function will be used

Pure virtual function

Public:

virtual void power-up()=0;

Using keyword this will make an object refer to itself

Abstract Base Classes

Ex. Electronics

This is never actually used to make an object, but is necessary because many classes of products may have fields that are common requiring reused inheritance across the board

Contains pure virtual functions

A derived class must override all virtual functions to not be an ABC

Templates

Can be used for classes or primitive types

Classes

Template <class T>

Class X {~~~ T, ~~};

Instantiate:

Classname <type> objname;

Many libraries are templated

Const function cannot be overwritten (an explicit way to label this)

If you have a templated class you need the .cc file to reference the implementation

Can be included at the end of the .h file

Stack

A stack is a data storage option where things are stored vertically

Use things like top and pop to check the order

Try catch blocks

Used in error handling

Essentially try a piece of code (has a higher propensity to fail)

Throw: creates an object which contains the error information if the code fails

Catch: you have multiple catch options containing error descriptions

Code is attempted to run, if it fails it snags a catch and runs the code in the catch section, then exits the try block

Static

Static type varname=value;

Static means that the variable is common throughout each class member. Static belongs to the class not the object in the class

Static in a function will store the variable in the heap, (it will be treated as global, but only available within its scope)

Static keyword in a class

The variable will be accessible by any class member (global within the class)

Ex. in constructor increment the number of of objects static int

Static functions in a class

No this pointer

Can only use static variables in the class

Const

For variables const means that the variable value will not change through the project

Declare a const class

Const class objectname

Create a constant function (constant object can only use constant functions)

Returntype scope::name() const{

}

OOD

S-O-L-I-D

- Single responsibility
  - A class should only have one reason to change, meaning that a class should have only one job.
    - Ex. calculation is one class, outputting to screen is another class to allow for other desired actions, such as outputting to a file type
- Open-closed
  - Objects should be open for extension but closed for modification
    - An interface can be used to make sure that objects are of a specific type
- Liskov substitution
  - Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T
    - Every subclass/derived class should be substitutable for their base/parent class
- Interface segregation
  - A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods that they do not use
- Dependency Inversion
  - Entities must depend on abstractions, not concretions. It states that the high level module must not depend on the low level module , but they should depend on abstractions.

Arrows point in the opposite direction of what you would expect

Inlining a function

Inline return_type Function_name ()

{

...

}

// In main
Function_name(); // Call normally
Will replace in the code every instance of a function call with the lines of code labeled in-line
This can save processing time because it won't create a spot on the stack and allocate registers
Inline is simply a compiler instruction, can be optimized out

Namespaces
Subdivide the global scope into distinct spaces, preventing name collisions in global scope
If you create a variable/function in two .cpp files, they will conflict

Instead, declare a namespace → call function with respect to the specific namespace

Using namespace std; // using the standard library
In another c++ file:
namespace namespace_name{
…
…
}
// In main
namespace :: function_name();
With using namespace, you can use all of the properties of the namespace, above syntax is used for a specific function call

Pointers
Variables without any malloc or memory commands have automatic scope
Within a set of brackets they will remain unaltered and will be destroyed/deallocated at the end of the block
This is invalid

```
int* pointerToNested;
        {
                int nestedNumber=42;
                pointerToNested=&nestedNumber;
        }
cout<<*pointerToNested<<endl;
```

// Invalid because the variable is deallocated within the brackets, and therefore an invalid memory location is referenced outside of the block (this still may work but is dangerous)
Dynamic Allocation
Memory wont be destroyed at the end of the block
New command will return a pointer of the same type as the object
Must be deallocated manually
In array use delete[]
After deallocation, the pointer will point to a freed and no longer valid memory location
It is good practice to redirect the used pointer and assign it to the value nullptr
This can then be used to check to see if the free was executed

If you don't deallocate there will be memory leaks, pieces of memory which are never deallocated

Smart pointer

Unique pointer (unique_ptr<type>)is a class which contains a variable which is the pointer to the variable (Must #include <memory>)

- When the pointer goes out of scope there are automatically no memory issues
- Easily reassignable
- Can be put in a container
- When you need to represent ownership, use a smart pointer
- When you just need to refer to an object, use raw pointers
- Auto keyword can be used to automatically determine the type of pointer being returned
- Has same performance compared to raw pointers
- There must be only one pointer to each memory address (each must be unique)

Move semantics

- Command "move()" (ex. string target = move(source);
  - Now the source is empty/corrupted, and the contents are in target

Use move semantics for smart pointers

- newpointer=move(oldpointer)

Shared pointer

- Higher overhead
- Just allows for multiple pointers to point to the same object
- The owned object is destroyed only when ever shared pointer pointing to the object is destroyed

Weak pointers are used in cyclic dependencies

- If you have two things pointing to the same three objects, and want to delete the objects when there is nothing pointing to them, use shared pointers. If these also have to point to each other, and the pointing to each other should not affect the lifetime of the objects, use weak pointers for the cyclic dependencies, resulting in the desired functionality
- Weak pointers are not just raw pointers, because they can know whether or not the object that they are pointing to has been deleted yet
  - Also weak pointers cannot be used to dereference the object, only tell if it is there
    - Weak pointers can be used to create a shared pointer pointing to the same object

Only use smart pointers for things that can be allocated with new, and deleted with delete

Use only smart pointers, or only raw pointers in a program

Smart pointers have a get() function to display the data inside

Avoid using explicit new keyword, instead opt for make_shared() or make_unique() commands

auto

A variable's type can be given by auto, which will automatically assign the type for you

- Ex. vector<string> v;
- auto s1=v[0]

Braced initializers

vector<string> v{“foo”, “bar”};

Lambda

Shorthand way of making a function easier and faster

auto isOdd=[](int x){return x%2;}

Creates an isOdd function

[]-pass variables which will be used in the function, creates its own scope

&=by reference

“=”=by value

()-pass parameters of the function

{}-function itself

Creates a function object

Enumerator

Users can define their own data types

```
enum Color
{
	COLOR_BLACK;
	COLOR_WHITE;
};
Color paint = COLOR_WHITE;
```

Reading In files

If you have a file being read in a .cpp file, the input file may be referenced by name within quotations and located in the directory of the executable or referenced in quotations by global path

Enum keyword

Used to create categorical variables

Enum color = {red, green, yellow};

Color mycolor = red;

Recursion

Always have a base case!

When will the recursion stop??

You can use data structures passed to functions to store things, then if the wrong path is taken, back tracing through the additions will remove them from the stack

The function picks up where it left off when it returns back to it

Recursion is used in trees

Recursion is very expensive, but it is very expensive

Stack

Heap

Trees
Contain a pointer to the left node, pointer to the right, and to the parent