EE402 Assignment 1

Ciarán O’Donnell

[ciaran.odonnell44@mail.dcu.ie](mailto:ciaran.odonnell44@mail.dcu.ie)

Student ID: 15414048

ECE4

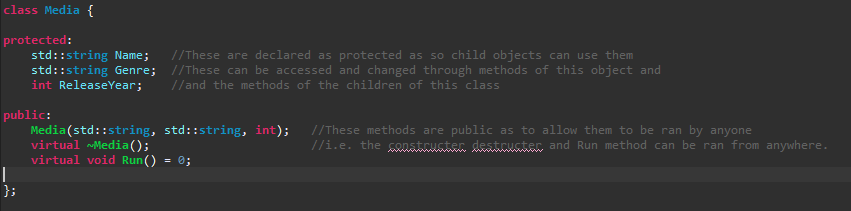
# Introduction

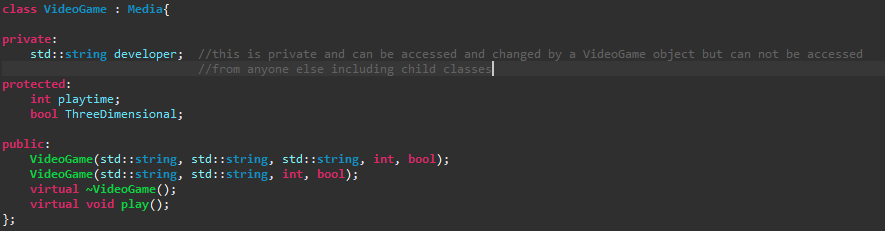
For the purpose of this assignment I have decided to build a media library program for storing movies, video games, and tv shows. There will be a class for each type of media, with attributes like runtime and release year for each. Each class will be a child of a general Media class. There is also a videogame movie class that is a child of both videogame and movie.

# Implementation

## Correct use of all access specifiers

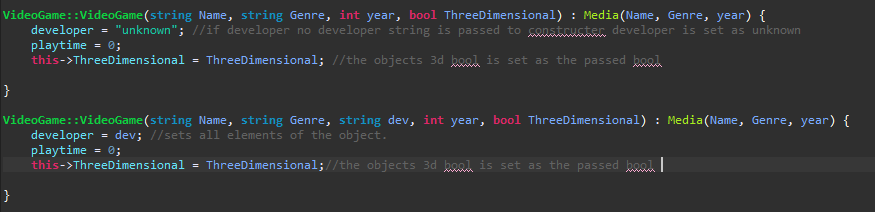
Here we can see that both public and protected are used in the Media class, the protected elements can be accessed and changed by the class and a child of this class. In the videogame class we can see that developer is private, this means that this data can be accessed and changed by any object of the videogame class but cannot be accessed or changed by anyone other than this class, not even a child class.





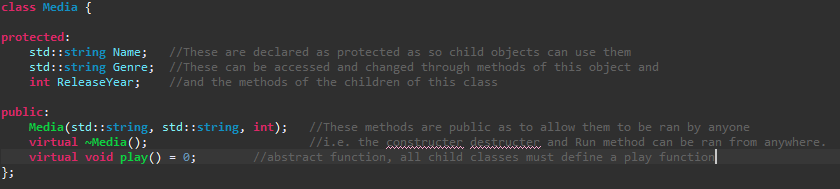
## Over-loading using either methods or constructors

To display a use of overloading I wrote two constructors for the VideoGame class. One had all the elements of the class defined, the second does similar however the developer string is not passed to the method and therefore the develepor elemtent is instead defined as the string “unknown”.



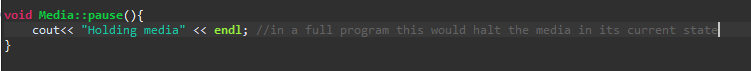
## Abstract classes with a method that is required in all derived classes

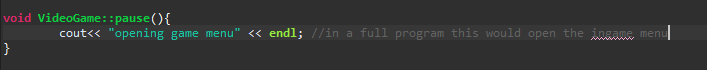
In the media class the function play is declared as abstract, this is so all classes will have to have a play function. It will be different for each child class and must be defined within them.



## Over-riding of a method.

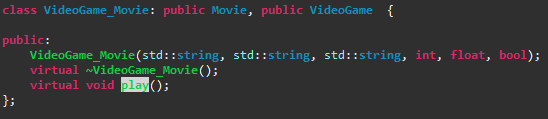
To display a use of over-riding of a method I added a pause function in the media class, this would simply hold the media when inherited, however in the videogame class I overrode this function to instead open a game menu.

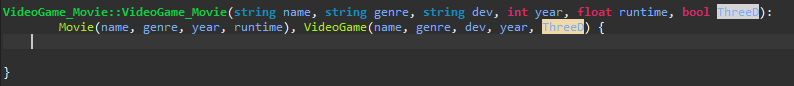




## An example of Multiple Inheritance

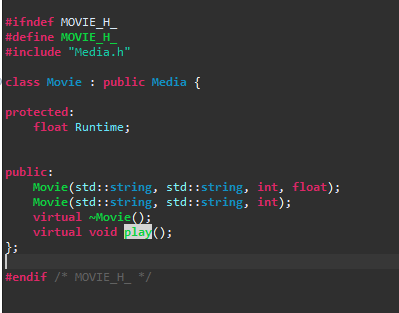
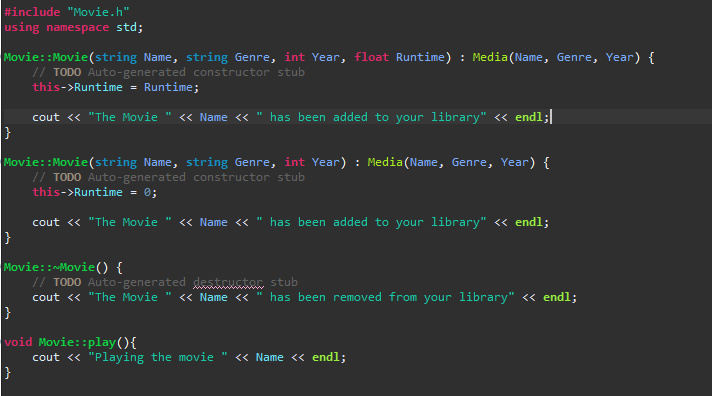
To show an example of multiple inheritance I made a class for video game movies, this inherited from both the Videogame and Movie classes. This allows it to use the methods of both types as well as store data applicable to both.





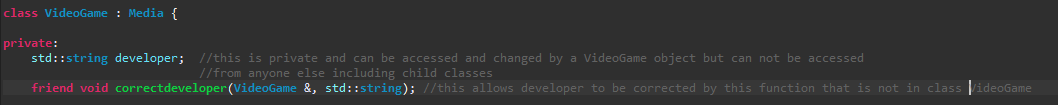
## Separate compilation

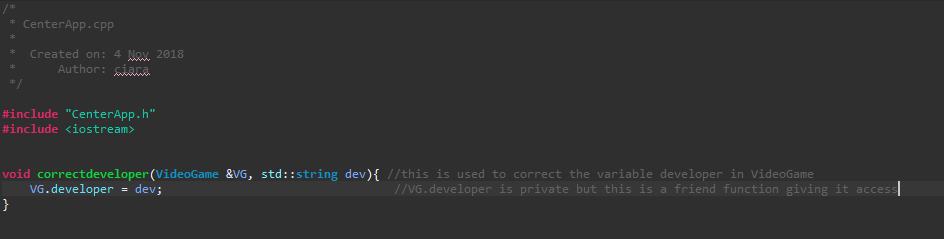
All of my classes were compiled using separate compilation. All declarations being made in the H file whilst all definitions were made using a .cpp source file. I was careful not to declare the use of any namespaces in H files as to make the files usable with others code. Here I show this within the movie class, but the rest of the separate compilation can be seen in my final source code below.

 H file cpp file

## Example use of a friend function

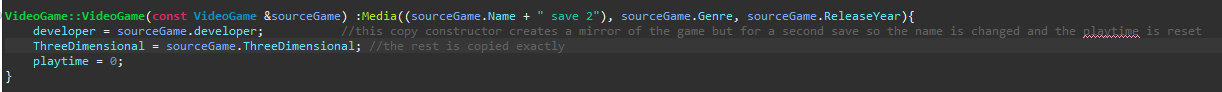
To show use of a friend function I decided to create a function to correct the developer variable in the VideoGame class from a Media Centre App class (this app class will be used to also house my main function). This friend function allows me to edit the developer variable from the centre\_app class function regardless of the fact that it is a private variable.

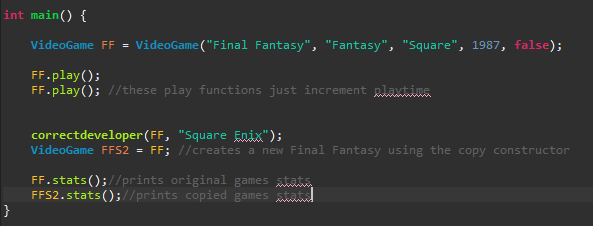
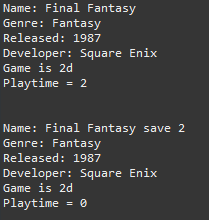




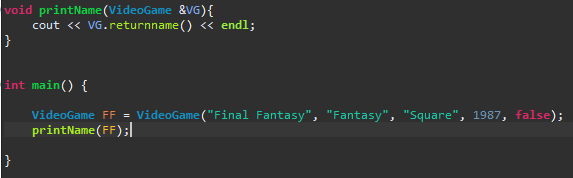
## A class with a modified copy constructor

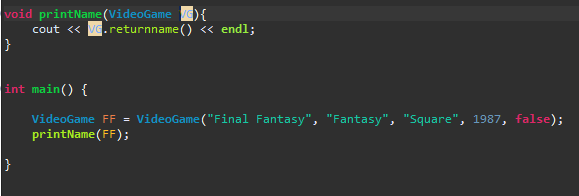
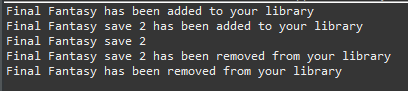
To show the use of a modified copy constructor I will change how video games are copied in my program. I will change it so that when a game is copied the new version of the game will be a second save for that game, I will name it as such and reset the playtime to 0 as this save will have no progress.



 output: 

To test the effect of the copy constructor on functions using pass by value and pass by reference I made a simple to print the name of the game passed to it, first I passed it by reference and then by value:

 output: 

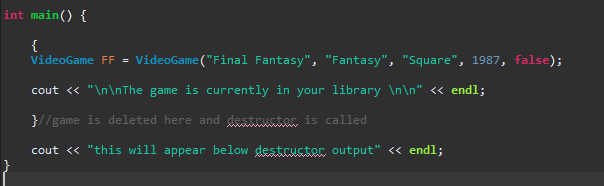
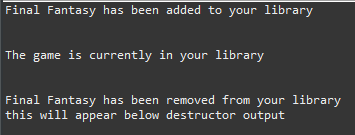
 Output: 

You can see here a new videogame called final fantasy save 2 is created by the copy constructer, its name displayed and then destroyed once the function is complete.

## A working destructor with basic functionality

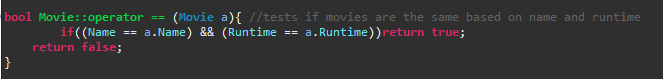
To demonstrate a possible use of a destructor I made the destructor in the Media class simply tell the user whenever a piece of media is removed, this will output for all child classes when they are deleted or go out of spec i.e. program ending or function call containing them ending ect.

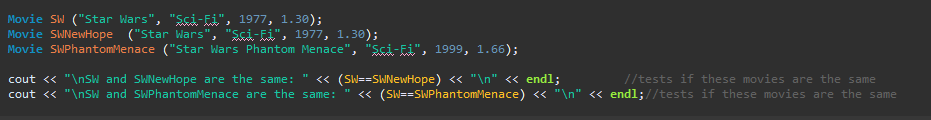
 output below:

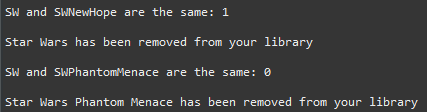
 

## Correct use of three over-loaded operators

For overloading of operators, I first used the == operator with movies to test if they were the same movie based on name and runtime:

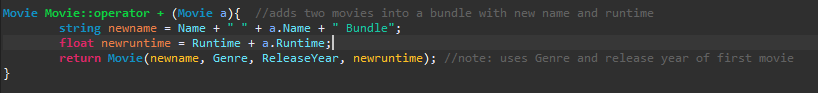




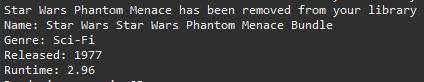


The destructor code for the second film is also displayed here as they were passed by value to the overloaded operator function and therefore a copy was created within that function and then destroyed in the same, the original for each was not destroyed.

I then overloaded the + operators in Movies to add two movies together into a Bundle, this returned a new Movie object with the name stating it’s a bundle of both and a runtime of both movies added together.

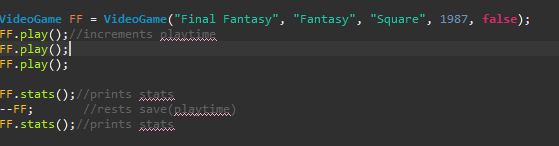


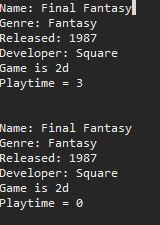


 We can again see a second Star Wars Phantom menace was created and destroyed in the operator function, and we can see the new name and runtime of the bundle.

Finally, I used the -- operator with the Videogame class to delete saved data and reset the game, in my program this meant simply resetting the games playtime to 0.

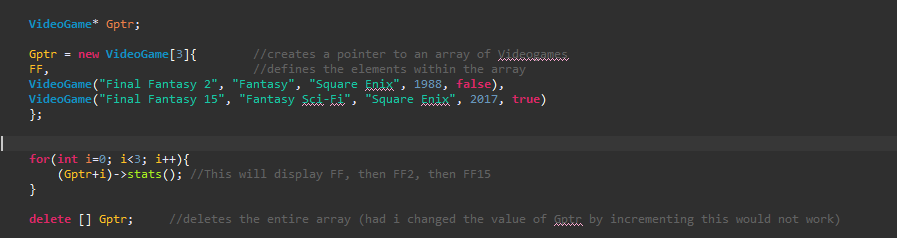


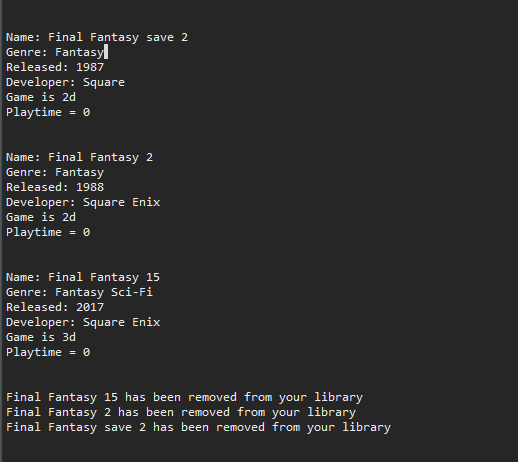




## Operations on pointers to arrays of object

To show operations on pointers to an array I created an array Gptr, pointed it at an array of three games and read the stats out of all of them and then deleted the array. Had I changed the value of my pointer deleting the array would have been impossible. This is in my CenterApp.cpp



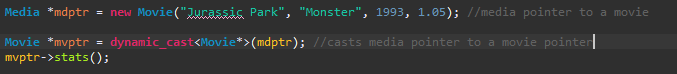


## Use all four explicit casts

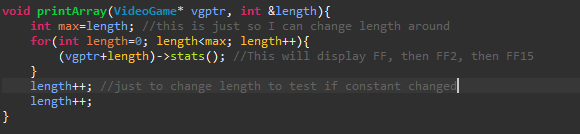
For static cast I simply casted the runtime float to an int in a return runtime as int function

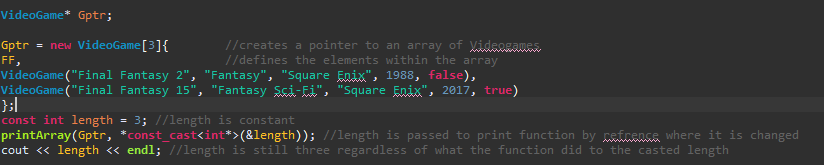


In my main I displayed the function of a dynamic cast by making a media pointer pointing to a movie object, then cast this dynamically to a movie pointer.

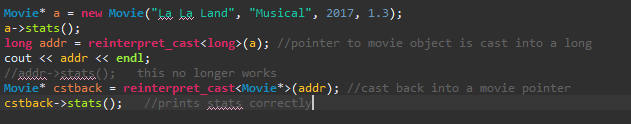


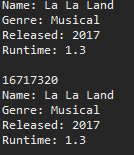
To display the use of static cast I created a function to print the stat of an array of videogames that was passed a pointer to an array and the length of the array, this function changed the value of the passed variable. I then created the array and a constant int of the length and passed both to the function, casting the constant using a constant cast.





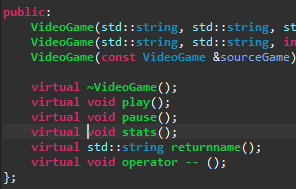
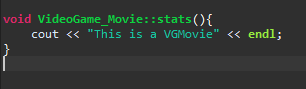
To display the use of reinterpret cast in my main I created a pointer to a movie, cast it to a long printed the long, cast it back to a movie pointer and printed the movies stats





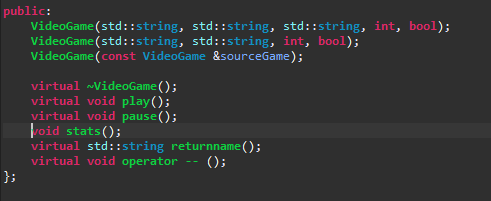
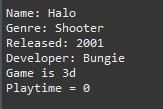
## Use of dynamic binding with virtual & non-virtual methods.

To test the effects of dynamic binding on virtual and non-virtual methods I created a Videogame pointer and pointed it at a VideoGame movie, I overrode the stats function in Videogame movie to outpout “this is a VGMovie” and tested it with both virtual and non-virtual parent methods.





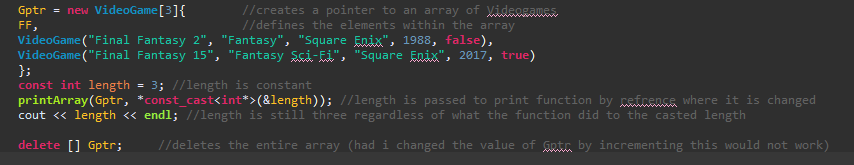
With the methods Virtual the over-ridden function ran.

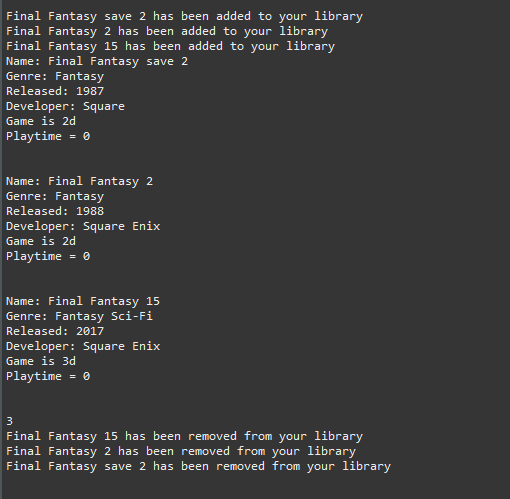


With the function made non-virtual the method of the original pointer type ran i.e. the VideoGame stats() function.

1. Correct use of new and delete for the allocation of an object/objects, with operations on the object using pointers. (Can be combined with other points)

I had already used both new and delete and operations on these objects via pointers when displaying the use of pointers to an array, we can see from the constructors and destructors how new and delete creates these objects in memory, had I changed the value of the pointer these objects could have been ‘lost’ and the memory left wasted until the application closes (a memory leak)

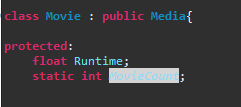
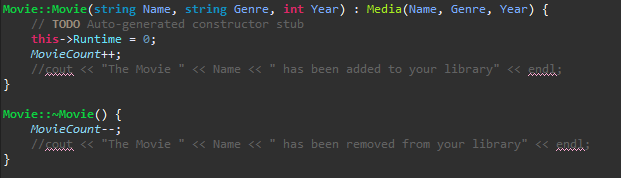


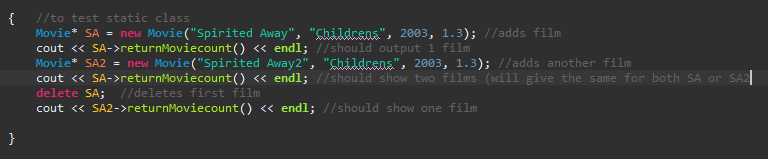


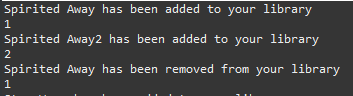
We can see the constructors and destructors of the three objects.

## Static states of a class and example usage

To show a possible use for static class I added a static int to the Movie class called MovieCount which would track how many films are in the media centre. It is incremented during the constructor of a movie and decremented in the destructor.

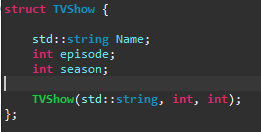
 



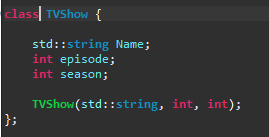
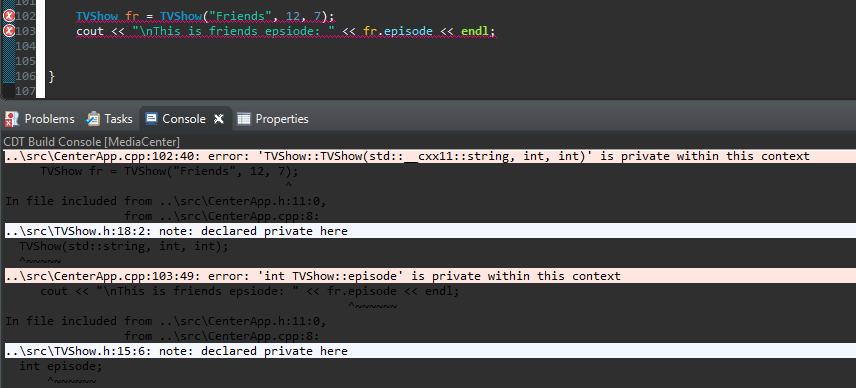


1. Demonstrates the difference between a C++ class and a C++ struct.

To demonstrate the difference between a struct and a class I created a struct called TVShow with variables Name, episode and season. I then from my main in CenterApp printed out the episode directly from the object with no errors. I then changed this to a class and this was no longer possible. The variables defaulted as public within the struct but as private within the class.



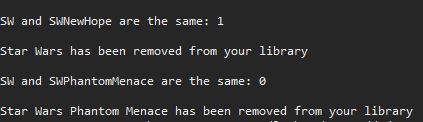
This worked perfectly.

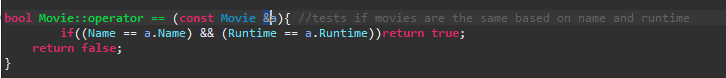
 This gave an error as the variables are now private.

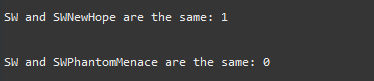
## Passing an object to a method by constant reference

To show the benefit of passing by constant refrence I returned to where I had earlier overloaded the == operator for movie. We could see when I originally did this it created a new movie everytime the operator was used for the second movie being called to the function as it was passing by value and therefore copying the object. If we however now pass by constant refrence the second movie will no longer need to be recreated and we do not need to worry about it being accidently modified as its passed as a constant. This increases efficiencty.

Originally:

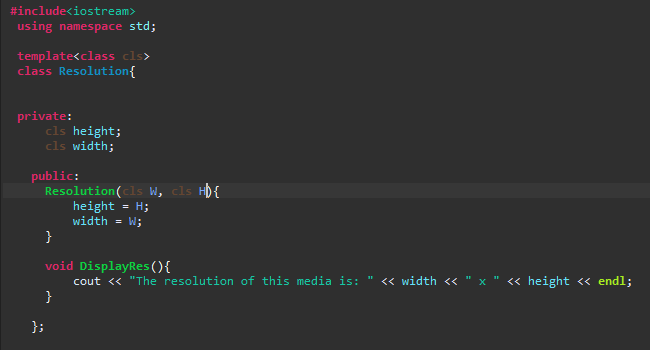
 We can see the destructor for a copy of the objects running directly after the operator function.

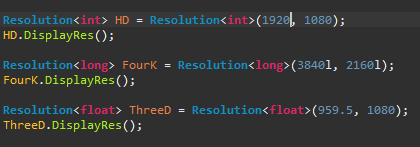


These destructors no longer appear as the objects are no longer copied and being constant they are safe from being modified.

## Writing a straightforward example class template

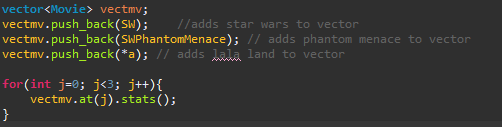
For my template class I made a class that simply contains a medias resolution, it therefore simply stores a vertical and height value for the amount of pixels, using a template this can be of any number type.

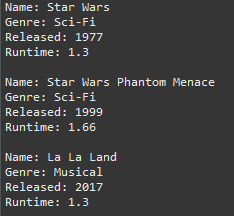


## Use the vector container to contain a number of objects of one of your classes.

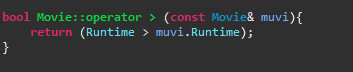
I used a vector container to stor a number of movie objects within my main in CenterApp the following is how I stored the objects within the vector.

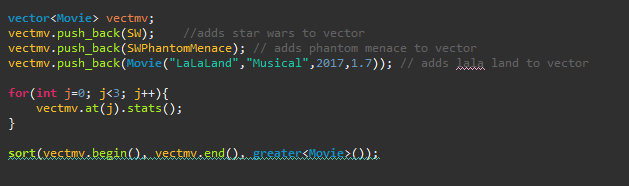




## Use of the sort algorithm on your container

Finally, I created a sort algorithm for my Movie class by overriding the > operator to order them based on their runtime so they would be sorted by shortest to longest.





# Source Code

## CenterApp.h

**#ifndef** CENTERAPP\_H\_

**#define** **CENTERAPP\_H\_**

**#include** "VideoGameMovie.h"

**#include** "TVShow.h"

**#include** "template.cpp"

**class** **CenterApp** {

**protected**:

**int** VGcount = 0;

**public**:

**void** **addGame**();

**void** **delGame**();

};

**#endif** /\* CENTERAPP\_H\_ \*/

## CenterApp.cpp

**#include** "CenterApp.h"

**#include** <iostream>

**#include** <vector>

**#include** <algorithm>

**using** **namespace** std;

**void** **correctdeveloper**(**VideoGame** &VG, std::**string** dev){ //this is used to correct the variable developer in VideoGame

VG.developer = dev; //VG.developer is //private but this is a friend function giving it access

}

**void** **printName**(**VideoGame**& VG){

cout << VG.returnname() << **endl**;

}

**void** **printArray**(**VideoGame**\* vgptr, **int** &length){

**int** max=length; //this is just so I can change length around

**for**(**int** length=0; length<max; length++){

(vgptr+length)->stats(); //This will display FF, then FF2, then FF15

}

length++; //just to change length to test if constant changed

length++;

}

**int** **main**() {

{ //to test static class

**Movie**\* SA = **new** **Movie**("Spirited Away", "Childrens", 2003, 1.3); //adds film

cout << SA->returnMoviecount() << **endl**; //should output 1 film

**Movie**\* SA2 = **new** **Movie**("Spirited Away2", "Childrens", 2003, 1.3); //adds another film

cout << SA->returnMoviecount() << **endl**; //should show two films (will give the same for both SA or SA2

**delete** SA; //deletes first film

cout << SA2->returnMoviecount() << **endl**; //should show one film

}

**Movie** SW ("Star Wars", "Sci-Fi", 1977, 1.30);

**Movie** SWNewHope ("Star Wars", "Sci-Fi", 1977, 1.30);

**Movie** SWPhantomMenace ("Star Wars Phantom Menace", "Sci-Fi", 1999, 1.66);

cout << "\nSW and SWNewHope are the same: " << (SW==SWNewHope) << "\n" << **endl**; //tests if //these movies are the same

cout << "\nSW and SWPhantomMenace are the same: " << (SW==SWPhantomMenace) << "\n" << **endl**;//tests if these //movies are the same

**Movie** SWBundle = SW + SWPhantomMenace; //adds star wars and phantom menace into a bundle

SWBundle.stats(); //prints the stats of this new film

**VideoGame** FF = **VideoGame**("Final Fantasy", "Fantasy", "Square", 1987, **false**);

FF.play();//increments playtime

FF.play();

FF.play();

FF.stats();//prints stats

--FF; //rests save(playtime)

FF.stats();//prints stats

cout << "\n\n\n\n\n\n" << **endl**; //this is too split up my output without deleting the above as it was getting //very messy.

**VideoGame**\* Gptr;

Gptr = **new** **VideoGame**[3]{ //creates a pointer to an array of Videogames

FF, //defines the elements within the array

**VideoGame**("Final Fantasy 2", "Fantasy", "Square Enix", 1988, **false**),

**VideoGame**("Final Fantasy 15", "Fantasy Sci-Fi", "Square Enix", 2017, **true**)

};

**const** **int** length = 3; //length is constant

**printArray**(Gptr, \***const\_cast**<**int**\*>(&length)); //length is passed to print function by refrence where it is //changed

cout << length << **endl**; //length is still three regardless of what the function did to the casted length

**delete** [] Gptr; //deletes the entire array (had i changed the value of Gptr by incrementing this would //not work)

cout << "\n\n\n\n\n\n" << **endl**; //this is too split up my output without deleting the above as it was getting //very messy.

cout << SW.returnintruntime() << "\n" << **endl**; //prints out the runtime of SW as an int

**Media** \*mdptr = **new** **Movie**("Jurassic Park", "Monster", 1993, 1.05); //media pointer to a movie

**Movie** \*mvptr = **dynamic\_cast**<**Movie**\*>(mdptr); //casts media pointer to a movie pointer

mvptr->stats();

**Movie**\* a = **new** **Movie**("La La Land", "Musical", 2017, 1.3);

a->stats();

**long** addr = **reinterpret\_cast**<**long**>(a); //pointer to movie object is cast into a long

cout << addr << **endl**;

//addr->stats(); this no longer works

**Movie**\* cstback = **reinterpret\_cast**<**Movie**\*>(addr); //cast back into a movie pointer

cstback->stats(); //prints stats correctly

**VideoGame** \*H = **new** **VideoGame\_Movie**("Halo", "Shooter", "Bungie", 2001, 1.4, **true**);

H->stats();

**TVShow** fr = **TVShow**("Friends", 12, 7);

cout << "\nThis is friends epsiode: " << fr.episode << **endl**;

**Resolution**<**int**> HD = **Resolution**<**int**>(1920, 1080);

HD.DisplayRes();

**Resolution**<**long**> FourK = **Resolution**<**long**>(3840l, 2160l);

FourK.DisplayRes();

**Resolution**<**float**> ThreeD = **Resolution**<**float**>(959.5, 1080);

ThreeD.DisplayRes();

**vector**<**Movie**> vectmv;

vectmv.push\_back(SW); //adds star wars to vector

vectmv.push\_back(SWPhantomMenace); // adds phantom menace to vector

vectmv.push\_back(**Movie**("LaLaLand","Musical",2017,1.7)); // adds lala land to vector

**for**(**int** j=0; j<3; j++){

vectmv.at(j).stats();

}

//sort(vectmv.begin(), vectmv.end(), greater<Movie>());

}

## Media.H

**#ifndef** MEDIA\_H\_

**#define** **MEDIA\_H\_**

**#include** <iostream>

**#include** <string>

**class** **Media** {

**protected**:

std::**string** Name; //These are declared as protected as so child objects can use them

std::**string** Genre; //These can be accessed and changed through methods of this object //and the methods of the children of this class

**public**:

**Media**(std::**string**, std::**string**, **int**);//These methods are public as to allow them to be //ran by anyone

**virtual** **~Media**//i.e. the constructer destructer and Run method can be ran from anywhere.

**virtual** **void** **play**()=0;//abstract function, all child classes must define a play function

**virtual** **void** **pause**();

**virtual** **void** **stats**();

**virtual** **int** **returnYear**();

**int** ReleaseYear;

};

**#endif** /\* MEDIA\_H\_ \*/

## Media.cpp

**#include** "Media.h"

**using** **namespace** std;

**Media::Media**(**string** Name, **string** Genre, **int** Year) {

**this**->Name = Name;

**this**->Genre = Genre;

ReleaseYear = Year;

cout << Name << " has been added to your library" << **endl**;

}

**Media::~Media**() {

cout << Name << " has been removed from your library" << **endl**;

}

**void** **Media::pause**(){

cout<< "Holding media" << **endl**; //in a full program this would halt the media in its

//current state

}

**int** **Media::returnYear**(){

**return** ReleaseYear;

}

**void** **Media::stats**(){

cout << "Name: " << Name << **endl**;

cout << "Genre: " << Genre << **endl**;

cout << "Released: " << ReleaseYear << **endl**;

}

## Movie.h

**#ifndef** MOVIE\_H\_

**#define** **MOVIE\_H\_**

**#include** "Media.h"

**#include** <iostream>

**class** **Movie** : **public** **Media**{

**protected**:

**float** Runtime;

**static** **int** *MovieCount*;

**public**:

**Movie**(std::**string**, std::**string**, **int**, **float**);

**Movie**(std::**string**, std::**string**, **int**);

**virtual** **~Movie**();

**virtual** **void** **play**();

**virtual** **bool** **operator ==** (**const** **Movie**&);

**virtual** **Movie** **operator +** (**Movie**);

**virtual** **void** **stats**();

**virtual** **int** **returnintruntime**();

**virtual** **int** **returnMoviecount**();

**virtual** **int** **returnYear**();

**bool** **operator >** (**const** **Movie**& str);

};

**#endif** /\* MOVIE\_H\_ \*/

## Movie.cpp

**#include** "Movie.h"

**using** **namespace** std;

**int** *Movie::MovieCount* = 0;

**Movie::Movie**(**string** Name, **string** Genre, **int** Year, **float** Runtime) : Media(Name, Genre, Year) {

**this**->Runtime = Runtime;

*MovieCount*++;

//cout << "The Movie " << Name << " has been added to your library" << endl;

}

**Movie::Movie**(**string** Name, **string** Genre, **int** Year) : Media(Name, Genre, Year) {

// **TODO** Auto-generated constructor stub

**this**->Runtime = 0;

*MovieCount*++;

//cout << "The Movie " << Name << " has been added to your library" << endl;

}

**Movie::~Movie**() {

*MovieCount*--;

//cout << "The Movie " << Name << " has been removed from your library" << endl;

}

**void** **Movie::play**(){

cout << "Playing the movie " << Name << **endl**;

}

**void** **Movie::stats**(){

**Media**::stats();

cout << "Runtime: " << Runtime << "\n" << **endl**;

}

**int** **Movie::returnintruntime**(){

**return** **static\_cast**<**int**>(Runtime);

}

**int** **Movie::returnMoviecount**(){

**return** *MovieCount*;

}

**int** **Movie::returnYear**(){

**return** **Media**::returnYear();

}

**bool** **Movie::operator >** (**const** **Movie**& muvi){

**return** (Runtime > muvi.Runtime);

}

**bool** **Movie::operator ==** (**const** **Movie** &a){ //tests if movies are the same based on name and //runtime

**if**((Name == a.Name) && (Runtime == a.Runtime))**return** **true**;

**return** **false**;

}

**Movie** **Movie::operator +** (**Movie** a){ //adds two movies into a bundle with new name and runtime

**string** newname = Name + " " + a.Name + " Bundle";

**float** newruntime = Runtime + a.Runtime;

**return** **Movie**(newname, Genre, ReleaseYear, newruntime); //note: uses Genre and //release year of first movie

}

## VideoGame.h

**#ifndef** VIDEOGAME\_H\_

**#define** **VIDEOGAME\_H\_**

**#include** "Media.h"

**#include** <iostream>

**class** **VideoGame** : **Media** {

**private**:

std::**string** developer; //this is private and can be accessed and changed by a VideoGame object but can not be accessed

//from anyone else including child classes

**friend** **void** **correctdeveloper**(**VideoGame**&, std::**string**); //this allows developer to be //corrected by this function that is not in class VideoGame

**protected**:

**int** playtime;

**bool** ThreeDimensional;

**public**:

**VideoGame**(std::**string**, std::**string**, std::**string**, **int**, **bool**);

**VideoGame**(std::**string**, std::**string**, **int**, **bool**);

**VideoGame**(**const** **VideoGame** &sourceGame);

**virtual** **~VideoGame**();

**virtual** **void** **play**();

**virtual** **void** **pause**();

**virtual** **void** **stats**();

**virtual** std::**string** **returnname**();

**virtual** **void** **operator --** ();

};

**#endif** /\* VIDEOGAME\_H\_ \*/

## VideoGame.cpp

**#include** "VideoGame.h"

**using** **namespace** std;

**VideoGame::VideoGame**(**string** Name, **string** Genre, **int** year, **bool** ThreeDimensional) : Media(Name, Genre, year) {

developer = "unknown"; //if developer no developer string is passed to constructer //developer is set as unknown

playtime = 0;

**this**->ThreeDimensional = ThreeDimensional; //the objects 3d bool is set as the passed //bool

}

**VideoGame::VideoGame**(**string** Name, **string** Genre, **string** dev, **int** year, **bool** ThreeDimensional) : Media(Name, Genre, year) {

developer = dev; //sets all elements of the object.

playtime = 0;

**this**->ThreeDimensional = ThreeDimensional;//the objects 3d bool is set as the passed //bool

}

**VideoGame::VideoGame**(**const** **VideoGame** &sourceGame) :Media((sourceGame.Name + " save 2"), sourceGame.Genre, sourceGame.ReleaseYear){

developer = sourceGame.developer; //this copy constructor creates a //mirror of the game but for a second save so the name is changed and the playtime is reset

ThreeDimensional = sourceGame.ThreeDimensional; //the rest is copied exactly

playtime = 0;

}

**VideoGame::~VideoGame**() {

}

**void** **VideoGame::play**() {

**if**(ThreeDimensional) cout<<"Rendering game in 3D" << **endl**;

**else** cout << "Rendering game in 2D" << **endl**;

playtime++; //in a fully developed program this would track playtime based //on time opened and closed

}

**void** **VideoGame::pause**(){

cout<< "opening game menu" << **endl**; //in a full program this would open the ingame //menu

}

**void** **VideoGame::stats**(){

**Media**::stats();

cout << "Developer: " << developer << **endl**;

**if**(ThreeDimensional) cout << "Game is 3d" << **endl**;

**else** cout << "Game is 2d" << **endl**;

cout << "Playtime = " << playtime << "\n\n" << **endl**;

}

**string** **VideoGame::returnname**(){

**return** Name;

}

**void** **VideoGame::operator --**(){ //deletes save data resets playtime to 0

playtime = 0;

## VideoGame\_Movie.h

**#ifndef** VIDEOGAMEMOVIE\_H\_

**#define** **VIDEOGAMEMOVIE\_H\_**

**#include** "VideoGame.h"

**#include** "Movie.h"

**class** **VideoGame\_Movie**: **public** **Movie**, **public** **VideoGame** {

**public**:

**VideoGame\_Movie**(std::**string**, std::**string**, std::**string**, **int**, **float**, **bool**);

**virtual** **~VideoGame\_Movie**();

**virtual** **void** **play**();

**void** **stats**();

};

**#endif** /\* VIDEOGAMEMOVIE\_H\_ \*/

## VideoGame\_Movie.cpp

**#include** "VideoGameMovie.h"

**using** **namespace** std;

**VideoGame\_Movie::VideoGame\_Movie**(**string** name, **string** genre, **string** dev, **int** year, **float** runtime, **bool** ThreeD):

Movie(name, genre, year, runtime), VideoGame(name, genre, dev, year, ThreeD) {

}

**VideoGame\_Movie::~VideoGame\_Movie**() {

// **TODO** Auto-generated destructor stub

}

**void** **VideoGame\_Movie::play**(){

**if**(ThreeDimensional) cout<<"Playing 3D movie for " << Runtime << " hours" << **endl**;

**else** cout<<"Playing 2D movie for " << Runtime << " hours" << **endl**;

}

**void** **VideoGame\_Movie::stats**(){

cout << "This is a VGMovie" << **endl**;

}

## Template.cpp

**#include**<iostream>

**using** **namespace** std;

**template**<**class** **cls**>

**class** **Resolution**{

**private**:

**cls** height;

**cls** width;

**public**:

**Resolution**(**cls** W, **cls** H){

height = H;

width = W;

}

**void** **DisplayRes**(){

cout << "The resolution of this media is: " << width << " x " << height << **endl**;

}

## TVShow.h

**#ifndef** TVSHOW\_H\_

**#define** **TVSHOW\_H\_**

**#include** <iostream>

**struct** **TVShow** {

std::**string** Name;

**int** episode;

**int** season;

**TVShow**(std::**string**, **int**, **int**);

};

**#endif** /\* TVSHOW\_H\_ \*/

## TVShow.cpp

};

**#include** "TVShow.h"

**using** **namespace** std;

**TVShow::TVShow**(**string** name, **int** ep, **int** s) {

Name=name;

episode=ep;

season = s;

}