**YouTube tutorials 55-57 – Polymorphism + Virtual and Pure Virtual Functions**

https://www.youtube.com/watch?v=R\_PPA9eejDw&index=55&list=PLAE85DE8440AA6B83

#include "stdafx.h"

#include <iostream>

using namespace std;

class Enemy {

protected:

int attackDamage;

public:

void setAttackDamage(int a) {

attackDamage = a;

}

virtual void attack() {} /\*This way you can call attack() on baby classes

(Archer, Footman) through the base(Enemy) class

virtual void attack()=0; <--This will make an abstact class. In other words, baby classes must override (not overload) the attack() function \*/

};

class Archer: public Enemy {

public: //Never forget the access modifiers

void attack() {

cout << "Archer dealt you damage -" << attackDamage << endl;

}

};

class Footman : public Enemy {

public: //Default access modifier is "private"

void attack() {

cout << "Footman dealt you damage -" << attackDamage << endl;

}

};

int main()

{

Archer a;

Footman f;

Enemy \*enemy1 = &a;

Enemy \*enemy2 = &f;

enemy1->setAttackDamage(25);

enemy2->setAttackDamage(99);

enemy1->attack(); //The reason we're using pointers is because it makes the

enemy2->attack(); //program more efficient because of how they're used in the computer

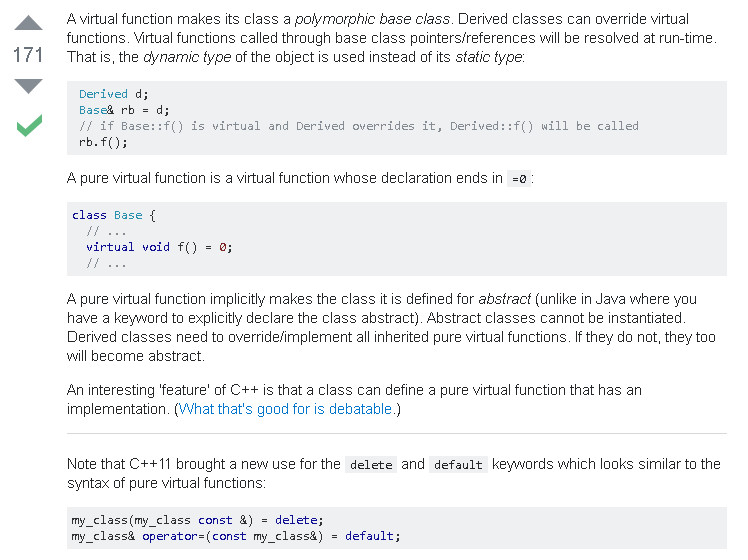
}

**Result:**



**Important notes:**

Look at next page ---V



* If you write statements inside the virtual function, that makes it default function. For example, if no such ***attack()*** function exists in footman class, the default function will be run.