1. Galaga

```
#include <iostream>
#include <Windows.h>
#include <conio.h>

using namespace std;



class Nave {
protected:
	int x, y;
	int corazones;
	int vidas;
public:
	Nave(int _x, int _y, int _corazones, int _vidas);
	void pierdeCorazon() { corazones--; }
	void dibujar();
	void borrar();
	void mover();
	void dibujarCorazones();
	void morir();
};

class NaveEnemiga : public Nave {
public:
	NaveEnemiga(int _x, int _y);
	difucultad();
};

class Disparos {
```

```
private:
	int x, y;
public:
	Disparos(int _x, int _y);
	void mover();
	bool fuera();
};
```

---

2. FPS

```
#include <iostream>
#include <vector>

using namespace std;

class PersonajePrincipal {
private:
	int salud;
	int municiones;
	int x, y, z;
	float alturaSalto;
	int velocidad;
public:
	PersonajePrincipal (int salud = 100, int municiones = 50, int x = 0, int y = 0, int z = 0 , float alturaSalto = 1.5, int velocidad = 50);
	void moverseAdelante();
	void moverseAtras();
	void moverseIzquierda();
	void moverseDerecha();
	void saltar();
	void recogerObjetos();
```

```
        int disparar() { return municiones--; };
        int correr() { return velocidad * 2; }
        int restaurarSalud() { return salud++; };
};

int main(int argc, char* argv[]) {

        return 0;
}
```

---

3. Plataformas

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Personaje {
private:
        int salud;
        bool cansado;
        vector<string> hechizos;
public:
        Personaje() { salud = 1, cansado = false;}
        void agregarHechizo(string hechizo) {hechizos.push_back(hechizo);}
        int saludActual() { return salud; }
        bool estarCansado() { return cansado; }
        void saltar();
        int recibirDanio() { return salud--; }
        int juntarEnergia() { return salud++; }
};
```

```
int main(int argc, char* argv[]) {


	return 0;
}
```

---

4. Star cough‼

```
class Unidad {
protected:
   int vida;
   int danio;
   string color1;
   string color2;
   string color3;
   string color4;
public:
   virtual void atacar() = 0;
   virtual void curar() = 0;
   virtual void setColorAtaque(string verde, string rojo) { color1 = verde, color3 = rojo; }
   virtual void setColorCuracion(string azul, string amarillo) { color2 = azul, color4 = amarillo; }
   virtual void getColorAtaque() { cout << "Unidad de color " << color1 << color3; }
   virtual void getColorCuracion() { cout << "Unidad de color " << color2 << color4; }
};


class BandoDocente : public Unidad {
public:
   void atacar() { cout << "Lanzando libros..." << endl; }
   void curar() { cout << "Curando heridos..." << endl; }
   void getColorAtaque() { cout << "Unidad de color " << color1 << endl; }
   void getColorCuracion() { cout << "Unidad de color " << color2 <<endl; }
};
```

```
class BandoAlumnos : public Unidad {
public:
   void atacar() { cout << "Lanzando libros..." << endl; }
   void curar() { cout << "Curando heridos..." << endl; }
   void getColorAtaque() { cout << "Unidad de color " << color3; }
   void getColorCuracion() { cout << "Unidad de color " << color4; }
};

int main(int argc, char* argv[]) {

        return 0;
}
```

---

5. Heroes

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

class Heroe {
protected:
   int salud;
private:
   int posInicial_x, posInicial_y;
   std::vector<std::string> princesa_rescatada;
public:
   Heroe() : salud(500), posInicial_x(0), posInicial_y(0) {}
   int _salud() const { return salud; }
   int _posInicial_x() const { return posInicial_x; }
```

```
   int _posInicial_y() const { return  posInicial_y; }

   void agregarPrincesaRescatada(const string& nombre_princesa) {
princesa_rescatada.push_back(nombre_princesa); }

   const vector<string>& _PrincesaRescatada() const { return princesa_rescatada; }


};


int main(int argc, char* argv[]) {


   Heroe heroe;


   cout << "Nuestro heroe comienza la partida con " << heroe._salud() << " puntos de salud\n";

   cout << "Se encuentra en la posicion " << heroe._posInicial_x() << " de |x| " <<
heroe._posInicial_y() << " de |y|\n";

   heroe.agregarPrincesaRescatada("Julia");

   heroe.agregarPrincesaRescatada("Natalia");

   heroe.agregarPrincesaRescatada("Mariana");

   cout << "Lista de princesas rescatadas:\n";

   for (const auto& princesa : heroe._PrincesaRescatada()) {

      cout << " - " << princesa << "\n";

   }




   return 0;

}
```

---

6. ¿No funciona?

```
//falta libreria #include <iostream> para entrada y salida de datos//

//falta libreria #include <tchar.h>

class Hechizo {

        int costo;

        void setCosto(int& val) { costo = val; }   //setCosto no debe ser pasado por referencia.//
```

```
        int getCosto() { return costo; }        //Ambos metodos setCosto y getCosto deben ser public para inicializar desde el main//
};



        Hechizo h;
        h.setCosto(10);
        std::cout << h.getCosto();

        return 0;
}
```

Ejemplo correcto:

```
#include <iostream>
#include <tchar.h>

using namespace std;

class Hechizo {
        int costo;
public:
        void setCosto(int val) { costo = val; }
        int getCosto() { return costo; }
};

int _tmain(int argc, _TCHAR* argv[]) {

        Hechizo h;
        h.setCosto(10);
        cout << h.getCosto();
```

```
        return 0;
}
```

---

7. Salidas

Dadas las siguientes definiciones de clase. Indique si es correcto, y de ser así, que mostrará el programa en cada caso:

```
class Unidad {
public:
 virtual void showClass() { cout << "Unidad Genérica"; }
};
class Soldado : public Unidad {
public:
 virtual void showClass() { cout << "Soldado"; }
};
class Granadero : public Unidad {
public:
 virtual void showClass() { cout << "Granadero"; }
};
class Ingeniero : public Unidad {
public:
 virtual void showClass() { cout << "Ingeniero"; }
};
```

a.//muestra soldado

```
Soldado *unidad1;
unidad1= new Soldado();
unidad1->showClass();
```

b.//muestra soldado

```
Unidad *unidad1;
```

```
 unidad1=new Soldado();
 unidad1->showClass();
```

c.//error, no puede convertir de ingeniero a soldado

```
Soldado *unidad1;
unidad1= new Unidad();
unidad1->showClass();
```

d.//error, no puede convertir de ingeniero a soldado

```
Soldado *unidad1;
unidad1= new Soldado();
unidad1= new Ingeniero();
unidad1->showClass();
unidad1->showClass();
```

e.//muestra soldado ingeniero

```
Soldado *unidad1;
Ingeniero* unidad2;
unidad1= new Soldado();
unidad2= new Ingeniero();
unidad1->showClass();
unidad2->showClass();
```

f.//muestra soldado

```
Soldado *unidad1;
Ingeniero* unidad2;
unidad1= new Soldado();
unidad2= new Ingeniero();
Unidad* unidad3;
unidad3=unidad1;
unidad3->showClass();
```

g.//error, no puede convertir e ingeniero a soldado

Soldado *unidad1;

Ingeniero* unidad2;

unidad1= new Soldado();

unidad2= new Ingeniero();

unidad1=unidad2;

¿Si no se hubiese utilizado “virtual” el resultado hubiese sido el mismo?

Respuesta: Las definiciones de las clases son correctas,

porque si no se hubiera caracterizado virtual en clase madre y derivadas

no se hubiera podido implementar el poliformismo en su plenitud.

---

8. Consola

```
#include <iostream>

using namespace std;

class ConsoleManager {
public:
	string generoJuego();
	string arma();
	int leerEntero();
	float leerFlotante();
	void mostrarMensajeBienvenida(const char* mensaje);
	void mostrarMensajeDespedida(const char* mensaje);
	void mostrarMensajeSalida(const char* mensaje);
	void pausa();
};
void ConsoleManager::mostrarMensajeBienvenida(const char* mensaje1) {
	cout << "Bienvenido!!" << endl;
```

```
}
string ConsoleManager::generoJuego() {
	string juego;
	cout << "Ingrese el genero de su juego: ";
	cin >> juego;
	return juego;
}
int ConsoleManager::leerEntero() {
	int numero;
	cout << "Ingrese la cantidad de jugadores maxima permitida: ";
	cin >> numero;
	return numero;
}
float ConsoleManager::leerFlotante() {
	float energia;
	cout << "Ingrese la energia maxima que tendrá su personaje: ";
	cin >> energia;
	return energia;
}
string ConsoleManager::arma() {
	string arma;
	cout << "Ingrese el arma preferida que tndrá su personaje: ";
	cin >> arma;
	return arma;
}
void ConsoleManager::mostrarMensajeDespedida(const char* mensaje2) {
	cout << "Esperamos haya disfrutado perder su tiempo, le agradecemos interactuar con nosotros" << endl;
}
void ConsoleManager::mostrarMensajeSalida(const char* mensaje3) {
	cout << "Pulsa enter para salir" << endl;
```

```
}
void ConsoleManager::pausa() {
	cin.ignore();
	cin.get();
}

int main(int argc, char* argv[]) {

	ConsoleManager manager;
	manager.mostrarMensajeBienvenida("");
	manager.generoJuego();
	  manager.leerEntero();
	  manager.leerFlotante();
	manager.arma();
	manager.mostrarMensajeDespedida("");
	manager.mostrarMensajeSalida("");
	manager.pausa();

	return 0;
}
```