índice

Manual de Usuario

Bienvenido al manual de este apasionante juego de carreras, el juego que se dispone a disfrutar ha sido diseñado para ordenadores con sistema operativo MS-DOS o bajo el sistema operativo Windows, el cual emula el sistema MS-DOS. El juego ha sido desarrollado utilizando las librerías gráficas de Borland, las cuales son suministradas junto al fichero ejecutable, también se suministran los ficheros de texto necesarios para que el programa pueda cargar la información necesaria durante el juego, a continuación pasaremos a describirle algunos aspectos que deberá tener en cuenta para poder jugar:

Este juego no requiere de instalación, por lo que el usuario solo deberá copiar los ficheros que se suministran en la carpeta deseada, y activar el fichero ejecutable. Los demás ficheros no requieren de la intervención del usuario.
Si durante la ejecución del programa principal se produce algún fallo durante la carga de ficheros, se avisará al jugador mediante un mensaje por pantalla, alertando del error.

Durante el transcurso de la partida será necesario el uso del teclado y el ratón, para seleccionar las distintas opciones y para jugar.
Nada más comenzar se mostrará una pantalla en la cual, mediante el uso del ratón deberemos elegir una de las opciones que se nos ofrecen:

- *Instrucciones de uso:* Nos mostrará la información relativa a la finalidad del juego, y nos dará las instrucciones necesarias para utilizarlo.
- *Jugar una partida:* Nos permite introducir nuestro nombre y nacionalidad por teclado, y nos muestra la pantalla de juego.
- *Búsqueda de turistas:* Mediante esta opción podremos buscar jugadores que hayan visitado la ciudad en una fecha concreta.

El juego se basa en una serie de visitas turísticas que puede realizar el jugador en la ciudad de Mérida. La partida comienza mostrando un gráfico en el que se muestran los distintos monumentos que se pueden visitar, así como las calles que unen los monumentos, en el caso de que existan calles entre ellos. El turista comenzará su visita siempre desde la oficina de turismo de la ciudad, y deberá elegir mediante el uso del ratón cual de los monumentos desea visitar, en el caso de que no exista una calle que una el lugar donde se encuentra con el monumento deseado, se mostrará un mensaje de advertencia por pantalla y se retornará al gráfico para que el usuario elija otro monumento para visitar. En todo momento se mostrará en la parte superior de la pantalla el monumento en el que se encuentra el turista actualmente.

Una vez seleccionada la ruta para llegar al monumento deseado, se cargarán la calle y el vehículo, y el visitante deberá completar el camino sorteando multitud de obstáculos, y recogiendo el mayor número de bonificaciones posibles. Durante el trayecto deberán esquivarse baches (representados por cuadrados) y accidentes (representados por rectángulos), y recogerse estatuas (representadas por circunferencias) y vasijas (representadas por elipses si cerrar), para mover el coche el jugador deberá usar las flechas de dirección del teclado.

A lo largo del viaje entre monumentos, el turista irá encontrándose con diferentes tipos de objetos, los cuales poseen diferente efectos sobre el coche:

- *Estatua (circunferencia)*: Cada vez que se coja una estatua el coche dispondrá de una rueda de repuesto en caso de pillar un bache. Todas las estatuas que se cojan se irán acumulando en una lista de bonificaciones para su posible uso en el futuro, además de proporcionar al turista distintas cantidades de puntos que le permitirán mejorar su puntuación y conseguir un bólido.
- *Vasija (elipse)*: Cada vasija que se obtenga poseerá una cantidad diferente de combustible, para poder repostar gasolina cuando el depósito baje de 20 litros, durante la carrera se guardarán en una lista de bonificaciones para usarlas cuando sea necesario. Las vasijas al igual que las estatuas proporcionan distintas cantidades de puntos cada vez que se recoge alguna.
- *Bache (cuadrado)*: Los baches son obstáculos muy peligrosos para el viajero, ya que cuando el coche coge alguno pierde una rueda del vehículo, y si no tenemos una rueda de repuesto (proporcionada por una estatua) el juego finalizará.
- *Accidente (rectángulo)*: En caso de que nos topemos con un accidente, nuestro coche dejará de responder a nuestras órdenes durante la cantidad de tiempo indicada por el mismo, dejando el coche abandonado a su suerte, chocando con todo lo que se encuentre por el camino.

El jugador comenzará usando un coche normal al iniciar el juego, el cual permite movimientos a izquierda y derecha (← →). El coche posee un depósito de gasolina de 50 litros como máximo, y comenzará con una cantidad determinada de combustible que irá disminuyendo a lo largo del recorrido, pudiéndose quedar sin gasolina y finalizando la partida. Si a lo largo del juego el turista obtiene una puntuación superior a 1000 puntos, obtendrá como coche un bólido, el cual le permitirá moverse horizontal y verticalmente (← ↕ →).

En todo momento el usuario podrá dar por finalizada una carrera, pulsando en el teclado escape (Esc), volviéndose a cargar el gráfico de monumentos.

Una vez que el turista complete el recorrido, se presentará por pantalla de nuevo el gráfico de monumentos, indicándose en la parte superior de la pantalla el monumento en el que se encuentra actualmente. En este momento se podrá elegir visitar un nuevo monumento o salir del juego.
Si el usuario selecciona un nuevo monumento, se comprobará que existe un camino entre el monumento donde se encuentra actualmente el turista y el monumento que se desea visitar, cargándose la calle en caso de que exista el camino, volviéndose a repetir la carrera tal y como se describió anteriormente.
Si por el contrario el usuario decide abandonar el juego, se guardará toda la información de su visita, así como su información personal.

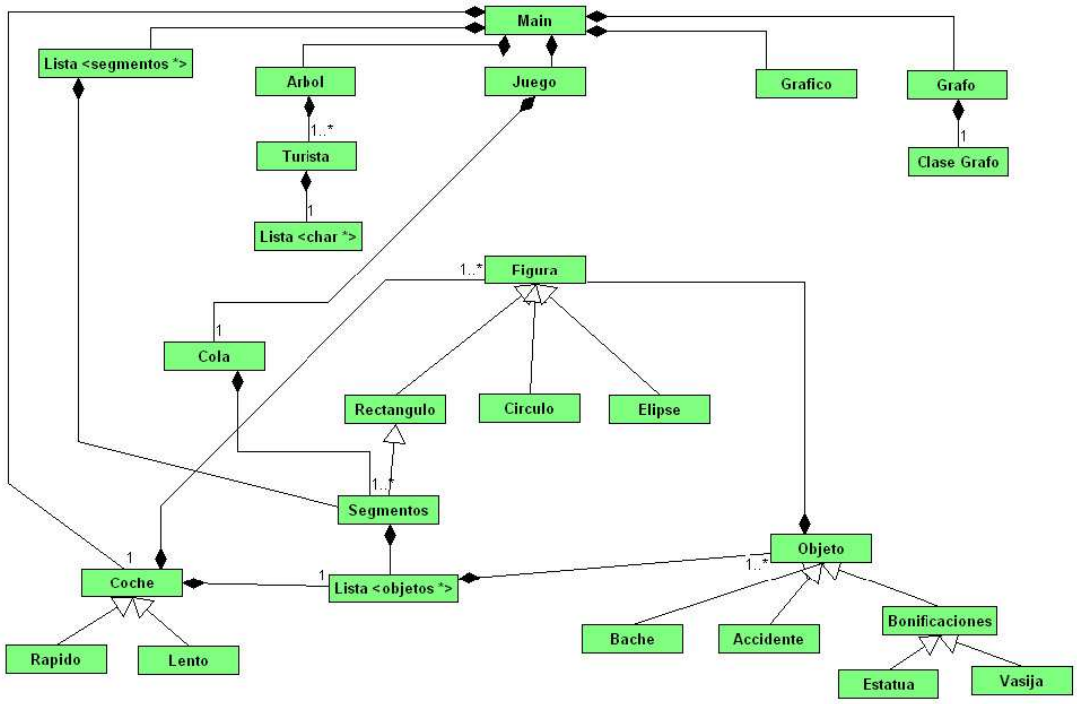
Así por ejemplo, si un jugador inicia una partida, deberá introducir sus datos de turista, y comenzará su visita desde la oficina de turismo. Si elige que quiere visitar el Acueducto de los Milagros por ejemplo, el juego le avisará que no hay ninguna calle que una esos dos puntos de la ciudad, por lo que tendrá que recorrer primero la calle que lleva al Templo de Diana, y una vez que se encuentre en el templo, podrá desplazarse hasta el acueducto, e incluso es posible que llegue montado en un bólido, si en los trayectos anteriores ha conseguido 1000 puntos o más. Cuando decida finalizar la partida se guardarán los monumentos visitados por el turista.

Los errores mas comunes que pueden producirse a la hora de ejecutar el juego, es que falte algún fichero que contenga información sobre los elementos del juego, o bien que falten las librerías gráficas de Borland, en tal caso se avisará al jugador por pantalla para que copie los ficheros necesarios junto al ejecutable.

Manual del Programador

Análisis

Diagrama:



Descripción de clases conceptuales:

Clase figura: La clase figura es clase base de rectángulo, círculo y elipse, y por tanto contiene los atributos comunes a estos tres tipos de figuras. Son atributos protegidos:

- int startx; Representa la coordenada x (en cuadrado la x de esquina superior)
- int starty; Representa la coordenada y (en cuadrado la y de esquina superior)
- int color; Representa el color del que se pinta

Posee los siguientes métodos para manipular los atributos:

- figura(int startx,int starty, int color); Constructor, inicializa la clase
- figura(const figura&original); Copia instancias de la clase
- virtual ~figura(); Destructor
- int get_x(); Devuelve el valor de startx
- void set_x(int startx); Inicializa startx
- int get_y(); Devuelve el valor de starty
- void set_y(int starty); Inicializa starty
- int get_color(); Devuelve el valor de color
- void set_color(int color); Inicializa color
- virtual void pintar()=0; Método virtual que pinta una figura
- virtual void borrar()=0; Método virtual que borra una figura

Clase círculo: La clase círculo es una subclase de figura, y por tanto los atributos que contiene son específicos de este tipo de figura. Son atributos privados:

- int radio; Representa el valor del radio de la circunferencia

Posee los siguientes métodos para manipular los atributos:

- circulo(int startx, int starty, int color, int radio); Constructor, inicializa atributos
- circulo(const circulo&original); Copia instancias de la clase
- void set_radio(int radio); Da un valor al radio
- int get_radio(); Obtiene el valor del radio
- void pintar(); Pinta una circunferencia
- void borrar(); Borra una circunferencia
- ~circulo(); Destructor de círculo

Clase elipse: La clase elipse es una subclase de figura, y por tanto los atributos que contiene son específicos de este tipo de figura. Son atributos privados:

- int startang; Representa el ángulo de inicio de la elipse
- int endang; Representa el ángulo de finalización de la elipse
- int radio_big; Representa el radio mayor de la elipse
- int radio_small; Representa el radio menor de la elipse

Posee los siguientes métodos para manipular los atributos:

- elipse(startx, starty, color, startang, endang, radio_small, radio_big); Constructor
- elipse(const elipse&original); Copia instancias de la clase
- void set_radio_big(int radio_big); Modifica el valor del radio mayor
- int get_radio_big(); Obtiene el valor del radio mayor
- void set_radio_small(int radio_small); Modifica el radio menor
- int get_radio_small(); Obtiene el valor del radio menor
- void pintar(); Pinta una elipse
- void borrar(); Borra una elipse
- ~elipse(); Destructor de la clase elipse

Clase rectángulo: La clase círculo es una subclase de figura, y por tanto los atributos que contiene son específicos de este tipo de figura. Son atributos privados:

- int endx; Representa el valor de x en la esquina inferior derecha
- int endy; Representa el valor de y en la esquina inferior derecha

Posee los siguientes métodos para manipular los atributos:

- rectangulo(int startx, int starty, int color, int endx, int endy); Constructor
- rectangulo(const rectangulo&original); Copia instancias de la clase
- void set_endx(int endx); Modifica la x inferior derecha
- void set_endy(int endy); Modifica la y inferior derecha
- int get_endx(); Obtiene la x inferior derecha
- int get_endy(); Obtiene la y inferior derecha
- void pintar(); Pinta un rectángulo
- void borrar(); Borra un rectángulo
- virtual ~rectangulo() Destructor rectángulo

Clase objeto: La clase objeto esta compuesta por agregación de la clase figura, ya que posee un atributo forma del tipo figura, además es clase base de bonificación, bache, y accidente, por lo tanto contiene los atributos comunes a estos tres tipos de objetos. Son atributos protegidos:

- figura *forma; Forma de la figura que representa al objeto
- int x; Coordenada x del centro del objeto
- int y; Coordenada y del centro del objeto
- int x1,y1,x2,y2; Coordenadas del rectángulo que engloba a el objeto

Posee los siguientes métodos para manipular los atributos:

- objeto(int x, int y); Constructor que inicializa parametros
- objeto(const objeto&original); Constructor copia de objetos
- figura * darforma(); Da la forma del objeto para modificar la figura
- void calcular(figura * fig); Calcula 2 coordenadas del rectángulo contenedor
- int get_x1(); Obtiene la x superior izquierda del rectángulo
- int get_y1(); Obtiene la y superior izquierda del rectángulo
- int get_x2(); Obtiene la x inferior derecha del rectángulo
- int get_y2(); Obtiene la y inferior derecha del rectángulo
- virtual void pintar()=0; Pinta un objeto
- virtual void borrar()=0; Borrar un objeto
- virtual ~objeto(); Destructor del objeto

Clase bache: La clase bache es una subclase de objeto, pero es una clase vacía de atributos, ya que no tiene ninguno particular, sin embargo es necesaria ya que posee una forma distinta al resto de objetos y por tanto tendrá un método pintar específico.
Posee los siguientes métodos para manipular los atributos:

- bache(int x, int y); Constructor que inicializa bache en (x,y)
- bache(const bache&original); Copia una instancia de tipo bache
- void pintar(); Pintar un bache
- void borrar(); Borrar un bache
- ~bache(); Destructor de bache

Clase accidente: La clase accidente es una subclase de objeto, y por tanto los atributos que contiene son específicos de este tipo de objeto. Son atributos privados:

- int tiempo; Representa el tiempo que permanece inmovilizado el coche

Posee los siguientes métodos para manipular los atributos:

- accidente(int x, int y, int tiempo); Constructor que inicializa en (x,y) y tiempo
- accidente(const accidente&original);Constructor copia
- int get_tiempo(); Da el tiempo de parada del accidente
- void pintar(); Pinta un accidente
- void borrar(); Borrar accidente
- ~accidente(); Destructor accidente

Clase bonificación: La clase bonificación es una subclase de objeto, y por tanto los atributos que contiene son específicos de este tipo de objeto, además es clase base de vasija y de estatua, por lo que son atributos protegidos:

- int puntos; Representa los puntos que se otorgan al coger la bonificación

Posee los siguientes métodos para manipular los atributos:

- bonificacion(int x, int y, int puntos); Constructor parametrizado
- bonificacion(const bonificacion&original); Copia una bonificación
- void set_puntos(int puntos); Modifica la puntuación de la bonificación
- int get_puntos(); Obtiene la puntuación de la bonificación
- virtual void pintar()=0; Pintar bonificación (estatua o vasija)
- virtual void borrar()=0; Borrar bonificación (estatua o vasija)
- virtual ~bonificacion(); Destructor de bonificación

Clase vasija: La clase vasija es una subclase de bonificación, la cual es a la vez subclase de objeto, por tanto los atributos que contiene vasija son específicos de este tipo de objeto. Son atributos privados:

- int litros; Representa los litros de gasolina que contiene una vasija

Posee los siguientes métodos para manipular los atributos:

- vasija(int x, int y, int puntos, int litros); Construye vasija en (x,y) con unos litros
- vasija(const vasija&original); Copia una vasija
- int get_litros(); Obtiene los litros de gasolina de la vasija
- void pintar(); Pinta una vasija
- void borrar(); Borra una vasija
- ~vasija(); Destructor vasija

Clase estatua: La clase estatua es una subclase de bonificación, la cual es a la vez subclase de objeto, por tanto los atributos que contiene estatua son específicos de este tipo de objeto. Son atributos privados:

- figura *rueda; Puntero a la figura círculo rueda extra

Posee los siguientes métodos para manipular los atributos:

- statua(int x, int y, int puntos); Constructor estatua en (x,y) con puntos
- estatua(const estatua&original); Copia una estatua
- void pintar(); Pinta estatua
- void borrar(); Borra estatua
- ~estatua(); Destructor estatua

Clase coche: La clase coche esta compuesta por agregación de la clase figura, ya que posee un vector de siete posiciones de tipo figura, además es clase base de normal y rápido, por lo tanto contiene los atributos comunes a estos dos tipos de coche. Son atributos protegidos:

- int x; Coordenada x del centro del coche
- int y; Coordenada y del centro del coche
- int x3,y3,x4,y4; Coordenadas del rectángulo que contiene la carrocería
- figura *v_figuras[7]; Vector de siete figuras 4 ruedas, 2 faros, 1 carrocería
- int vel_horiz; Velocidad horizontal
- int gasolina; Gasolina
- lista<objeto> *extras; Lista de bonificaciones recogidas (estatuas y vasijas)

Posee los siguientes métodos para manipular los atributos:

- coche(int x,int y); Constructor con parámetros de coordenadas
- lista<objeto> * darextras(); Da la lista de bonificaciones acumuladas
- figura * darcarroceria(); Obtiene la figura de la carrocería (v[6])
- void calcularc(figura * figu); Calcula rectángulo contenedor de carrocería
- void set_gasolina(int gasolina); Actualiza la gasolina
- int get_gasolina(); Obtiene la gasolina que tiene el coche
- void set_vel_horiz(int vel_horiz); Inicializa la velocidad horizontal del coche
- int get_vel_horiz(); Obtiene la velocidad horizontal
- int get_x3(); Da la x superior izquierda del rectángulo
- int get_y3(); Da la y superior izquierda del rectángulo
- int get_x4(); Da la x inferior derecha del rectángulo
- int get_y4(); Da la y inferior derecha del rectángulo
- virtual void pintar()=0; Pintar coche (normal ó rápido)
- virtual void borrar()=0; Borrar coche
- virtual void mover(int tecla)=0; Mueve el coche
- virtual ~coche(); Destructor

Clase normal: La clase normal es una subclase de coche, por lo tanto los atributos que contiene normal son específicos de este tipo de coches. Son atributos privados:

- figura *v_ruedas_extras[2]; Vector de dos figuras circulo

Posee los siguientes métodos para manipular los atributos:

- normal(int x, int y); Construye un coche normal con centro en (x,y)
- void pintar(); Pinta coche
- void borrar(); Borra coche
- void mover(int tecla); Mover coche
- void operator --(void); Elimina una rueda
- ~normal(); Destructor

Clase rápido: La clase rápido es una subclase de coche, por lo tanto los atributos que contiene rápido son específicos de este tipo de coches. Son atributos privados:

- Int vel_vert Velocidad vertical de los coches rápidos o bólidos

Posee los siguientes métodos para manipular los atributos:

- rapido(int x, int y); Constructor sin parametrizar
- void operator --(void); Elimina una rueda
- void set_vel_horiz(int v); Introduce el valor de la velocidad horiz.
- void pintar(); Pinta un coche rápido o bólido
- void borrar(); Borra coche rápido
- void mover(int tecla); Mueve el coche rápido
- ~rapido(); Destructor

Clase segmento: La clase segmento hereda de la clase rectángulo el atributo que da forma a la carretera, sus atributos son privados:

- figura *carretera; Forma de la carretera
- lista<objeto> *lista_objetos; Lista de obstáculos y bonificaciones
- objeto *p_obj_aux; Puntero que almacena los objetos de la lista
- objeto *copia_obj; Nuevo objeto que se inserta en la nueva lista

Posee los siguientes métodos para manipular los atributos:

- segmento(int xs, int ys, int col, int xi, int yi); Crea un segmento
- segmento(const segmento&original); Copia un segmento
- lista<objeto> * darlista(); Da la lista con los objetos de segmento
- figura * darcarretera(); Da el rectángulo que de forma al segmento
- ~segmento(); Destructor

Clase turista: La clase turista representa los datos de los jugadores, sus atributos son:

- struct date visita; Fecha de visita
- lista<char> *visitados; Lista de monumentos visitados
- char * nombre_turista; Nombre del turista
- char * nacionalidad; Nacionalidad del turista
- int puntos; Puntuación del turista

Tiene los siguientes métodos públicos para manipular los atributos:

- turista(char *nombre, char *nacion); Constructor de turista
- char * get_nombre(); Da el nombre del turista
- int get_puntos(); Da puntos acumulados por turista
- void set_puntos(int p); Actualiza puntos
- void pilla_fecha(); Obtiene la fecha actual
- void get_fecha(int &d,int &m,int &a); Da fecha de la última visita
- void set_fecha(int d,int m,int a); Modifica la fecha
- void imprime_fecha(); Muestra la fecha
- char * get_nacionalidad(); Da nacionalidad del turista
- void mostrar_monumentos(int nm); Da monumento posición i del vector
- char * encadenar(); Forma la cadena para el fichero
- void set_monumento(char * m); Inserta un monumento en la lista
- char * get_monumento(); Obtiene monumento de la posición
- lista<char> * darlista(); Da la lista de monumentos visitados
- ~turista(); Destructor

Clase lista: La clase lista consiste en una colección de nodos simplemente enlazados, cada uno de los cuales contiene un puntero a datos, y un puntero al siguiente nodo en la colección, sus atributos son privados y sus métodos públicos:

- ptrnodo cabeza; Puntero al nodo inicial de la lista
- ptrnodo fin; Puntero al nodo final de la lista
- ptrnodo p_actual; Puntero a la posición actual de la lista
- ptrnodo anterior; Puntero a la posición anterior a la actual
- ptrnodo aux; Puntero auxiliar a un nodo

Posee los siguientes métodos para manipular los atributos:

- void crear_lista(); Crea la lista, inicializando los punteros
- void insertar(datos_lista *m); Inserta un puntero a datos en un nodo de la lista
- bool eliminar_actual(); Elimina el nodo actual
- void eliminar(datos_lista *p); Elimina el nodo que coincide con el puntero p
- void iniciar(); Coloca p_actual al inicio de la lista
- datos_lista* leer(); Obtiene el dato que se encuentra en el nodo actual
- bool noesfin(); Devuelve “true” si p_actual no está vacio
- void avanzar(); Avanza un nodo p_actual y anterior
- bool vacia(); Devuelve “trae” si la cabeza está vacía
- void borradototal(); Elimina la lista
- ~lista(); Destructor

Clase cola: La clase cola consiste en una colección de nodos simplemente enlazados (una lista), en la que todas las inserciones se realizan por el final, y todas las extracciones por el principio. La cola no puede ser recorrida. Cada nodo contiene un puntero a datos, y un puntero al siguiente nodo de la cola, sus atributos son privados y sus métodos públicos:

- ptrnodo inicio; Puntero al comienzo de la cola (extracciones)
- ptrnodo fin; Puntero al final de la cola (inserciones)

Posee los siguientes métodos para operar con los atributos:

- cola (); Construye la cola inicializando sus punteros
- void encolar (datos_cola *d); Inserta un nodo en la cola
- datos_cola* primero (); Devuelve el dato del primero de la cola
- void desencolar (); Extrae el primero de la cola
- bool cola_vacia (); Devuelve “true” si la cola está vacía
- void borradototal (); Elimina la cola
- ~cola (); Destructor

Clase grafo: La clase grafo consta de dos estructuras: Un vector de seis posiciones de punteros a cadenas de caracteres, el cual almacena los monumentos que pueden visitar los turistas. Además de una matriz de 6x6 posiciones que contiene en cada celda una estructura de un puntero de cadena de caracteres (calle) y un entero (longitud), sus atributos son privados y sus métodos públicos:

- T_vectornodos VN; Vector [6] de punteros a cadenas (monumentos)
- T_matrizadyacencia M; Matriz 6x6 de la estuctura: cadena y entero
- int vel_horiz; Almacena la velocidad horizontal del coche
- int gasolina; Almacena la gasolina inicial del coche
- char * p_origen; Puntero al monumento actual

Tiene los siguientes métodos para manipular los atributos:

- grafo(char * fich); Crea el grafo a partir del nombre de fichero que se le pasa
- bool vacio(); Comprueba si el grafo está completamente vacio
- bool nuevo_arco(char * m_origen, char * m_destino, char * calle, int distancia);
- bool borrar_arco(char * origen, char * destino); Borra arco entre 2 monumentos
- void mostrar_nodos(); Muestra todos los monumentos que hay
- char * get_nodo(int i); Muestra el monumento de la posición i
- void mostrar_arcos(); Muestra todas las calles y sus longitudes
- void pintar_arcos(); Busca los arcos y los pinta en pantalla
- bool existe_arco (char * origen, char * destino); Comprueba que existe un arco
- int get_vel_horiz(); Obtiene la velocidad horizontal de coche
- int get_gasolina(); Obtiene la gasolina inicial
- void set_p_origen(int i); Posiciona al turista en el monumento i tras jugar
- char * get_p_origen(); Obtiene el nombre del monumento en que se está
- int posicion_origen(); Obtiene la posición del monumento en que se está
- char * get_p_vn(int i); Obtiene el nombre del monumento de posición i
- char * get_calle(int i, int j); Da la calle entre monumento i y monumento j
- ~grafo(); Destructor

Clase línea: La clase línea se utiliza para ir leyendo las líneas de texto de los ficheros, y separarlas por campos, ya que los ficheros usan de separación la almohadilla (#).
Sus atributos son privados y sus métodos públicos:

- char *param[12]; 12 campos de 30 caracteres cada uno

Tiene los siguientes métodos:

- linea(); Constructor línea
- void parametro(char *L, int n); Obtiene los parámetros
- char *obtener_campo(int n); Obtiene un campo de la línea leída
- void separar (char *L); Separa los campos

Clase árbol: La clase árbol es una estructura no lineal formada por un conjunto de nodos y un conjunto de ramas que los unen. En el árbol existe un nodo especial denominado raiz del que salen el resto de nodos. Asimismo, de un nodo pueden salir una o dos ramas denominandose nodo bifurcación. Cada nodo posee un puntero de tipo turista y dos punteros (uno izquierdo y otro derecho) de tipo nodo, sus atributos son:

- pnodo arbol; Puntero al nodo raíz
- pnodo padre; Puntero a un nodo bifurcación
- pnodo actual; Puntero al nodo actual
- pnodo nuevo; Puntero a un nodo nuevo

Posee los siguientes métodos públicos:

- arbol (); Constructor de árbol
- arbol (pnodo aux); Constructor árbol parametrizado
- bool insertar (turista *t); Inserta un turista en árbol basándose en el nombre
- bool borrar (turista *t); Borra un turista del árbol
- arbol hijoizq (); Devuelve el subárbol izquierdo
- arbol hijoder (); Devuelve el subárbol derecho
- bool vacio (pnodo aux); Comprueba si está vacío el árbol
- void inorden (pnodo aux); Ordena los turistas por nombre alfabéticamente
- void mostrar (); Muestra los turistas ordenados
- ~arbol (); Destructor árbol

Clase gráfico: La clase gráfico se utiliza para realizar operaciones en modo gráfico, sus atributos son privados y sus métodos públicos:

- int GraphDriver; Variable para almacenar driver gráfico
- int GraphMode; Variable para almacenar valor del modo
- double AspectRatio; Aspecto de ratio de un pixel sobre la pantalla
- int MaxX, MaxY; Máxima resolución de la pantalla
- int MaxColors; Máximo conjunto de colores posibles
- int ErrorCode; Variable para almacenar errores
- struct palettetype palette; Estructura para la información de la paleta
- int col_raton, fil_raton; Variable para almacenar la columna del ratón

Posee los siguientes métodos públicos:

- grafico(); Constructor gráfico
- void Inicializacion_modo_grafico(void); Inicia el modo gráfico
- int IniciarRaton(void); Inicializa el ratón
- void MostrarRaton(void); Visualiza el ratón por pantalla
- void OcultarRaton(void); Oculta el ratón
- int BotonesRaton(void); Devuelve una pulsación del ratón
- int PosXRaton(void); Devuelve posición x de la pulsación
- int PosYRaton(void); Devuelve posición y
- void tratar_raton(void); Analiza la pulsación
- int esperarpulsacion(); Analiza la zona de la pantalla
- void borramenu(); Borra el menú de pantalla
- void pintainstr(); Muestra la pantalla de instrucciones
- int esperarseleccion(); Selección de la acción a realizar
- void pintamenu(); Muestra el menú de selección
- void pintagrafo(grafo *gr2); Muestra el menú del juego
- ~grafico(); Destructor gráfico

Clase juego: La clase juego contiene las rutinas y algoritmos que dan forma a la carrera de coche que se produce al desplazarse de un monumento a otro de la ciudad, posee los siguientes métodos:

- juego();
- bool choque (int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
- void comp_choque (coche *pc, clock_t &start_parada, lista<segmento> *lista_seg, int &c, bool &sw, int &puntuacion, int &tiempo);
- bool carrera (char * file,grafo *gra,coche *p_coche, turista *t);
- void cargar_carrera (grafico *g1, grafo *gr, coche *p_coche, turista *turi, int j, bool visitados[]);
- coche * eligecoche (coche *pc_normal, coche *pc_rapido, turista *turi, bool &creado_n, bool &creado_r);
- ~juego();

Los métodos de la clase juego, realizan las siguientes funciones:

- Juego Constructor de la clase juego
- choque Verifica si se ha producido un choque entre el coche y los objetos
- comp_choque Comprueba contra que tipo de objeto chocó y realiza una acción
- carrera Muestra el coche y desplaza los segmentos de la carretera
- cargar_carrera Verifica si existe un camino, si es así carga el fichero correcto
- eligecoche Comprueba la puntuación y devuelve el puntero al coche correcto
- ~juego Destructor de la clase juego

Estructuras de datos utilizadas

Para la implementación de este programa se han usado diversas estructuras de almacenamiento dinámica. En concreto, una lista lineal simplemente enlazada, y una cola, ambas implementadas mediante plantillas. Así como un árbol binario de turistas, y un grafo formado por un vector y una matriz de tamaño fijo.

En primer lugar abordaremos el uso de la lista, la cual hemos utilizado para almacenar datos cuya cantidad era variable, es decir que almacenaremos una cantidad distinta de elementos cada vez que la usemos, por lo que no es óptimo el uso de una estructura de tamaño fijo, como podría ser un vector. Además necesitábamos acceder a la información repetidas veces, por lo que una cola por ejemplo no nos hubiese servido, ya que al desencolar perderíamos los elementos.

Se optó por implementar la lista mediante plantillas para poder reutilizarla con distintas clases, así por ejemplo, se ha utilizado una lista para almacenar los objetos de cada segmento, otra para almacenar los segmentos que forman la carretera, la cual debe ser recorrida constantemente para poder moverlos de posición, otra lista para almacenar aquellas bonificaciones que vaya recogiendo el coche durante la carrera, y otra para almacenar los monumentos visitados por un turista.

Para almacenar los segmentos mientras se iban creando, conforme se iba leyendo la información del fichero, se optó por usar una cola, ya que sólo se usaría una vez, para posteriormente almacenar los segmentos en una lista y poder trabajar con ellos. Además el primer elemento leído del fichero debía ser el primero insertado en la lista, así que una cola era la estructura perfecta, ya que el primer elemento en entrar es el primero en salir.

El grafo resultaba ser la estructura más eficiente para almacenar los distintos caminos que comunicaban los monumentos, debido a que los monumentos eran un número constante, pudimos usar una estructura fija. De este modo el grafo se compuso de un vector de seis posiciones de punteros a cadenas de caracteres, en las que se almacenarán los nombres de los monumentos leídos de los ficheros, además de una matriz de seis por seis posiciones en cuyas celdas se almacenarán las calles que unen los distintos monumentos y la longitud que tienen. Para ello se definieron las celdas como estructuras que contenían un puntero a la cadena con el nombre de la calle, y un entero con el tamaño de la calle.

Para almacenar a los turistas de forma ordenada, se utilizó otra estructura de tipo dinámico, un árbol de búsqueda binario, en el que las ramas de cada nodo están ordenadas de modo que si es menor que el nodo raíz, se sitúa en el subárbol izquierdo y si es mayor en el subárbol derecho, facilitándose así la tarea de encontrar a los turistas. Cada nodo se forma pues, por un puntero de tipo turista y dos punteros de tipo nodo (uno para la raíz del subárbol izquierdo y otro para la del subárbol derecho).

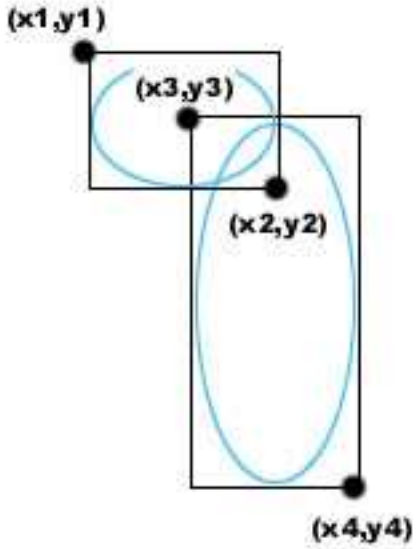
Implementación

Algoritmos de especial interés

Uno de los algoritmos más importantes del programa es el bucle infinito que se repite de forma continua a lo largo de la función carrera, de la clase juego. Mediante este algoritmo se obtiene el movimiento de la carretera. Este efecto se consigue mediante la continua repetición de una serie de acciones que a continuación se detallan:

```
mientras (verdadero)
{
        Si se pulsa una tecla ...
        {
                se almacena la tecla en la variable c ...
                se mueve el coche en función de la tecla.
                Comprueba que tipo de objeto chocó en caso de producirse un choque
        }

        Tomamos la hora de fin del reloj

        Si han pasado más de 2 segundos
        {

        Se copia el segmento de la cola y se inserta en la lista de segmentos
        Borramos el segmento de la pantalla, pintándolo de color negro
        Movemos los segmentos, modificando los valores de sus coordenadas
        Se pintan los segmentos de la lista
        Comprueba que tipo de objeto chocó en caso de producirse un choque
        Elimina el segmento de la lista que se sale por la parte inferior de la pantalla

        Si la lista está vacía
                {
                        Se muestran los puntos
                        Se incrementa la puntuación
                        Se marca como carrera completada
                        Se inicializa c como si se pulsase escape
                }

        Tomamos la hora de inicio del reloj

        }

        Si se pulsa una tecla se captura, y se iguala a c
        Si c igual a escape
        {
                Rompemos el bucle infinito
        }
  }
```

Otro algoritmo especialmente importante es el encargado de determinar cuando se produce un choque entre un objeto y el vehículo. Como está operación debe realizarse continuamente, cada vez que se mueve el coche o un segmento, se determino sacar el algoritmo fuera del programa principal, insertandolo en una función de la clase juego.
La función se basa en el siguiente principio, para que se produzca un choque alguna de las coordenadas del rectángulo que delimita la carrocería del coche debe estar entre las coordenadas del rectángulo que delimita al objeto, o viceversa.



CONDICIONES PARA QUE SE PRODUZCA UNA COLISIÓN

((x1<=x3<=x2) ó (x1<=x4<=x2)) y ((y1<=y3<=y2) ó (y1<=y4<=y2))
((x3<=x1<=x4) ó (x3<=x2<=x4)) y ((y3<=y1<=y4) ó (y3<=y2<=y4))

```
bool choque(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
        Se inicializan las variables booleanas a falso
        Si ((x1<=x3) y (x3<=x2))      condicion1=verdadera;
                //Condicion1= si la x de la esquina superior izq del coche
                //está entre las esquinas del objeto a nivel x
        Si ((x1<=x4) y (x4<=x2))      condicion2=verdadera;
                //Condicion2= si la x esquina inf der del coche
                //está entre las esquinas del objeto a nivel x
        Si ((condicion1) ó (condicion2))       peligrox=verdadera;
                //Si cualquier esquina del coche está entre las
                //esquinas del objeto existe peligro de choque
                //en el eje x.
        Si ((y1<=y3) y (y3<=y2))      condicion3=verdadera;
                //Condicion3= si la "y" de la esquina superior izq del coche
                //está entre las esquinas del objeto a nivel y
        Si ((y1<=y4) y (y4<=y2))      condicion4=verdadera;
                //Condicion4= si la "y" esquina inferior der del coche
                //está entre las esquinas del objeto a nivel y
        Si ((condicion3) ó (condicion4))       peligroy=verdadera;
                //Si cualquier esquina del coche está entre las
                //esquinas del objeto existe peligro de choque
                //en el eje y.
        Si ((peligrox) y (peligroy))    devuelve choque verdadero;
                //Si hay peligro de choque en ambos ejes
                //existe choque
        Si no
        {
                Volvemos a inicializar variable a falso
                Si ((x3<=x1) y (x1<=x4))      condicion1=verdadera;
                Si ((x3<=x2) y (x2<=x4))      condicion2=verdadera;
                Si ((condicion1) ó (condicion2))       peligrox=verdadera;
                Si ((y3<=y1) y (y1<=y4))      condicion3=verdadera;
                Si ((y3<=y2) y (y2<=y4))      condicion4=verdadera;
                Si ((condicion3) ó (condicion4))       peligroy=verdadera;
                Si ((peligrox) y (peligroy))    devuelve choque verdadero;
                Si no
                        devuelve choque falso;
    }

}
```

Antes de saber si hay choque hay que comprobarlo, y averiguar contra que tipo de objeto, y que acción debemos realizar.

```
void comp_choque    (coche *pc_rapido, clock_t &start_parada, lista<segmento>
                    *lista_seg, int &c, bool &sw, int &puntuacion, int &tiempo)
{
        Comprobamos la colision del coche con los objetos
        Para ello obtenemos la forma de la carrocería
        Y calculamos el rectángulo que contiene a la carrocería
        Obtenemos las coordenadas del rectángulo

        /Recorremos la lista de segmentos buscando choques
        {
                        leemos segmento de la lista
                        obtenemos la lista de objetos
                        inicializamos la lista
                        Recorremos la lista de objetos
                                {
                                        Leemos un objeto
                                        Obtenemos su forma
                                        Calculamos las coordenadas
                                        Del rectángulo que contiene la figura
                                        Obtenemos las coordenadas del objeto
                                        Llamamos a la función choque
                                        para comprobar si ha habido una colisión
                {
                        Si es de tipo accidente
                                {
                                        Tomamos el tiempo
                                        Ponemos sw a falso
                                        Borramos el objeto
                                        Tomamos el tiempo de parada del accidente
                                }

                        Si es de tipo bache
                                {
                                        Obtenemos la lista de bonificaciones
                                        encontrado=false;

                                        Si no está vacía la lista de bonificaciones
                                        {
                                                Buscamos una estatua
                                                Si la hay la borramos
                                                Si no c=escape
                                        }
                                }
```

```
                    Si es de tipo estatua
                        {
                            Obtenemos la lista de bonificaciones
                            copiando objeto estatua
                            lo insertamos en la lista
                            aumentamos la puntuación de la carrera
                        }

                    Si es de tipo vasija
                        {
                            Obtenemos la lista de bonificaciones
                            copiando objeto vasija
                            lo insertamos en la lista
                            aumentamos la puntuación de la carrera

                        }
                Si no choca avanza al siguiente objeto del segmento
                Avanza al siguiente segmento
        }
}
```

Otro algoritmo importante es el encargado de varifiacar la existencia de algún camino que una el monumento donde se encuentra actualmente el turista, con el monumento que desea visitar. Además, en el caso de que exista un camino, el algoritmo debera extraer el nombre del monumento destino y generar la cadena con el nombre del fichero que se debe cargar, para pasarselo a la función carrera.

```
cargar_carrera (grafico *g1, grafo *gr, coche *p_coche, turista *turi, int j, bool visitados[])
{
        Igualamos origen al valor de la variable origen del grafo
        Igualamos destino al valor de la variable destino del grafo
        Borramos el menú
        Si no existe un arco del grafo entre origen y destino
                {
                    Mostramos por pantalla un mensaje de advertencia
                    Pintamos el grafo de nuevo
                }
        Si existe arco
                {
                    Iniciamos posi a la posición de origen
                    Reservamos memoria para el nombre de la calle, más 5 posiciones
                    Copiamos el nombre de la calle desde el grafo
                    Encadenamos ".txt" más fin de línea
                    Si termina la carrera correctamente
                        {
                            Imprimimos un mensaje
                            Ponemos en el grafo el nuevo origen
                            Inicializamos origen al nuevo origen del grafo
```

```
                                        Si no fue visitado con anterioridad
                                          {
                                                Marcamos el monumento como visitado
                                                Agregamos el monumento al turista
                                          }
                                    }

                        Si no termina la carrera correctamente. Mostramos un mensaje

                        Borramos la variable kalle
                }
}
```

Para poder cambiar de coche durante el desarrollo del juego, se pensó en declarar dos tipos de coches en el programa principal (normal y rápido), e implementar un puntero de la clase base coche, el cual mediante el siguiente algoritmo, debería apuntar al coche correcto, a la vez que ese coche conserva o hereda la lista de bonificaciones, obtenidas en las anteriores carreras.

```
coche * eligecoche (coche *pc_normal, coche *pc_rapido, turista *turi, bool  &creado_r)
{
        Si turista posee menos de 1000 puntos.  Devolvemos el puntero al coche normal
        Si turista tiene 1000 puntos o más y nunca se apuntó al coche rápido
        {
                Lista_b se iguala al puntero de la lista de bonificaciones de coche normal
                Se inicializa lista_b
                Mientras que lista_b no se termine
                  {
                        Obj se iguala al objeto que se le de lista_b
                        Si es de tipo estátua se copia una estatua y se inserta en la lista
                        Si es de tipo  vasija se copia una vasija y se inserta en la lista
                        Encolamos en una cola de bonificaciones
                  }
                Igualamos lista_b a la lista de bonificaciones del coche rápido
                Mientras tenga bonificaciones la cola
                  {
                        Se saca el primer objetos de la cola
                        Si es de tipo estátua se copia una estatua y se inserta en la lista
                        Si es de tipo  vasija se copia una vasija y se inserta en la lista
                        Se inserta el objeto en la lista_b
                        Desencolamos
                  }
                Ponemos creado_r a verdadero
                Devolvemos el puntero al coche rápido
        }
        Si tiene 1000 puntos o más y se apuntó al coche rápido
        {
                Devolvemos el puntero al coche rápido
        }
}
```

Historial de desarrollo

A continuación describiremos a grandes rasgos, los objetivos de cada avance y las modificaciones que se han introducido, así como los problemas que se han encontrado en cada paso del desarrollo.

- 19-02-08: Transformación del ejemplo de funcionamiento del entorno gráfico, en una clase con atributos y métodos, para su uso en el proyecto.
- 26-02-08: Diseño bajo el modo gráfico de la clase figura, sus atributos y métodos.
- 05-03-08: Diseño de las clases que heredan de figura: Círculo, rectángulo y elipse. Manejo de métodos hijos desde instancias de la clase base. Virtualización de métodos.
- 07-03-08: Diseño de la clase base coche, usando dos vectores comunes a ambos tipos de coche, uno para las cuatro ruedas y otro de dos posiciones para los faros. A su vez las clases derivadas de coche incluyen un atributo carrocería de tipo figura que variará en función del tipo de coche, y en el caso del coche rápido un vector de dos posiciones para las ruedas extras.
- 10-03-08: Rediseño de la clase coche, que incluye los faros, las ruedas y la carrocería en un vector de siete posiciones, en la clase base coche. La clase coche normal incluye como atributo un vector de dos posiciones para las ruedas extras. La clase coche rápido posee el atributo específico de velocidad vertical.
- 11-03-08: Diseño de los algoritmos necesarios para pintar y mover un coche desde el programa principal, así como para controlar la memoria perdida.
- 18-03-08: Acotar el movimiento del coche en pantalla, diseño de la clase objeto de la que heredan las clases bases bonificación y obstáculo, así como sus respectivas clases derivadas: estatua, vasija, accidente, y bache.
- 26-03-08: Esbozo de la clase segmento (sin lista de objetos), y de los métodos necesarios para pintar en pantalla los distintos objetos. Diseño de una lista de datos de tipo entero, para su comprensión y posterior reutilización.
- 02-04-08: Optimización de memoria, optimización de la lista de enteros (para luego conseguir una lista de objetos), y corrección de diversos fallos.
- 03-04-08: Definición en el programa principal de una lista de objetos, en la que se insertan distintos objetos y se muestran por pantalla junto al coche.
- 07-04-08: Diseño de una cola de datos de tipo entero con sus métodos para encolar, desencolar, comprobar si está vacía, y mostrar por pantalla el primer elemento.
- 08-04-08: Adaptación de la cola de enteros a una cola de segmentos, y corrección de fallos.
- 09-04-08: Lectura de los ficheros de texto, para lo cual se usa y modifica el fichero línea. Almacenamiento de los valores contenidos en los ficheros y obtención de los distintos campos.
- 11-04-08: Primera entrega para la evaluación por partes, en la que se leen los valores de los ficheros y se crean los distintos objetos que aparecen, para a continuación insertarlos en la lista de objetos que posee el segmento. Una vez completado el segmento se inserta en la cola de segmentos. Se muestran los segmentos por pantalla, junto con un coche de cada tipo, los cuales se moverán por pantalla.

- 18-04-08: Se optimizan la lista y la cola para que no se pierda memoria. Se inicia la adaptación de la lista y la cola para uso genérico (templates).
- 22-04-08: Sustitución de la lista y la cola por estructuras genéricas (templates). Se suprime la clase base obstáculo, de modo que las clases bache y accidente heredan directamente de la clase base objeto. Diseño de los constructores copia necesarios para copiar un segmento.
- 30-04-08: Se incluye una lista de bonificaciones en la clase coche. El programa principal se modifica para leer los segmentos desde los ficheros, los cuales se insertan en una cola de segmentos, para posteriormente sacar los segmentos de la cola, copiarlos e insertarlos en una lista de segmentos. A continuación se leen los segmentos de la lista y se muestran por pantalla.
- 10-05-08: Diseño del algoritmo de movimiento de los segmentos que componen la carretera, los cuales se desplazan a través de la pantalla. Comienzo del diseño del algoritmo de control de choques, calculando el rectángulo que contiene a cada objeto.
- 15-05-08: Diseño del bucle infinito que simula el movimiento de la carretera. Optimización del movimiento de segmentos, para mostrar el primer segmento al comienzo de la pantalla, y para que desaparezcan todos por el fondo de la pantalla. Diseño de la función de evaluación de choques, que usa como parámetros las coordenadas de los rectángulos que engloban los objetos.
- 16-05-08: Diseño del grafo, e inclusión del control de gasolina al desaparecer tres segmentos por pantalla. También se comienza a almacenar la puntuación obtenida al chocar un coche con una bonicicación.
- 19-05-08: Depuración del grafo, e inclusión del control de baches, para que finalice el programa en caso de que el coche no posea ninguna rueda en su lista de bonificaciones.
- 22-05-08: Inclusión del control de accidentes, y depuración del código sacando el algoritmo de comprobación de colisiones a una función fuera del programa principal, inicio de la lectura del fichero monument.txt, con el cual se rellena el contenido del grafo.
- 23-05-08: Se optimiza el código, y se extrae a una función el código que decide la acción a realizar dependiendo del objeto con el que se choca. Se corrige la eliminación del objeto con el que se colisiona en el segmento, creando el método eliminar actual de la lista, ya que el anterior, que comparaba punteros a objetos y lo eliminaba, ocasionaba errores.
- 24-05-08: Se optimiza el bucle de movimiento de la carretera para que muestre la carretera y el coche en movimiento de forma simultánea. Se optimiza el código, depurando pequeños errores.
- 27-05-08: Lectura del fichero monument.txt y correcto volcado de información en el grafo. Diseño de métodos necesarios para el grafo.
- 29-05-08: Diseño del entorno gráfico de los nodos del grafo y control de las pulsaciones del ratón.
- 04-06-08: Reutilización de una estructura de árbol, y diseño del algoritmo para pintar los arcos, que unen los nodos del grafo. Implementación del código para la carga de un fichero u otro en función del destino elegido.

- 09-06-08: Reestructuración del código para sacar el algoritmo de la carrera, a una función independiente del programa principal, al igual que se hace con el código encargado de pintar y borrar las distintas pantallas gráficas, con las que interactúa el usuario. Se dota al coche de las características leídas de los ficheros (velocidad y gasolina), para lo cual se crean nuevos métodos en las clases.
- 10-06-08: Se crea la clase turista, y se almacena en ella el atributo de la puntuación, que permite crear un coche rápido, en el caso de que sea superior mil. Se extrae del programa principal a una función el código que controla si hay arco entre origen y destino, y que carga el fichero de carrera en caso afirmativo.
- 12-06-08: Se adapta un ejemplo de estructura de árbol para incorporarlo al proyecto, se lee el fichero de turistas y se cargan en la estructura de árbol, para finalmente mostrarlos ordenados alfabéticamente por pantalla.
- 14-06-08: Se diseña el código, para almacenar la fecha actual de la visita del turista que juega, así como la lectura y escritura de fechas desde ficheros. Se estrecha el ancho del segmento para poder mostrar por pantalla de forma constante la puntuación, las bonificaciones, y la gasolina.
- 16-06-08: Se corrigen errores de apertura de ficheros que impedían la correcta escritura de los datos del turista en el fichero. Se solucionan pequeños fallos de código en el control de gasolina.
- 28-06-08: Se crean en el programa principal dos coches: uno normal y otro rápido, y un puntero de tipo coche será el encargado de discernir que tipo de coche se usa en cada carrera, en función de la puntuación que acumule el turista. Para ello se implementan las funciones necesarias.
- 01-07-08: Se optimiza el código y se sacan todas las funciones del programa principal como métodos de la clase juego y de la clase gráfico.

Listado del código fuente

Principal.cpp

```
/*********************************************************
 * Alumno: Raúl J. Zambrano Maestre                      *
 * Asignatura: Labratorio de Progrmación II              *
 * Titulación: I. T. Informática de Gestión              *
 * Curso: 2007-2008                                      *
 * Fecha de creación: 10 de Marzo de 2008                *
 * Última modificación: 21 de Agosto de 2008             *
 * Compilador Utilizado: Borland C++ 5.02 para Windows   *
 *********************************************************/
#include <alloc.h>
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include <iostream.h>
#include <time.h>
#include <typeinfo.h>
#include <dos.h>
#include <fstream.h>
#include "turista.h"
#include "grafico.h"
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "coche.h"
#include "normal.h"
#include "rapido.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"
#include "vasija.h"
#include "bache.h"
#include "accidente.h"
#include "lista.h"
#include "segmento.h"
#include "cola.h"
#include "linea.h"
#include "grafo.h"
#include "arbol.h"
#include "juego.h"
```

```
#define MAX 6

/*********************************/
/*      Programa principal       */
/*********************************/

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

  juego *j1=new juego();
  grafico *g1=new grafico();
  arbol *olivo=new arbol();
  coche *p_coche,*pc_normal,*pc_rapido;
  char * name, *nacion, *monumento;
  turista *nuevot, *turi;
  int d,m,a;
  bool visitados[MAX]={true,false,false,false,false,false};
  bool creado_r=false;

  fstream monument(argv[1],ios::in);   //Abrimos fichero monumentos para lectura
  if (!monument)
    {
          cout<<"Error al abrir el fichero de monumentos "<<argv[1]<<endl;
          getchar();
          exit(1);
     }

  grafo *gr=new grafo(argv[1]);

  monument.close();                    //Cerramos el fichero de monumentos

  linea *lin=new linea();              //Puntero linea

   char *cadena;                       //Puntero a cadena

   fstream turistas(argv[2],ios::in);  //Abrimos el fichero para lectura
   if (!turistas)
     {
          cout<<"Error al abrir el fichero de turistas "<<argv[2]<<endl;
          getchar();
          exit(1);
     }
   if ((cadena=new char[200])==NULL)
     {
          cout<<"Error en la reserva de memoria..";
          getchar();
          exit(1);
     }
```

```
while (!turistas.eof())
 {
      cadena[0]='\0';              //Inicializamos la cadena en blanco.
      turistas>>cadena;            //Leemos una cadena del fichero de turistas
      lin->separar(cadena);        //Separamos los campos de la cadena

      nuevot=new turista(lin->obtener_campo(0),lin->obtener_campo(1));
      //Creamos un nuevo turista con los campos nombre y nacionalidad
      d=atoi(lin->obtener_campo(2));      //Obtenemos el dia de la ultima visita
      m=atoi(lin->obtener_campo(3));      //Obtenemos el mes de la ultima visita
      a=atoi(lin->obtener_campo(4));      //Obtenemos el año de la ultima visita
      nuevot->set_fecha(d,m,a);           //Le ponemos a ese turista la fecha leida

      int ind=6;      //Indice de campos que comienza en el primer monumento
                      //visitado (excluye a la O.Turismo)

      while (strlen(lin->obtener_campo(ind))>1)
        {
              monumento=new char[strlen(lin->obtener_campo(ind))+1];
              //Reserva memoria para la cadena
              strcpy(monumento,lin->obtener_campo(ind));
              //Copiamos el campo
              monumento[strlen(lin->obtener_campo(ind))]='\0';
              //Insertamos fin de línea
              nuevot->set_monumento(monumento);
              //inserta el monumento en la lista de visitados

              ind++;
        }

      olivo->insertar(nuevot);  //Una vez leido los campos, lo insertamos en el arbol
   }

turistas.close();                 //Cerramos el fichero de turistas

g1->pintamenu();

bool finalizar=false;
while (!finalizar)
 {
      switch (g1->esperarseleccion())
      {
              case 1:

                      g1->pintainstr();
                      g1->pintamenu();
                      break;
```

```
                case 2:

                        g1->borramenu();
                        finalizar=true;
                        break;

                case 3:

                        g1->borramenu();
                        char *mm,*aa;
                        mm=new char[2];
                        aa=new char[4];
                        gotoxy(1,1);
                        cout<<"Busqueda de turistas por fecha: "<<endl;
                        gotoxy(1,3);
                        cout<<"Inserta el mes: ";
                        cin>>mm;
                        gotoxy(1,4);
                        cout<<"Inserta el anio: ";
                        cin>>aa;
                        olivo->buscar(mm,aa);

                        g1->borramenu();
                        g1->pintamenu();

                        break;

                case 4:

                        g1->borramenu();
                        /* Ocultar raton para finalizar */
                        g1->OcultarRaton();
                        /*Cierre del modo gráfico*/
                        closegraph();

                        cout<<"FIN";
                        getchar();
                        return(0);

        }

}

g1->borramenu();

name=new char[20];
nacion=new char[20];
name[0]='\0';
nacion[0]='\0';
```

```
gotoxy(1,1);
cout<<"Inserta el nombre del turista: ";
cin>>name;
gotoxy(1,2);
cout<<"Inserta la nacionalidad: ";
cin>>nacion;

turi=new turista(name,nacion);
turi->pilla_fecha();

pc_normal=new normal(320,420);  //Creamos un coche normal
pc_rapido=new rapido(320,420);    //Creamos un coche rapido

g1->pintagrafo(gr);

 bool salir=false;
 while (!salir)
       {
         int j;
         switch (g1->esperarpulsacion())
           {
               case 1:
                        j=0;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;

               case 2:
                        j=1;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;

               case 3:
                        j=2;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;

               case 4:
                        j=3;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;
```

```
                case 5:
                        j=4;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;

                case 6:
                        j=5;
                        p_coche=j1->eligecoche(pc_normal, pc_rapido, turi, creado_r);
                        j1->cargar_carrera (g1,gr,p_coche,turi,j,visitados);
                        g1->pintagrafo(gr);
                        break;

                case 7:
                        delete pc_normal;
                        delete pc_rapido;
                        //delete p_coche;

                        g1->borramenu();
                        salir=true;

                        cout<<"NOMBRE: "<<turi->get_nombre()<<endl;
                        cout<<"NACIONALIDAD: "<<turi->get_nacionalidad()<<endl;
                        cout<<"PUNTUACION TOTAL: "<<turi->get_puntos()<<endl;
                        turi->mostrar_monumentos();
                        getchar();

                        olivo->insertar(turi);     //Insertamos el turista en el árbol

                        cout<<endl<<"Contenido del arbol..."<<endl;
                        olivo->mostrar();

                        g1->borramenu();

                        olivo->guardar(argv[2]);

                        break;
            }
        }

        g1->OcultarRaton();             // Ocultar raton para finalizar

        closegraph();                   // Cierre del modo gráfico

        cout<<"FIN";
        getchar();
        return(0);
}
```

Juego.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#ifndef juego_h
#define juego_h

class juego
{
protected:


public:

juego ();
bool choque (int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);
void comp_choque (coche *pc, clock_t &start_parada, lista<segmento> *lista_seg, int
&c, bool &sw, int &puntuacion, int &tiempo);
bool carrera (char * file,grafo *gra,coche *p_coche, turista *t);
void cargar_carrera (grafico *g1, grafo *gr, coche *p_coche, turista *turi, int j, bool
visitados[]);
coche* eligecoche (coche *pc_normal, coche *pc_rapido, turista *turi, bool &creado_r);
~juego();

};

#endif
```

Juego.cpp

```
#include <alloc.h>
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include <iostream.h>
#include <time.h>
#include <typeinfo.h>
#include <dos.h>
#include <fstream.h>
#include "turista.h"
#include "grafico.h"
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "coche.h"
#include "normal.h"
#include "rapido.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"
#include "vasija.h"
#include "bache.h"
#include "accidente.h"
#include "lista.h"
#include "segmento.h"
#include "cola.h"
#include "linea.h"
#include "grafo.h"
#include "arbol.h"
#include "juego.h"
#define MAX 6

juego::juego()
{
        cout<<"Ha entrado en el constructor juego"<<endl;
        getchar();
}

juego::~juego()
{
        cout<<"Ha entrado en el destructor juego"<<endl;
}
```

```
bool juego::choque(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
        // CONDICION DE CHOQUE DE LOS OBJETOS CON EL COCHE

        bool condicion1=false;
        bool condicion2=false;
        bool condicion3=false;
        bool condicion4=false;
        bool peligrox=false;
        bool peligroy=false;

        if ((x1<=x3) && (x3<=x2))  condicion1=true;
                                //Condicion1= si la x de la esquina superior izq del coche
                                //está entre las esquinas del objeto a nivel x
        if ((x1<=x4) && (x4<=x2))  condicion2=true;
                                //Condicion2= si la x esquina inf der del coche
                                //está entre las esquinas del objeto a nivel x
        if ((condicion1)||(condicion2))         peligrox=true;
                                //Si cualquier esquina del coche está entre las
                                //esquinas del objeto existe peligro de choque
                                //en el eje x.
        if ((y1<=y3) && (y3<=y2))  condicion3=true;
                        //Condicion3= si la "y" de la esquina superior izq del coche
                        //está entre las esquinas del objeto a nivel y
        if ((y1<=y4) && (y4<=y2))  condicion4=true;
                                //Condicion4= si la "y" esquina inferior der del coche
                                //está entre las esquinas del objeto a nivel y
        if ((condicion3)||(condicion4))         peligroy=true;
                                        //Si cualquier esquina del coche está entre las
                                        //esquinas del objeto existe peligro de choque
                                        //en el eje y.
        if ((peligrox)&&(peligroy))   return true;
                                        //Si hay peligro de choque en ambos ejes
                                        //existe choque
        else
          {
                condicion1=false;
                condicion2=false;
                condicion3=false;
                condicion4=false;
                peligrox=false;
                peligroy=false;

                if ((x3<=x1) && (x1<=x4))  condicion1=true;
                                        //Condicion1= si la esquina superior izq del obj
                                        //está entre las esquinas del coche a nivel x
                if ((x3<=x2) && (x2<=x4))  condicion2=true;
                                        //Condicion2= si la esquina inf derecha del obj
                                        //está entre las esquinas del coche a nivel x
```

```
                if ((condicion1)||(condicion2))         peligrox=true;
                                        //Si cualquier esquina del objeto está entre las
                                        //esquinas del coche existe peligro de choque
                                        //en el eje x.
                if ((y3<=y1) && (y1<=y4))   condicion3=true;
                                        //Condicion3= si la esquina superior izq del objeto
                                        //está entre las esquinas del coche a nivel "y"
                if ((y3<=y2) && (y2<=y4))   condicion4=true;
                                        //Condicion4= si la esquina inf der del objeto
                                        //está entre las esquinas del coche a nivel "y"
                if ((condicion3)||(condicion4))         peligroy=true;
                                        //Si cualquier esquina del objeto está entre las
                                        //esquinas del coche existe peligro de choque
                                        //en el eje y.
                if ((peligrox)&&(peligroy))   return true;
                else      return false;
        }
}

void juego::comp_choque (coche *pc, clock_t &start_parada, lista<segmento>
*lista_seg, int &c, bool &sw, int &puntuacion, int &tiempo)
{
        int x1,y1,x2,y2,x3,y3,x4,y4;

        /*COLISIÓN DEL COCHE CON LOS OBJETOS*/

        bool encontrado=false;
        objeto *p_obj, *item;
        segmento *p_seg;
        figura *p_fig;
        lista<objeto> *list_obj;
        lista<objeto> *list_ext;

        p_fig=pc->darcarroceria();
        pc->calcularc(p_fig);

        x3=pc->get_x3();
        y3=pc->get_y3();
        x4=pc->get_x4();
        y4=pc->get_y4();

        lista_seg->iniciar();
        while(lista_seg->noesfin())             //Recorremos la lista buscando choques
                {
                        p_seg = lista_seg->leer();      //leemos segmento de la lista
                        list_obj = p_seg->darlista();   //obtenemos la lista de objetos
                        list_obj->iniciar();
```

```
            while (list_obj->noesfin())    //Recorremos la lista de objetos
            {
              p_obj=list_obj->leer();      //Leemos un objeto
              p_fig=p_obj->darforma();  //Obtenemos su forma

              p_obj->calcular(p_fig);      //Calculamos las coordenadas del
                                           //rectangulo que contiene la figura
              x1=p_obj->get_x1();
              y1=p_obj->get_y1();   //Obtenemos las coordenadas del objeto
              x2=p_obj->get_x2();   //(x1,y1),(x2,y2)
              y2=p_obj->get_y2();

// CONDICIONES DE COLISIÓN DEL COCHE CON OBJETOS

if (choque(x1,y1,x2,y2,x3,y3,x4,y4))
  //Llama a la función choque para comprobar si ha habido una colisión
  {
    if (typeid(*(p_obj))==typeid(accidente))
      {
          start_parada=clock();
          sw=false;
          p_obj->borrar();
          tiempo=((accidente *)p_obj)->get_tiempo();
       }

    else if (typeid(*(p_obj))==typeid(bache))
      {
          list_ext=pc->darextras();      //Da la lista de bonificaciones
          encontrado=false;

          if (!list_ext->vacia())
            {
                list_ext->iniciar();
                while ((list_ext->noesfin())&&(!encontrado))
                  {
                      item=list_ext->leer(); //leemos un item de la lista
                      if (typeid(*(item))==typeid(estatua)) //Si es estatua
                      {
                          list_ext->eliminar_actual();   //Borra estatua
                          encontrado=true;          //item encontrado
                      }
                      else     list_ext->avanzar();
                          //Si no se encuentra avanzamos
                }
            }

          if (!encontrado)        c=27;
          p_obj->borrar();

        }          //fin condición bache
```

```
                        else if (typeid(*(p_obj))==typeid(estatua))
                          {
                             list_ext=pc->darextras();
                             item=new estatua(*((estatua *)p_obj));   //copiando estatua
                             list_ext->insertar(item);
                              puntuacion=puntuacion+((bonificacion *)p_obj)->get_puntos();
                               p_obj->borrar();
                           }                        //fin condición estatua

                        else if (typeid(*(p_obj))==typeid(vasija))
                           {
                              list_ext=pc->darextras();
                              item=new vasija(*((vasija *)p_obj)); //copiando vasija
                               list_ext->insertar(item);
                              puntuacion=puntuacion+((bonificacion *)p_obj)->get_puntos();
                              p_obj->borrar();
                           }                        //fin condición vasija

                        list_obj->eliminar_actual();

                }       //** FIN DE CONDICION DE CHOQUE DEL COCHE **

                else    //Si no choca avanza al siguiente objeto del segmento
                {
                                list_obj->avanzar();
                }

            }           //fin while lectura lista objetos

            lista_seg->avanzar();       //Avanza al siguiente segmento
        }
}

bool juego::carrera(char * file,grafo *gra,coche *p_coche, turista *t)
{
        gotoxy(1,6);
        cout<<"CALLE: "<<file<<endl;
        linea *lin=new linea();                 //Puntero linea
        char *cadena;                           //Puntero a cadena

        fstream fiche(file,ios::in);            //Abrimos el fichero para lectura
        if (!fiche)
        {
                cout<<"Error al abrir el fichero."<<file;
                getchar();
                exit(1);
        }
```

```
if ((cadena=new char[80])==NULL)
{
        cout<<"Error en la reserva de memoria.";
        getchar();
        exit(1);
}
bool completado=false;        //Carrera completada
bool encontrado=false;
cadena[0]='\0';
int cont=0;
int cuenta_seg=0;
int col,xs,ys,xi,yi,xx,yy,cod_seg,litros,puntos,gasoil,tiempo;
int contenido=0;
int puntuacion=0;
segmento *p_seg,*copia_seg;
objeto *p_obj, *item;
char *qes;                    //Contenido del campo 3 de fiche (tipo objeto)
figura *kretera,*p_fig;
lista<objeto> *list_obj, *list_ext;

while ((!fiche.eof()) && (cont<5))   //Sacamos las primeras 5 lineas
{
        fiche>>cadena;               //lee una línea del fichero
        lin->separar(cadena);        //separa los campos marcados por #

        if (cont==0)   col=atoi(lin->obtener_campo(0));     //1ª línea=color
        if (cont==1)   xs=atoi(lin->obtener_campo(0));  //2ª línea=x superior
        if (cont==2)   ys=atoi(lin->obtener_campo(0));  //3ª línea=y superior
        if (cont==3)   xi=atoi(lin->obtener_campo(0));  //4ª línea=x inferior
        if (cont==4)   yi=atoi(lin->obtener_campo(0));  //5ª línea=y inferior

        cont=cont++;  //se incrementa el contador de líneas
}

cola<segmento> *p_cola;
p_cola=new cola<segmento>();

int cod=1;

fiche>>cadena;                       //lee la 1ª cadena del fichero

lin->separar(cadena);                //separa los campos de la cadena leida
cod_seg=atoi(lin->obtener_campo(0));   //1º campo=número del segmento

while (!fiche.eof())                 //mientras no sea fin de fichero
{
        p_seg=new segmento(xs,ys,col,xi,yi);  //creamos nuevo segmento
        list_obj=p_seg->darlista();           //da el puntero a lista obj de seg
```

```
		while ((cod_seg==cod) && (!fiche.eof()))
				//Mientras el campo de segmento sea = q cod,
		{		//significa que es un objeto del segmento con ese cod
			qes=(lin->obtener_campo(2));		//Indica que tipo de objeto es
			xx=atoi(lin->obtener_campo(3))+xs;
			yy=atoi(lin->obtener_campo(4))+ys;

			if (strcmp(qes,"bache")==0)
			//Compara el 3 campo con la cadena "bache"
			{
				p_obj=new bache(xx,yy);	//crea un objeto bache
				list_obj->insertar(p_obj);
			}
			if (strcmp(qes,"accidente")==0)
			//Compara el 3 campo con la cadena "accidente"
			{
				tiempo=atoi(lin->obtener_campo(5)); //inmoviliza coche
				p_obj=new accidente(xx,yy,tiempo); //crea un accidente
				list_obj->insertar(p_obj);		//inserta el objeto
			}
			if (strcmp(qes,"estatua")==0)
			//Compara el 3 campo con la cadena "estatua"
			{
				puntos=atoi(lin->obtener_campo(5));
				//Puntos que da esa bonificacion
				p_obj=new estatua(xx,yy,puntos);
				//Crea un objeto estatua
				list_obj->insertar(p_obj);
			}
			if (strcmp(qes,"vasija")==0)
			//Compara el 3 campo con la cadena "vasija"
			{
				puntos=atoi(lin->obtener_campo(5));	//Puntos que da
				litros=atoi(lin->obtener_campo(6));	//Litros de gasolina
				p_obj=new vasija(xx,yy,puntos,litros);//crea una vasija
				list_obj->insertar(p_obj);
			}

			fiche>>cadena;				//Lee línea del fichero

			lin->separar(cadena);			//Separa la línea en campos
			cod_seg=atoi(lin->obtener_campo(0));  //Obtiene el campo cod
		}						//Fin while cod_seg==cod

		p_cola->encolar(p_seg);			//Inserta el segmento en la cola

		cod++;				//Incrementa cod al código del siguiente segmento

	}	//Fin while not eof fiche
```

```
fiche.close();                                          //Cerramos el fichero

delete lin;
delete cadena;

lista<segmento> *lista_seg;            //Creamos un puntero a una lista de segmentos
lista_seg=new lista<segmento>();    //Creamos la lista
lista_seg->crear_lista();

p_coche->set_vel_horiz(gra->get_vel_horiz());
//Inicializamos la velocidad dada por el fichero q lee grafo
p_coche->set_gasolina(gra->get_gasolina());
//Inicializamos la gasolina dada por el fichero q lee grafo
p_coche->pintar();               //Pintamos el coche

clock_t start, end, start_parada, end_parada;

start = clock();                    //Tomamos el tiempo

int c;                                          //C tomará el valor recogido por teclado

//BUCLE INFINTO QUE MUESTRA LA CARRETERA EN MOVIMIENTO

 bool sw=true;       //SW=true indica que no hay accidentes

 while (true)
{
        if (sw)          //Si sw=true no hay accidentes y se leen las pulsaciones
         {
           if (kbhit())                          //Si se pulsa una tecla ...
              {
                  c=getch();                 //... se almacena en la variable c ...
                  p_coche->mover(c);    //... se mueve el coche en función de la tecla.

                  comp_choque (p_coche, start_parada, lista_seg, c, sw, puntuacion, tiempo);
                  //Comprueba que tipo de objeto chocó en caso de producirse un choque
              }
         }
        else               //Si sw no es true, es decir si hay un accidente
           {
                  end_parada=clock();              //Tomamos tiempo fin de parada
                  if (((end_parada - start_parada) / CLK_TCK)>tiempo)  sw=true;
                  //Si el fin de parada-el inicio supera el tiempo del accidente
                  //sw=true (no hay accidente).
           }

        end = clock();
```

```
  if (((end - start) / CLK_TCK)>2)       //Si han pasado mas de 2 segundos
  {

  //COPIA DE SEG. DE LA COLA SEGMENTOS, QUE SE INSERTAN EN LISTA SEG.

  if (!(p_cola->cola_vacia()))           //Mientras tenga segmentos la cola
    {
          p_seg=p_cola->primero();     //Sacamos el primer elemento de la cola
          copia_seg=new segmento(*(p_seg));   //copiamos el segmento
          lista_seg->insertar(copia_seg);          //enlistamos el seg. copiado
          p_cola->desencolar();          //Desencolamos el seg. y lo borramos
    }

//BORRADO DE SEGMENTO

  lista_seg->iniciar();
  while(lista_seg->noesfin())
    {
          p_seg = lista_seg->leer();             //leemos segmento de la lista
          kretera = p_seg->darcarretera();       //obtenemos la forma de la carretera
          kretera->borrar();                     //pintamos la carretera
          list_obj = p_seg->darlista();          //obtenemos la lista de objetos
          list_obj->iniciar();
          while (list_obj->noesfin())            //pintamos los objetos de la lista
            {
                p_obj=list_obj->leer();
                p_obj->borrar();
                list_obj->avanzar();
            }
          lista_seg->avanzar();
    }

//DESPLAZAMIENTO DE SEGMENTOS

  lista_seg->iniciar();
  while(lista_seg->noesfin())
    {
      p_seg=lista_seg->leer();
      kretera=p_seg->darcarretera();
      list_obj=p_seg->darlista();
      kretera->set_y(kretera->get_y()-ys);
      ((rectangulo *)kretera)->set_endy(((rectangulo *)kretera)->get_endy()-ys);
      list_obj->iniciar();                   //iniciamos el recorrido de lista objetos
      while (list_obj->noesfin())            //hasta que termine lista objetos
          {
                p_obj=list_obj->leer();      //bajamos la posición de los objetos
                p_fig=p_obj->darforma();     //obtenemos la forma del objeto
                p_fig->set_y(p_fig->get_y()-ys);
                //Se toma la Y q se dio a fig en el constuctor,
                //y se desplaza el ancho del segmento (valor -)
```

```
            if (typeid(*(p_fig))==typeid(rectangulo))
            ((rectangulo *)p_fig)->set_endy(((rectangulo *)p_fig)->get_endy()-ys);
            //Si es un rectangulo tambien se incrementan las (x,y) inferiores

            list_obj->avanzar();

            }

    lista_seg->avanzar();

    }

//PINTA MARCADOR

    gotoxy(1,1);
    cout<<"PUNTOS: "<<puntuacion;
    gotoxy(1,2);
    cout<<"           ";
    gotoxy(1,2);
    cout<<"GASOLINA: "<<p_coche->get_gasolina();
    gotoxy(1,8);
    cout<<"                ";
    gotoxy(1,8);
    cout<<"VASIJAS: ";

    list_ext=p_coche->darextras();      //Obtenemos la lista de bonificaciones
    list_ext->iniciar();                //La inicializamos
    while (list_ext->noesfin())         //Hasta que no se acabe la lista
      {
            p_obj=list_ext->leer();                     //Leemos objeto
            if (typeid(*(p_obj))==typeid(vasija))       cout<<"*";
                                                        //Si es una vasija pintamos *
            list_ext->avanzar();                        //Siguiente
      }

    gotoxy(1,9);
    cout<<"              ";
    gotoxy(1,9);
    cout<<"ESTATUAS: ";
    list_ext->iniciar();            //Iniciamos la lista de bonificaciones
    while (list_ext->noesfin())
    {
            p_obj=list_ext->leer();                 //Leemos
            if (typeid(*(p_obj))==typeid(estatua))      cout<<"@";
                                                    //Si es una vasija pintamos @
            list_ext->avanzar();                    //Siguiente
    }
```

```
//PINTA LOS SEGMENTOS DE LA LISTA

  lista_seg->iniciar();
  while(lista_seg->noesfin())
  {
        p_seg = lista_seg->leer();          //leemos segmento de la lista
        kretera = p_seg->darcarretera();    //obtenemos la forma de la carretera
        kretera->pintar();                  //pintamos la carretera
        list_obj = p_seg->darlista();       //obtenemos la lista de objetos
        list_obj->iniciar();
        while (list_obj->noesfin())         //pintamos los objetos de la lista
        {
              p_obj=list_obj->leer();
              p_fig=p_obj->darforma();
              p_obj->pintar();
              list_obj->avanzar();
        }
        lista_seg->avanzar();
  }

  comp_choque(p_coche, start_parada, lista_seg, c, sw, puntuacion, tiempo);


  //ELIMINA EL SEG. DE LA LISTA QUE SE SALE DE LA PANTALLA

  lista_seg->iniciar();
  p_seg = lista_seg->leer();          //leemos segmento de la lista
  kretera = p_seg->darcarretera();    //obtenemos la forma de la carretera
  if ((kretera->get_y())>=480)
    {
        lista_seg->eliminar(p_seg);   //borra 1º segmento de la lista>480
        cuenta_seg++;
        if ((cuenta_seg%3)==0)
          {
              p_coche->set_gasolina(p_coche->get_gasolina()-10);
              gasoil=p_coche->get_gasolina();
              if (gasoil<=20)
                {
                    contenido=0;
                    list_ext=p_coche->darextras();
                    if (!list_ext->vacia())
                    {
                          encontrado=false;
                          list_ext->iniciar();
                          while (list_ext->noesfin())
                          {
                                item=list_ext->leer();
```

```
                    if ((typeid(*(item))==typeid(vasija))&&(!encontrado))
                      {
                            contenido=((vasija *)item)->get_litros();
                            list_ext->eliminar_actual();
                            encontrado=true;
                      }

                    else  list_ext->avanzar();
                    }

                 if ((p_coche->get_gasolina()+contenido)>50)
                {
                        p_coche->set_gasolina(50);
                 }
                 else
                 {
                 p_coche->set_gasolina(p_coche->get_gasolina()+contenido);
                 gasoil=p_coche->get_gasolina();
                 }
               } //fin if list_ext no vacia

               if ((gasoil<=0) && (contenido==0))
                 {
                        gotoxy(1,2);
                        cout<<"            ";
                         gotoxy(1,2);
                        cout<<"GASOLINA: "<<p_coche->get_gasolina();
                        gotoxy(1,3);
                        cout<<"PUNTUACION: "<<t->get_puntos();
                        gotoxy(1,4);
                        cout<<"FALTA GASOLINA"<<endl;
                        c=27;
                   }
          }             //Fin if gasoil<20
   }                    //Fin if cuenta_seg%3

if (lista_seg->vacia())
  {
     gotoxy(1,1);
     cout<<"PUNTOS: "<<puntuacion;
     t->set_puntos(t->get_puntos()+puntuacion);
     gotoxy(1,3);
     cout<<"PUNTUACION: "<<t->get_puntos();
     gotoxy(1,2);
     cout<<"            ";
     gotoxy(1,2);
     cout<<"GASOLINA: "<<p_coche->get_gasolina();
     completado=true;
     c=27;                                          //fin de la carretera
  }
```

```
                } //fin if carretera>480

                start=clock();                          //tomamos el tiempo

        }  //Fin if han pasado 3 seg.

          if (kbhit()) c = getch();         //si se pulsa una tecla se captura
          if (c==27)                        //si se  pulsa escape
                {
                        break;
                }
  }                                         //Fin while true

  c=0;
  delete lista_seg;
  delete p_cola;

  return completado;
}

void juego::cargar_carrera(grafico *g1, grafo *gr, coche *p_coche, turista *turi, int j,
bool visitados[])
{
        int posi;
        char *kalle, *origen, *destino;

        origen=gr->get_p_origen();
        destino=gr->get_p_vn(j);
        g1->borramenu();
        if (!(gr->existe_arco(origen,destino)))
          {
                gotoxy (11,11);
                cout<<"No existe camino entre "<<origen<<" y "<<destino<<endl;
                getchar();
                g1->pintagrafo(gr);
          }
        else
          {
                posi=gr->posicion_origen();
                kalle=new char[strlen(gr->get_calle(posi,j))+5];
                strcpy(kalle,gr->get_calle(posi,j));
                strcat(kalle,".txt\0");
                if (carrera(kalle,gr,p_coche,turi))
                {
                        gotoxy (1,6);
                        cout<<"FINISH"<<endl;
                        gr->set_p_origen(j);
                        origen=gr->get_p_origen();
```

```
                        if (!(visitados[j]))
                                {
                                        visitados[j]=true;
                                        turi->set_monumento(gr->get_nodo(j));
                                }
                    }
                  else
                    {
                        gotoxy (1,5);
                        cout<<"GAME OVER"<<endl;
                        gotoxy (1,6);
                        cout<<"INSERT COIN        "<<endl;
                    }
                  getchar();
                  delete kalle;
        }
}

coche * juego::eligecoche(coche *pc_normal, coche *pc_rapido, turista *turi, bool
&creado_r)
{

  objeto *obj,*bonificacion;          //Punteros a objetos
  lista<objeto> *lista_b;             //Puntero a lista de bonificaciones
  cola<objeto> *cola_b;               //Puntero a una cola de bonificaciones

  cola_b=new cola<objeto>();          //cola de objetos.
  lista_b=new lista<objeto>();        //lista de objetos.

  if (turi->get_puntos()<1000)        return pc_normal;
                                      //Si puntos<1000 devuelve puntero a coche normal
  if ((turi->get_puntos()>=1000) && (!creado_r))
                                      //Si puntos>1000 y nunca se creo coche rapido
        {

                  lista_b=pc_normal->darextras();

                  lista_b->iniciar();
                  while (lista_b->noesfin())      //Recorremos la lista de bonificaciones
                  {
                        obj=lista_b->leer();
                        if (typeid(*(obj))==typeid(estatua))
                              bonificacion=new estatua(*((estatua *)obj));
                              //copiando objeto estatua
                        else if (typeid(*(obj))==typeid(vasija))
                              bonificacion=new vasija(*((vasija *)obj));
                              //copiando objeto estatua
                        cola_b->encolar(bonificacion);
                        lista_b->avanzar();
                  }
```

```
        lista_b=pc_rapido->darextras();       //Da la lista de bonificaciones
        while (!(cola_b->cola_vacia()))       //Mientras tenga bonificaciones la cola
        {
                obj=cola_b->primero();        //Sacamos el primer elemento de la cola
                if (typeid(*(obj))==typeid(estatua))
                                bonificacion=new estatua(*((estatua *)obj));
                                //copiando objeto estatua
                else if (typeid(*(obj))==typeid(vasija))
                                bonificacion=new vasija(*((vasija *)obj));
                                //copiando objeto vasija
                lista_b->insertar(bonificacion);
                cola_b->desencolar();
        }

      creado_r=true;
      return pc_rapido;                 //Devuelve puntero a coche rapido
    }

  if ((turi->get_puntos()>=1000) && (creado_r))   return pc_rapido;
}
```

Grafo.h

```
#ifndef grafo_h
#define grafo_h
#define MAX 6

struct celda
                {
                        public:
                                char* nombre_calle;
                                int   longitud;
                };

typedef char * T_vectornodos[MAX];
typedef celda T_matrizadyacencia[MAX][MAX];

class grafo
  {

    private:

        T_vectornodos VN;
        T_matrizadyacencia M;
        int vel_horiz,gasolina;
        char * p_origen;
```

```
    public:

        grafo();
        grafo(char * fich);
        bool vacio();
        bool nuevo_arco(char * m_origen, char * m_destino, char * calle, int distancia);
        bool borrar_arco(char * origen, char * destino);
        void mostrar_nodos();
        char * get_nodo(int i);
        void mostrar_arcos();
        void pintar_arcos();
        bool existe_arco (char * origen, char * destino);
        int get_vel_horiz();
        int get_gasolina();
        void set_p_origen(int i);
        char * get_p_origen();
        int posicion_origen();
        char * get_p_vn(int i);
        char * get_calle(int i, int j);
        ~grafo();

  };

# endif
```

Grafo.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <typeinfo.h>
#include <graphics.h>
#include <fstream.h>
#include "linea.h"
#include "grafo.h"

grafo::grafo()
{
        cout<<"Ha entrado en el contructor grafo sin parametrizar";
        getchar();
}
```

```
grafo::grafo(char * fich)
{
        //cout<<"Ha entrado en el contructor grafo parametrizado";
        //getchar();

        linea *lin=new linea();
        char * cadena;

        fstream fichero(fich,ios::in);          //Abrimos el fichero fich pasado,
                                                //como parametro, para lectura
        if (!fichero)
          {
                cout<<"Error al abrir el fichero.";
                getchar();
                exit(1);
          }

        if ((cadena=new char[80])==NULL)
          {
                cout<<"Error en la reserva de memoria.";
                getchar();
                exit(1);
          }

        cadena[0]='\0';

        int cont=0;
        int ejex,ejey;

        while ((!fichero.eof()) && (cont<8))        //mientras que no termine el fichero
                                                    //y contador<8
        {
                fichero>>cadena;                    //leemos las primeras 7 líneas
                lin->separar(cadena);               //separamos los campos
                if ((cont==0)||(cont==1))           //Las 2 primeras lineas ...
                  {
                        int campo=atoi(lin->obtener_campo(0));
                        //.. se transforman a enteros ..
                        if (cont==0)    vel_horiz=campo;
                        //.. se guarda la velocidad horizontal
                        if (cont==1)    gasolina=campo;
                        //.. se guarda la gasolina inicial
                  }
                else            //Llenado del vector de monumentos
                  {
                        VN[cont-2]=new char[strlen(lin->obtener_campo(0))+1];
                        //Se reserva memoria para la long. de cadena + 1 (fin de línea),
                        //que se guardará en el vector de nodos
                        strcpy(VN[cont-2],lin->obtener_campo(0));
```

```
                    VN[cont-2][strlen(lin->obtener_campo(0))]='\0';
                    //Insertamos el fin de linea en la última posición de la cadena
                  }
                cont++;
        }

ejex=0;
ejey=0;
while ((!fichero.eof()) && (ejex<MAX))
                                    //Recorremos ejex = líneas del fichero de texto
        {
                fichero>>cadena;            //Leemos la línea
                lin->separar(cadena);       //Separamos los campos

                while ((!fichero.eof()) && (ejey<2*MAX))
                //Recorremos ejey = campos de una línea (Calle1,coste1,Calle2,coste2,...)
                {
                        if (strcmp(lin->obtener_campo(ejey),"NULL")!=0)
                        //Comparamos si el campo CalleN, no es una calle vacía
                          {

        M[ejex][ejey/2].nombre_calle=new char[strlen(lin->obtener_campo(ejey))+1];
        //Reserva memoria para la cadena
        strcpy(M[ejex][ejey/2].nombre_calle,lin->obtener_campo(ejey));
        //Copiamos el campo CalleN en M(linea,campo/2) ya que las líneas se recorren
        de 2 en 2.
       M[ejex][ejey/2].nombre_calle[strlen(lin->obtener_campo(ejey))]='\0';
       //Insertamos fin de línea
       M[ejex][ejey/2].longitud=atoi(lin->obtener_campo(ejey+1));
       //Copiamos campo long. en M(línea,campo/2)

                          }
                        else                            //Si es una calle vacía
                          {

        M[ejex][ejey/2].nombre_calle=new char[strlen(lin->obtener_campo(ejey))+1];
        //Reservamos memoria
        strcpy(M[ejex][ejey/2].nombre_calle,"NULL");
        //Copiamos cadena NULL
       M[ejex][ejey/2].nombre_calle[strlen(lin->obtener_campo(ejey))]='\0';
        //Insertamos fin de línea
       M[ejex][ejey/2].longitud=-1;

                          }
        ejey=ejey+2;
                        //Incrementamos ejey en dos campos hasta la siguiente calle
                }       //(Calle1,coste1,Calle2,coste2,Calle3,...)
     ejey=0;  //Volvemos al primer campo
     ejex++;  //Pasamos a la siguiente línea
   }
```

```
  fichero.close();

  p_origen=VN[0];

  delete lin;
  delete cadena;
}

bool grafo::vacio()
{
        cout<<"Comprobando grafo ... "<<endl;
        int i=0;
        int j=0;
        while (i<MAX)
          {
                while (j<MAX)
                  {
                        if (strcmp(M[i][j].nombre_calle,"NULL")!=0)
                          {
                                cout<<" El grafo no esta vacio.";
                                return false;
                          }
                        j++;
                  }
                i++;
          }
        cout<<"Grafo vacio."<<endl;
        return true;
}

bool grafo::nuevo_arco(char * origen, char * destino, char * calle, int coste)
{
  bool encontrado_o=false;
  bool encontrado_d=false;
  bool arco=false;
  int i=0;
  int ejex,ejey;
  while (((!encontrado_o)&&(!encontrado_d))||(i<MAX))
    {
        if ((strcmp(VN[i],origen)==0)&&(!encontrado_o))
          {
                ejex=i;
                encontrado_o=true;
          }
        if ((strcmp(VN[i],destino)==0)&&(!encontrado_d))
          {
                ejey=i;
                encontrado_d=true;
          }
```

```
        if ((encontrado_o)&&(encontrado_d))
          {
        cout<<"Se ha creado un nuevo arco entre "<<origen<<" y "<<destino<<endl;
        M[ejex][ejey].nombre_calle=new char[strlen(calle)+1];   //Reserva memoria cad
        strcpy(M[ejex][ejey].nombre_calle,calle);
        M[ejex][ejey].nombre_calle[strlen(calle)]='\0';         //Insertamos fin de línea
        M[ejex][ejey].longitud=coste;
        getchar();
        arco=true;
        return arco;
          }
        i++;
    }
  cout<<"No se ha creado el arco ..."<<endl;
  return arco;
}

bool grafo::borrar_arco(char * origen, char * destino)
{
  bool encontrado_o=false;
  bool encontrado_d=false;
  bool borrado=false;
  int i=0;
  int ejex,ejey;
  while (((!encontrado_o)&&(!encontrado_d))||(i<MAX))
  {
        if ((strcmp(VN[i],origen)==0)&&(!encontrado_o))
        {
                ejex=i;
                encontrado_o=true;
        }
    if ((strcmp(VN[i],destino)==0)&&(!encontrado_d))
        {
                ejey=i;
                encontrado_d=true;
        }
    if ((encontrado_o)&&(encontrado_d))
        {
                cout<<"Arco borrado ..."<<endl;
                strcpy(M[ejex][ejey].nombre_calle,"NULL");
                M[ejex][ejey].longitud=-1;
                getchar();
                borrado=true;
                return borrado;
        }
    i++;
  }
  cout<<"Arco no encontrado ..."<<endl;
  return borrado;
}
```

```
void grafo::pintar_arcos()
{
  //cout<<"Ha entrado en pintar arcos...";
  struct coordenadas
        {
                int x;
                int y;
        };
  coordenadas v_centros[6];          //Definición de un vector de coordenadas constante

  v_centros[0].x=105;
  v_centros[0].y=225;
  v_centros[1].x=235;
  v_centros[1].y=125;
  v_centros[2].x=405;
  v_centros[2].y=125;
  v_centros[3].x=535;
  v_centros[3].y=225;
  v_centros[4].x=405;
  v_centros[4].y=325;
  v_centros[5].x=235;
  v_centros[5].y=325;

  int i,j,x1,y1,x2,y2;

  for (i=0;i<MAX;i++)
        {
                for (j=0;j<MAX;j++)
                {
                        if (existe_arco(VN[i],VN[j]))
                          {
                                x1=v_centros[i].x;
                                y1=v_centros[i].y;
                                x2=v_centros[j].x;
                                y2=v_centros[j].y;
                                line(x1,y1,x2,y2);  //Pinta el arco entre (x1,y1) y (x2,y2)
                          }
                }
        }
}

void grafo::mostrar_arcos()
{
        int ejex,ejey;
        for (ejex=0;ejex<MAX;ejex++)
        for (ejey=0;ejey<MAX;ejey++)
                cout<<"Calle: "<<M[ejex][ejey].nombre_calle;
                cout<<" Distancia: "<<M[ejex][ejey].longitud<<endl;
}
```

```
void grafo::mostrar_nodos()
{
        for (int i=0;i<MAX;i++)
        cout<<"Monumento: "<<VN[i]<<endl;
}

char * grafo::get_nodo(int i)
{
        return VN[i];
}

bool grafo::existe_arco(char * origen, char * destino)
{
  bool encontrado_o=false;
  bool encontrado_d=false;
  bool arco=false;
  int i=0;
  int ejex=0;
  int ejey=0;
  while (((!encontrado_o)&&(!encontrado_d))||(i<MAX))
  {
        if ((strcmp(VN[i],origen)==0)&&(!encontrado_o))
          {
                ejex=i;
                encontrado_o=true;
          }
        if ((strcmp(VN[i],destino)==0)&&(!encontrado_d))
          {
                ejey=i;
                encontrado_d=true;
          }
        if ((encontrado_o)&&(encontrado_d))
          {
                if (strcmp(M[ejex][ejey].nombre_calle,"NULL")!=0)
                  {
                        arco=true;
                        return arco;
                  }
          }
        i++;
  }
  //cout<<"No existe arco alguno entre "<<origen<<" y "<<destino<<endl;
  //getchar();
  return arco;
}
```

```
int grafo::posicion_origen()
{
        for (int i=0; i<MAX; i++)
                if (strcmp(VN[i],p_origen)==0)        return i;
}

int grafo::get_vel_horiz()
{
        return vel_horiz;
}

int grafo::get_gasolina()
{
        return gasolina;
}

char * grafo::get_p_origen()
{
        return p_origen;
}

void grafo::set_p_origen(int i)
{
        p_origen=VN[i];
}

char * grafo::get_p_vn(int i)
{
  return VN[i];
}

char * grafo::get_calle(int i, int j)
{
        return M[i][j].nombre_calle;
}

grafo::~grafo()
{
  cout<<"Ha entrado en el destructor grafo";
  getchar();

  for (int ejex=0;ejex<MAX;ejex++)
        for (int ejey=0;ejey<MAX;ejey++)
          {
                delete M[ejex][ejey].nombre_calle;
          }

  for (int ejex=0;ejex<MAX;ejex++)          delete VN[ejex];
}
```

Linea.h

```
//linea.h se utiliza para ir leyendo las líneas de texto de los ficheros
//y separarlas por campos, usando de separacion la almohadilla #

#ifndef linea_h
#define linea_h

#include <string.h>

class linea
{
	char *param[12];  //12 campos de 30 caracteres cada uno

	public:
		linea();
		void parametro(char *L, int n);		//para obtener los parámetros
		char *obtener_campo(int n);		//obtiene un campo de la línea
		void separar (char *L);			//separa los campos
};

#endif
```

Linea.cpp

```
#include "linea.h"

linea::linea()
{
	int contador;
	for(contador=0; contador<12; contador++)
		{
			param[contador]=new char[30];
		}
}

void linea::parametro(char *L, int n)			//para obtener los parámetros
{
	int i,j,tamano;
	for(i=0; i < strlen(L) && L[i] != '#'; i++)
							param[n][i]=L[i];
	param[n][i]='\0';
	tamano=strlen(L);
	for(j=0; j < tamano - i; j++)			//machacar las primeras posiciones
					L[j]=L[j+i+1];
	L[j]='\0';

}
```

```
char *linea::obtener_campo(int n)
{
        return (param[n]);
}

void linea::separar (char *L)      //separa los campos
{
        parametro(L, 0);
        parametro(L, 1);
        parametro(L, 2);
        parametro(L, 3);
        parametro(L, 4);
        parametro(L, 5);
        parametro(L, 6);
        parametro(L, 7);
        parametro(L, 8);
        parametro(L, 9);
        parametro(L, 10);
        parametro(L, 11);

}
```

Gráfico.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include "grafo.h"
#ifndef grafico_h
#define grafico_h

class grafico
{
  private:

        int    GraphDriver;          /* Variable para almacenar driver gráfico          */
        int    GraphMode;            /* Variable para almacenar valor del modo          */
        double AspectRatio;          /* Aspecto de ratio de un pixel sobre la pantalla  */
        int    MaxX, MaxY;           /* Máxima resolución de la pantalla                */
        int    MaxColors;            /* Máximo conjunto de colores posibles             */
        int    ErrorCode;            /* Variable para almacenar errores                 */
        struct palettetype palette;  /* Estructura para la información de la paleta      */
        int col_raton;               /* Variable para almacenar la columna del ratón    */
        int fil_raton;               /* Variable para almacenar la columna del ratón    */
```

```
        public:

                grafico();
                ~grafico();
                void Inicializacion_modo_grafico(void);
                int IniciarRaton(void);
                void MostrarRaton(void);
                void OcultarRaton(void);
                int BotonesRaton(void);
                int PosXRaton(void);
                int PosYRaton(void);
                void tratar_raton(void);
                int esperarpulsacion();
                void borramenu();
                void pintainstr();
                int esperarseleccion();
                void pintamenu();
                void pintagrafo(grafo *gr2);

};

#endif
```

Gráfico.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "grafo.h"
#include "grafico.h"


grafico::grafico()
{
  cout<<"Ha entrado en el constructor grafico"<<endl;
  Inicializacion_modo_grafico();
}
```

```
grafico::~grafico()
{
        cout<<"Ha entrado en el destructor grafico"<<endl;
}

void grafico::Inicializacion_modo_grafico(void)
{
  int xasp, yasp;                          /* Variables para almacenar el aspecto*/

  GraphDriver = DETECT;                    /* Detección del modo gráfico        */
  initgraph( &GraphDriver, &GraphMode, "" );
  ErrorCode = graphresult();               /* Resultado de la inicializacion    */
  if( ErrorCode != grOk )                  /* Posible error en inicializacion   */
        {
   printf(" Error al iniciliacizar el modo gráfico: %s\n", grapherrormsg( ErrorCode ) );
   exit( 1 );
        }

  getpalette( &palette );             /* Lectura de la paleta de colores    */
  MaxColors = getmaxcolor() + 1;       /* Máximo número de coloares posibles */

  MaxX = getmaxx();
  MaxY = getmaxy();                    /* Lectura del tamaño de la pantalla  */

  getaspectratio( &xasp, &yasp );      /* Lectrua del aspecto del hardware   */
  AspectRatio = (double)xasp / (double)yasp; /*Obtección de corrección del factor */

}

/* Inicializacion raton */
int grafico::IniciarRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x00;
  intr(0x33,&Registros);
  if (Registros.r_ax==0xffff)
     return (1);  /*Si Existe  */
  else
     return(0);   /*No existe */
}

/* Visualiza el raton en pantalla */

void grafico::MostrarRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x01;
  intr(0x33,&Registros);
}
```

```
/* Oculta el raton */

void grafico::OcultarRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x02;
  intr(0x33,&Registros);
}

/* Devuelve una pulsación del ratón */

int grafico::BotonesRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x03;
  intr(0x33,&Registros);
  return(Registros.r_bx);
}

/* Devuelve la posicion X de pulsacion del raton */

int grafico::PosXRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x03;
  intr(0x33,&Registros);
  return(Registros.r_cx);
}

/* Devuelve la posicion Y de pulsacion del raton */

int grafico::PosYRaton(void)
{
  struct REGPACK Registros;
  Registros.r_ax=0x03;
  intr(0x33,&Registros);
  return(Registros.r_dx);
}
```

```
/* Analisis de pulsaciones */

void grafico::tratar_raton(void)
{
 int pulsado;                                /* Esperar para limpieza pulsacion antigua */
 pulsado=1;
 while (pulsado!=0) pulsado=BotonesRaton();
                                                          /* Captura de nueva pulsación */
 pulsado=0;
 while (pulsado==0)
          if (BotonesRaton())
            {
                  col_raton=PosXRaton();
                  fil_raton=PosYRaton();
                  pulsado=1;
            }
}


/* Analisis de zona de la pantalla pulsada */

int grafico::esperarpulsacion()
{
 int validacion;
 validacion=0;

 while (validacion==0)
   {
          tratar_raton();
          if ((col_raton>=70)&&(col_raton<=140))
            {
                  if ((fil_raton>=200)&&(fil_raton<=250)) validacion=1;
            }

         if ((col_raton>=200)&&(col_raton<=270))
         {
            if ((fil_raton>=100)&&(fil_raton<=150)) validacion=2;
                  if ((fil_raton>=300)&&(fil_raton<=350)) validacion=6;
         }

         if ((col_raton>=370)&&(col_raton<=440))
         {
            if ((fil_raton>=100)&&(fil_raton<=150)) validacion=3;
                  if ((fil_raton>=300)&&(fil_raton<=350)) validacion=5;
         }

         if ((col_raton>=500)&&(col_raton<=570))
         {
            if ((fil_raton>=200)&&(fil_raton<=250)) validacion=4;
         }
```

```
   /* Botón salir */
          if ((col_raton>=270)&&(col_raton<=370))
            if ((fil_raton>=420)&&(fil_raton<=470)) validacion=7;

          if (validacion==0)
            {
             sound(300);
             delay(100);
             nosound();
            }
          }
  return(validacion);
}

void grafico::borramenu()
{
          OcultarRaton();

          setfillstyle(SOLID_FILL,BLACK);
          bar(0,0,640,480);
}

void grafico::pintainstr()
{
  OcultarRaton();

          setfillstyle(SOLID_FILL,BLACK);
          setcolor(WHITE);

          bar(0,0,640,480);

          setfillstyle(SOLID_FILL,GREEN);

          /* Cuadros del menú */
          bar(50,50,590,400);
          bar(50,10,590,45);

          setfillstyle(SOLID_FILL,RED);

          /* Tipo de letra utilizada */
          settextstyle(1, HORIZ_DIR, 3);

          bar(51,11,589,44);
          outtextxy(240,10,"Instrucciones");

          settextstyle(7, HORIZ_DIR, 1);
          outtextxy(70,60,"Bienvenido al juego, al seleccionear jugar se pedira ");
          outtextxy(70,80,"su nombre y nacionalidad, para pasar de pantalla ");
          outtextxy(70,100,"y elegir con el raton el monumento que desea visitar.");
```

```
        outtextxy(70,120,"Durante la carrera debe esquivar los obstaculos y ");
        outtextxy(70,140,"coger las bonificaciones, representadas asi: ");

        figura * r;
        r=new rectangulo(70,175,4,85,190);
        r->pintar();
        delete r;

        setcolor(WHITE);
        outtextxy(90,170,"= bache, pierdes una rueda si no tienes repuesto");

        figura * re;
        re=new rectangulo(70,195,5,100,210);
        re->pintar();
        delete re;

        setcolor(WHITE);
        outtextxy(105,190,"= accidente, inmoviliza el vehiculo un tiempo");

        figura * c;
        c=new circulo(80,223,8,8);
        c->pintar();
        delete c;

        setcolor(WHITE);
        outtextxy(95,210,"= estatua, rueda de repuesto en caso de baches");

        figura * v;
        v=new elipse(85,243,3,120,60,15,8);

        v->pintar();
        delete v;

        setcolor(WHITE);
        outtextxy(105,230,"= vasija, rellena el deposito de gasolina");

        outtextxy(70,260,"Si consigue superar 1000 puntos durante el juego");
        outtextxy(70,280,"sera recompensado con un coche mas rapido, el cual");
        outtextxy(70,300,"puede desplazarse horizontal y verticalmente.");
        outtextxy(70,320,"Para desplazar el coche debe usar las teclas de ");
        outtextxy(70,340,"de direccion del teclado (flechas).");
        outtextxy(70,360,"Puede avandonar la partida pulsando la tecla Esc.");

        setcolor(RED);
        outtextxy(250,420,"Pulse ENTER para continuar -->");
        getchar();

        borramenu();

}
```

```
int grafico::esperarseleccion()
{
  int seleccion;
  seleccion=0;

  while (seleccion==0)
          {
           tratar_raton();
           if ((col_raton>=200)&&(col_raton<=440))
           {
              if ((fil_raton>=100)&&(fil_raton<=150)) seleccion=1;
              if ((fil_raton>=200)&&(fil_raton<=250)) seleccion=2;
              if ((fil_raton>=300)&&(fil_raton<=350)) seleccion=3;
           }

          /* Botón salir */
          if ((col_raton>=270)&&(col_raton<=370))
              if ((fil_raton>=420)&&(fil_raton<=470)) seleccion=4;

          if (seleccion==0)
              {
                    sound(300);
                    delay(100);
                    nosound();
              }
          }
   return(seleccion);
}

void grafico::pintamenu()
{
  OcultarRaton();

  setfillstyle(SOLID_FILL,BLACK);
  setcolor(WHITE);

  bar(0,0,640,480);

  setfillstyle(SOLID_FILL,GREEN);

  /* Cuadros del menú */
  bar(50,50,590,400);
  bar(50,10,590,45);

  setfillstyle(SOLID_FILL,RED);

  /* fila 1 de botones */
  bar(200,100,440,150);
```

```
  /* fila 2 de botones */
  bar(200,200,440,250);

  /* fila 3 de botones */
  bar(200,300,440,350);

  /*Botón salir */
  bar(285,420,355,470);

  /* Tipo de letra utilizada */
  settextstyle(7, HORIZ_DIR, 1);

  /*texto fila de botones 1 */
  outtextxy(230,110,"Instrucciones de uso");

  /*texto fila de botones 2 */
  outtextxy(233,210,"Jugar una partida");

  /*texto fila de botones 3 */
  outtextxy(223,310,"Busqueda de turistas");

  /*texto botón salir*/
  outtextxy(300,430,"Salir");

  bar(51,11,589,44);
  outtextxy(60,13,"Seleccione la accion que desea realizar: ");

  /*Inicialización del ratón*/
  IniciarRaton();
  MostrarRaton();

 }

void grafico::pintagrafo(grafo *gr2)
 {
  char *toyaki;
  toyaki=new char[strlen(gr2->get_p_origen())];
  strcpy(toyaki,gr2->get_p_origen());

  OcultarRaton();

  setfillstyle(SOLID_FILL,BLACK);
  setcolor(WHITE);

  bar(0,0,640,480);

  setfillstyle(SOLID_FILL,GREEN);

  /* Cuadros del menú */
  bar(50,50,590,400);
```

```
  bar(50,10,590,45);

  gr2->pintar_arcos();

  setfillstyle(SOLID_FILL,RED);

  /* fila 1 de botones */
  bar(200,100,270,150);
  bar(370,100,440,150);

  /* fila 2 de botones */
  bar(70,200,140,250);
  bar(500,200,570,250);

  /* fila 3 de botones */
  bar(200,300,270,350);
  bar(370,300,440,350);

  /*Botón salir */
  bar(285,420,355,470);

  /* Tipo de letra utilizada */
  settextstyle(7, HORIZ_DIR, 1);

  /*texto fila de botones 1 */
  outtextxy(205,110,"Alkzba");
  outtextxy(375,110,"Colum");

  /*texto fila de botones 2 */
  outtextxy(75,210,"O. Tur.");
  outtextxy(505,210,"Teatro");

  /*texto fila de botones 3 */
  outtextxy(205,310,"Diana");
  outtextxy(375,310,"Acue.M");

  /*texto botón salir*/
  outtextxy(300,430,"Salir");

  bar(51,11,589,44);
  outtextxy(60,15,toyaki);

  /*Inicialización del ratón*/
  IniciarRaton();
  MostrarRaton();

}
```

Figura.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#ifndef figura_h
#define figura_h

class figura
{
        protected:

        int startx; //coordenada x
        int starty; //coordenada y
        int color;      //color de relleno

  public:

        figura();                                   //constructor sin parametrizar
        figura(int startx,int starty, int color);   //constructor parametrizado
        figura(const figura&original);
        virtual ~figura();                          // destructor
        int get_x();                                // muestra startx
        void set_x(int startx);                     // inicializa startx
        int get_y();                                // muestra starty
        void set_y(int starty);                     // inicializa starty
        int get_color();                            // muestra color
        void set_color(int color);                  // inicializa color
        virtual void pintar()=0;                    // pinta una figura
        virtual void borrar()=0;                    // borra una figura

};

#endif
```

Figura.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"

figura::figura()
{
        cout<<"Ha entrado en el contructor figura sin parametrizar";
        getchar();
}

figura::figura(int startx, int starty, int color)
{
        this->startx=startx;
        this->starty=starty;
        this->color=color;
        //cout<<"Ha entrado en el contructor figura parametrizado";
}

figura::figura(const figura&original)
{
        //cout<<"Ha entrado en el contructor copia figura";
        //getchar();
        startx=original.startx;
        starty=original.starty;
        color=original.color;
}

figura::~figura()
{
        //cout<<"Ha entrado en el destructor figura";
        //getchar();
}

int figura::get_x()
{
        return startx;
}

void figura::set_x(int startx)
{
        this->startx=startx;
}
```

```
int figura::get_y()
{
        return starty;
}
void figura::set_y(int starty)
{
        this->starty=starty;
}
int figura::get_color()
{
        return color;
}
void figura::set_color(int color)
{
        this->color=color;
}
```

Rectángulo.h

```
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#ifndef rectangulo_h
#define rectangulo_h

class rectangulo:public figura
{
        private:
        int endx;
        int endy;

  public:
        rectangulo();
        rectangulo(int startx, int starty, int color, int endx, int endy);
        rectangulo(const rectangulo&original);
        void set_endx(int endx);
        void set_endy(int endy);
        int get_endx();
        int get_endy();
        void pintar();
        void borrar();
    virtual ~rectangulo();
};
#endif
```

Rectángulo.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"

rectangulo::rectangulo()
{
  cout<<"Constructor rectangulo sin parametrizar";
  getchar();
}

rectangulo::rectangulo(int startx,int starty, int color, int endx, int
endy):figura(startx,starty,color)
{
        this->endx=endx;
        this->endy=endy;
        //cout<<"Constructor parametrizado de rectangulo";
        //getchar();
}

rectangulo::rectangulo(const rectangulo&original):figura(original)
{
        //cout<<"Ha entrado en el contructor copia rectangulo ...";
        endx=original.endx;
        endy=original.endy;
}

rectangulo::~rectangulo()
{
        //cout<<"Ha entrado en el destructor rectangulo";
        //getchar();
}

int rectangulo::get_endx()
{
        return endx;
}

int rectangulo::get_endy()
{
        return endy;
}
```

```
void rectangulo::set_endx(int endx)
{
        this->endx=endx;
}

void rectangulo::set_endy(int endy)
{
        this->endy=endy;
}

void rectangulo::pintar()
{
        setcolor(color);
        rectangle(startx,starty,endx,endy);
}

void rectangulo::borrar()
{
        setcolor(0);
        rectangle(startx,starty,endx,endy);
}
```

Elipse.h

```
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#ifndef elipse_h
#define elipse_h

class elipse:public figura
{
        private:
                int startang;
                int endang;
                int radio_big;
                int radio_small;

        public:
                elipse();
                elipse(int startx, int starty, int color, int startang, int endang, int
                radio_small, int radio_big);
```

```
                elipse(const elipse&original);
                void set_radio_big(int radio_big);
                int get_radio_big();
                void set_radio_small(int radio_small);
                int get_radio_small();
                void pintar();
                void borrar();
                ~elipse();
};
#endif
```

Elipse.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "elipse.h"

elipse::elipse()
{
        cout<<"Ha entrado en el constructor elipse sin parametros"<<endl;
        getchar();
}

elipse::elipse(int startx, int starty, int color, int startang, int endang, int radio_small, int
radio_big):figura(startx,starty,color)
{
        this->startang=startang;
        this->endang=endang;
        this->radio_big=radio_big;
        this->radio_small=radio_small;
        //cout<<"Constructor parametrizado de elipse";
        //getchar();
}

elipse::~elipse()
{
        //cout<<"Ha entrado en el destructor de elipse";
        //getchar();
}
```

```
void elipse::pintar()
{
	setcolor(color);
	ellipse(startx,starty,startang,endang,radio_small,radio_big);
}

void elipse::borrar()
{
	setcolor(0);
	ellipse(startx,starty,startang,endang,radio_small,radio_big);
}

elipse::elipse(const elipse&original):figura(original)
{
	//cout<<"Ha entrado en el contructor copia de elipse ...";
	//getchar();
	startang=original.startang;
	endang=original.endang;
	radio_big=original.radio_big;
	radio_small=original.radio_small;
}

int elipse::get_radio_big()
{
	return radio_big;
}

void elipse::set_radio_big(int radio_big)
{
	this->radio_big=radio_big;
}

int elipse::get_radio_small()
{
	return radio_small;
}

void elipse::set_radio_small(int radio_small)
{
	this->radio_small=radio_small;
}
```

Círculo.h

```
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#ifndef circulo_h
#define circulo_h

class circulo:public figura
{
        private:
        int radio;

  public:
        circulo();
        circulo(int startx, int starty, int color, int radio);
        circulo(const circulo&original);
        void set_radio(int radio);
        int get_radio();
        void pintar();
        void borrar();
        ~circulo();
};
#endif
```

Círculo.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "circulo.h"

circulo::circulo()
{
        cout<<"Constructor circulo sin parametros"<<endl;
        getchar();
}
```

```
circulo::circulo(int startx,int starty, int color, int radio):figura(startx,starty,color)
{
	this->radio=radio;
	//cout<<"Constructor parametrizado de circulo";
	//getchar();
}

circulo::~circulo()
{
	//cout<<"Ha entrado en el destructor de circulo";
	//getchar();
}

void circulo::pintar()
{
	setcolor(color);
	circle(startx,starty,radio);
}

void circulo::borrar()
{
	setcolor(0);
	circle(startx,starty,radio);
}

circulo::circulo(const circulo&original):figura(original)
{
	//cout<<"Ha entrado en el contructor copia de circulo ...";
	//getchar();
	radio=original.radio;
}

int circulo::get_radio()
{
	return radio;
}

void circulo::set_radio(int radio)
{
	this->radio=radio;
}
```

Objeto.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#ifndef objeto_h
#define objeto_h

class objeto
{
        protected:

        figura *forma;              //forma del objeto
        int x;                      //coordenada x
        int y;                      //coordenada y
        int x1,y1,x2,y2;            //Coordenadas del rectángulo contenedor del objeto

  public:

        objeto(int x, int y);               //Constructor sin parametros
        objeto(const objeto&original);      //Constructor copia
        figura * darforma();                //Devuelve la forma del objeto para poder
        modificar la figura
        void calcular(figura * fig);        //Calcula coordenadas del rectangulo
        int get_x1();                       //Obtiene las coordenadas calculadas
        int get_y1();
        int get_x2();
        int get_y2();
        virtual void pintar()=0;                    //Pintar objeto
        virtual void borrar()=0;                    //Borrar objeto
        virtual ~objeto();                          //Destructor

};

#endif
```

Objeto.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include <typeinfo.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"

objeto::objeto(int x, int y)
{
        this->x=x;
        this->y=y;
}

figura* objeto::darforma()
{
        return forma;
}

void objeto::calcular(figura * fig)
{
        int rad_small, rad_big;
        //cout<<"Has entrado en calcular rectangulo de figura..."<<endl;
        if (typeid(*(fig))==typeid(circulo))
          {
                rad_small=((circulo *)fig)->get_radio();
                x1=x-rad_small;
                y1=((((circulo *)fig)->get_y())-rad_small);
                x2=x+rad_small;
                y2=((((circulo *)fig)->get_y())+rad_small);
          }

        if (typeid(*(fig))==typeid(elipse))
          {
                rad_small=((elipse *)fig)->get_radio_small();
                rad_big=((elipse *)fig)->get_radio_big();
                x1=x-rad_small;
                y1=(((elipse *)fig)->get_y()-rad_big);
                x2=x+rad_small;
                y2=(((circulo *)fig)->get_y()+rad_big);
          }
```

```
        if (typeid(*(fig))==typeid(rectangulo))
          {
                x1=fig->get_x();
                y1=fig->get_y();
                x2=((rectangulo *)fig)->get_endx();
                y2=((rectangulo *)fig)->get_endy();
          }
}

int objeto::get_x1()
{
        return x1;
}

int objeto::get_y1()
{
        return y1;
}

int objeto::get_x2()
{
        return x2;
}

int objeto::get_y2()
{
        return y2;
}

objeto::objeto(const objeto&original)
{
        //cout<<"Ha entrado en el contructor copia de objeto ...";
        //getchar();
        x=original.x;
        y=original.y;
}

objeto::~objeto()
{
        //cout<<"Ha entrado en el destructor objeto";
        //getchar();
}
```

Bonificación.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "objeto.h"
#ifndef bonificacion_h
#define bonificacion_h

class bonificacion:public objeto
{

  protected:

        int puntos;                     //puntos

  public:

        bonificacion(int x, int y, int puntos);          //constructor sin parametrizar
        bonificacion(const bonificacion&original);       //constructor copia
        void set_puntos(int puntos);                     //Modifica los puntos
        int get_puntos();                                //Obtiene los puntos
        virtual void pintar()=0;                         //pintar bonificacion
        virtual void borrar()=0;                         //borrar bonificacion
        virtual ~bonificacion();                         //destructor

};

#endif
```

Bonificación.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "objeto.h"
#include "bonificacion.h"
```

```
bonificacion::bonificacion(int x, int y, int puntos):objeto(x,y)
{
	this->puntos=puntos;
}

void bonificacion::set_puntos(int puntos)
{
	this->puntos=puntos;
}

int bonificacion::get_puntos()
{
	return puntos;
}

bonificacion::bonificacion(const bonificacion&original):objeto(original)
{
	//cout<<"Ha entrado en el constructor copia de bonificacion ...";
	//getchar();
	this->puntos=original.puntos;
}

bonificacion::~bonificacion()
{
	//cout<<"Ha entrado en el destructor bonificacion";
	//getchar();
}
```

Vasija.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "objeto.h"
#include "bonificacion.h"
#ifndef vasija_h
#define vasija_h
```

```
class vasija:public bonificacion
{
	private:
			int litros;	//litros de gasolina

	public:

			vasija(int x, int y, int puntos, int litros);	//constructor vasija
			vasija(const vasija&original);			//constructor copia
			int get_litros();	//litros de gasolina que contiene la vasija
			void pintar();					//pintar vasija
			void borrar();					//borrar vasija
			~vasija();					//destructor vasija
};

#endif
```

Vasija.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "bonificacion.h"
#include "vasija.h"

vasija::vasija(int x, int y, int puntos, int litros):bonificacion(x,y,puntos)
{
	this->litros=litros;
	forma=new elipse(x,y,15,120,60,18,10);
}

int vasija::get_litros()
{
	return litros;
}
```

```
vasija::vasija(const vasija&original):bonificacion(original)
{
        //cout<<"Ha entrado en el constructor copia de vasija ...";
        //getchar();
        this->litros=original.litros;
        forma = new elipse(*((elipse *)original.forma));
}

void vasija::pintar()
{
  forma->pintar();
}

void vasija::borrar()
{
        //cout<<"Ha entrado en borrar vasija";
        forma->borrar();
}

vasija::~vasija()
{
        //cout<<"Ha entrado en el destructor de vasija";
        //getchar();
        delete forma;
}
```

Estatua.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "objeto.h"
#include "bonificacion.h"
#ifndef estatua_h
#define estatua_h

class estatua:public bonificacion
{
        private:
                        figura *rueda;          //puntero a rueda
```

```
        public:

                estatua(int x, int y, int puntos);      //constructor estatua
                estatua(const estatua&original);        //constructor copia
                void pintar();                          //pintar estatua
                void borrar();                          //borrar estatua
                ~estatua();                             //destructor estatua

};
#endif
```

Estatua.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"

estatua::estatua(int x, int y, int puntos):bonificacion(x,y,puntos)
{
        forma=new circulo(x,y,8,8);
}

estatua::estatua(const estatua&original):bonificacion(original)
{
        forma = new circulo(*((circulo *)original.forma));
}

void estatua::pintar()
{
  forma->pintar();
}

void estatua::borrar()
{
  forma->borrar();
}
```

```
estatua::~estatua()
{
        //cout<<"Ha entrado en el destructor estatua";
        //getchar();
        delete forma;
}
```

Bache.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "objeto.h"
#ifndef bache_h
#define bache_h

class bache:public objeto
{

  public:

        bache(int x, int y);                    //constructor
        bache(const bache&original);            //constructor copia
        void pintar();                          //pintar bache
        void borrar();                          //borrar bache
        ~bache();                               //destructor

};

#endif
```

Bache.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "bache.h"

bache::bache(int x, int y):objeto(x,y)
{
        forma=new rectangulo(x-5,y-5,4,x+5,y+5);
}

bache::bache(const bache&original):objeto(original)
{
        forma = new rectangulo(*((rectangulo *)original.forma));
}

void bache::pintar()
{
  forma->pintar();
}

void bache::borrar()
{
  forma->borrar();
}

bache::~bache()
{
  delete forma;
}
```

Accidente.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "objeto.h"
#ifndef accidente_h
#define accidente_h
```

```
class accidente:public objeto
{
	private:
			int tiempo;		//Tiempo de penalización

	public:

		accidente(int x, int y, int tiempo);	//constructor accidente
		accidente(const accidente&original);//constructor copia
		int get_tiempo();		//Da el tiempo de parada del accidente
		void pintar();			//pintar accidente
		void borrar();			//borrar accidente
		~accidente();			//destructor accidente

};

#endif
```

Accidente.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "accidente.h"

accidente::accidente(int x, int y, int tiempo):objeto(x,y)
{
	this->tiempo=tiempo;
	forma=new rectangulo(x-15,y-5,5,x+15,y+5);
}

accidente::accidente(const accidente&original):objeto(original)
{
	//cout<<"Ha entrado en el constructor copia de accidente ...";
	//getchar();
	this->tiempo=original.tiempo;
	forma = new rectangulo(*((rectangulo *)original.forma));
}
```

```
int accidente::get_tiempo()
{
	return tiempo;
}

void accidente::pintar()
{
	forma->pintar();
}

void accidente::borrar()
{
	//cout<<"Ha entrado en borrar accidente";
	forma->borrar();
}

accidente::~accidente()
{
	//cout<<"Ha entrado en el destructor de accidente";
	//getchar();
	delete forma;
}
```

Coche.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "lista.h"
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"
#include "vasija.h"
#include "bache.h"
#include "accidente.h"
#ifndef coche_h
#define coche_h
```

```
class coche
{
        protected:

                int x;              //coordenada x
                int y;              //coordenada y
                int x3,y3,x4,y4; //coordenadas del rectángulo q bordea la carroceria
                figura *v_figuras[7];  //vector de 7 figuras 4 ruedas, 2 faros, 1 carroceria
                int vel_horiz;           //velocidad horizontal
                int gasolina;            //gasolina
                lista<objeto> *extras; //lista de bonificaciones recogidas

        public:

                coche(int x,int y);              //Constructor con parametros
                lista<objeto> * darextras();   //Da la lista de bonificaciones del coche
                figura * darcarroceria();        //Obtiene la figura de la carroceria v[6]
                void calcularc(figura * figu);      //Calcula las coordenadas del rectángulo
                void set_gasolina(int gasolina);    //Inicializa el atributo gasolina
                int get_gasolina();              //Obtiene la gasolina que tiene el coche
                void set_vel_horiz(int vel_horiz);  //Inicializa la velocidad horizontal
                int get_vel_horiz();             //Obtiene la velocidad horizontal
                int get_x3();            //Obtiene las coordenadas del rectangulo contenedor
                int get_y3();
                int get_x4();
                int get_y4();
                virtual void pintar()=0;                //pintar coche
                virtual void borrar()=0;                //borrar coche
                virtual void mover(int tecla)=0;        //mover coche
                virtual ~coche();                       //destructor

};

#endif
```

Coche.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include <typeinfo.h>
#include "coche.h"
```

```
coche::coche(int x,int y)
{
	this->x=x;
	this->y=y;
	extras=new lista<objeto>();
	gasolina=20;
	//cout<<"Ha entrado en el contructor coche parametrizado";
	//getchar();
}

lista<objeto> * coche::darextras()
{
	lista<objeto> *p_list;
	p_list=extras;
	return p_list;
}

figura * coche::darcarroceria()
{
	//cout<<"obtenemos la carroceria del coche"<<endl;
	return v_figuras[6];
}

void coche::calcularc(figura * figu)
{
	int rad_small, rad_big;
	//cout<<"Calcular rectangulo de la figura"<<endl;
	if (typeid(*(figu))==typeid(elipse))
		{
			rad_small=((elipse *)figu)->get_radio_small();
			rad_big=((elipse *)figu)->get_radio_big();
			x3=x-rad_small;
			y3=y-rad_big;
			x4=x+rad_small;
			y4=y+rad_big;
		}

	if (typeid(*(figu))==typeid(rectangulo))
		{
			x3=figu->get_x();
			y3=figu->get_y();
			x4=((rectangulo *)figu)->get_endx();
			y4=((rectangulo *)figu)->get_endy();
		}
}

void coche::set_gasolina(int gasolina)
{
	this->gasolina=gasolina;
}
```

```
void coche::set_vel_horiz(int vel_horiz)
{
	this->vel_horiz=vel_horiz;
}

int coche::get_vel_horiz()
{
	return vel_horiz;
}

int coche::get_gasolina()
{
	return gasolina;
}

int coche::get_x3()
{
	return x3;
}

int coche::get_y3()
{
	return y3;
}

int coche::get_x4()
{
	return x4;
}

int coche::get_y4()
{
	return y4;
}

coche::~coche()
{
	//cout<<"Ha entrado en el destructor coche";
	//getchar();
	extras->iniciar();
	while (extras->vacia())
		{
			delete extras;
		}
}
```

Normal.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "coche.h"
#ifndef normal_h
#define normal_h

class normal:public coche
{
        private:
                        figura *v_ruedas_extras[2];   //vector de dos figuras circulo

        public:

                        normal(int x, int y);         //constructor parametrizado
                        void pintar();                //pintar coche
                        void borrar();                //borrar coche
                        void mover(int tecla);        //mover coche
                        void operator --(void);       //eliminar rueda
                        ~normal();                    //destructor
};

#endif
```

Normal.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "coche.h"
#include "normal.h"
```

```
normal::normal(int x, int y):coche(x,y)
{
	//cout<<"Ha entrado en el contructor coche normal sin parametrizar";
	//getchar();
	this->x=x;
	this->y=y;
	v_figuras[0]=new circulo(x-25,y-30,8,10);
	v_figuras[1]=new circulo(x+25,y-30,8,10);
	v_figuras[2]=new circulo(x-25,y+30,8,10);
	v_figuras[3]=new circulo(x+25,y+30,8,10);
	v_figuras[4]=new elipse(x-15,y-50,14,0,360,10,5);
	v_figuras[5]=new elipse(x+15,y-50,14,0,360,10,5);
	v_figuras[6]=new rectangulo(x-25,y-50,1,x+25,y+50);

	v_ruedas_extras[0]=new circulo(x-25,y,8,10);
	v_ruedas_extras[1]=new circulo(x+25,y,8,10);

	vel_horiz=10;
}

normal::~normal()
{
	//cout<<"Ha entrado en el destructor coche normal";
	//getchar();

	for (int i=0;i<7;i++)   delete v_figuras[i];
	for (int i=0;i<2;i++)   delete v_ruedas_extras[i];
}

void normal::operator --(void)
{
	cout<<"Operador sobrecargado -- "<<endl;
	v_figuras[1]->set_color(0);
	v_figuras[1]->pintar();
}

void normal::pintar()
{
	for (int i=0; i<7;i++)		v_figuras[i]->pintar();
	for (int i=0; i<2;i++)		v_ruedas_extras[i]->pintar();
}

void normal::borrar()
{
  for (int i=0; i<7;i++)		v_figuras[i]->borrar();
  for (int i=0; i<2;i++)		v_ruedas_extras[i]->borrar();
}
```

```
void normal::mover(int tecla)
{
        borrar();                                                   //Borrar coche
        if (tecla==77)
          {
                if (x<590)                                                  //Tecla derecha
                  {
                        x=x+vel_horiz;
                        for (int i=0; i<7;i++)
                          {
                                v_figuras[i]->set_x(v_figuras[i]->get_x()+vel_horiz);
                           }

((rectangulo *)v_figuras[6])->set_endx(((rectangulo *)v_figuras[6])->get_endx()+vel_horiz);

                        for (int i=0; i<2; i++)

v_ruedas_extras[i]->set_x(v_ruedas_extras[i]->get_x()+vel_horiz);

                   }
          }

        if (tecla==75)                  //Tecla izquierda
          {
                if (x>210)
                  {
                        x=x-vel_horiz;
                        for (int i =0; i<7;i++)
                          {
                                v_figuras[i]->set_x(v_figuras[i]->get_x()-vel_horiz);
                           }

((rectangulo *)v_figuras[6])->set_endx(((rectangulo *)v_figuras[6])->get_endx()-vel_horiz);

                        for (int i=0; i<2; i++)

v_ruedas_extras[i]->set_x(v_ruedas_extras[i]->get_x()-vel_horiz);

                   }
          }
        pintar();
}
```

Rápido.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "coche.h"
#ifndef rapido_h
#define rapido_h

class rapido:public coche
{
        private:
                        int vel_vert;           //velocidad vertical del coche rápido

        public:

                        rapido(int x, int y);           //constructor sin parametrizar
                        void operator --(void);         //eliminar rueda
                        void set_vel_horiz(int v);      //Introduce la velocidad horiz.
                        void set_gasolina(int g);       //Introduce la gasolina inicial
                        void pintar();                  //pintar coche rapido
                        void borrar();                  //borrar coche rapido
                        void mover(int tecla);          //mover coche rapido
                        ~rapido();                      //destructor
};
#endif
```

Rápido.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "coche.h"
#include "rapido.h"
```

```
rapido::rapido(int x, int y):coche(x,y)
{
	//cout<<"Ha entrado en el contructor coche rapido parametrizado";
	//getchar();
	this->x=x;
	this->y=y;

	v_figuras[0]=new circulo(x-20,y-30,15,6);		//Ruedas
	v_figuras[1]=new circulo(x+20,y-30,15,6);
	v_figuras[2]=new circulo(x-20,y+30,15,6);
	v_figuras[3]=new circulo(x+20,y+30,15,6);
	v_figuras[4]=new elipse(x-10,y-45,15,0,360,6,3);	//Faros
	v_figuras[5]=new elipse(x+10,y-45,15,0,360,6,3);
	v_figuras[6]=new elipse(x,y,4,0,360,20,50);		//Carrocería

	vel_vert=15;
}

void rapido::operator --(void)
{
	//cout<<"Operador sobrecargado -- "<<endl;
	v_figuras[1]->set_color(0);
	v_figuras[1]->pintar();
}

void rapido::set_vel_horiz(int v)
{
	vel_horiz=v;
}

void rapido::set_gasolina(int g)
{
	gasolina=g;
}

rapido::~rapido()
{
	//cout<<"Ha entrado en el destructor coche rapido";
	//getchar();
	for (int i=0;i<7;i++)
				delete v_figuras[i];
}

void rapido::pintar()
{
	//cout<<"Ha entrado en el metodo pintar coche rapido";
	//getchar();
	for (int i=0; i<7;i++)		v_figuras[i]->pintar();
}
```

```
void rapido::borrar()
{
	for (int i =0; i<7;i++)		v_figuras[i]->borrar();
}

void rapido::mover(int tecla)
{
	borrar();			//Borrar coche
	if (tecla==77)			//Tecla derecha
	  {
		if (x<605)
		  {
			x+=vel_horiz;
			for (int i =0; i<7;i++)
				v_figuras[i]->set_x(v_figuras[i]->get_x()+vel_horiz);
		  }
	   }

	if (tecla==75)			//Tecla izquierda
	  {
		if (x>190)
		  {
			x=x-vel_horiz;
			for (int i=0; i<7; i++)
				v_figuras[i]->set_x(v_figuras[i]->get_x()-vel_horiz);
		  }
	    }

	if (tecla==72)			//Tecla arriba
	  {
		if (y>60)
		  {
			y=y-vel_vert;
			for (int i=0; i<7; i++)
				v_figuras[i]->set_y(v_figuras[i]->get_y()-vel_vert);
		  }
	   }

	if (tecla==80)			//Tecla abajo
	  {
		if (y<420)
		  {
			y=y+vel_vert;
			for (int i=0; i<7; i++)
				v_figuras[i]->set_y(v_figuras[i]->get_y()+vel_vert);
		  }
	   }
	pintar();
}
```

Segmento.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"
#include "vasija.h"
#include "bache.h"
#include "accidente.h"
#include "lista.h"
#ifndef segmento_h
#define segmento_h

class segmento:public rectangulo
{
        private:
                        figura *carretera;                  //forma de la carretera
                        lista<objeto> *lista_objetos; //lista de obstaculos y bonificaciones
                        objeto *p_obj_aux;              //puntero a objetos de la lista
                        objeto *copia_obj;     //nuevo objeto q se inserta en la nueva lista

        public:
                segmento(int xs, int ys, int col, int xi, int yi);       //Constructor
                segmento(const segmento&original);                  //Constructor copia
                lista<objeto> * darlista();              //Da la lista de objetos del segmento
                figura * darcarretera();                  //Da la forma de la carretera
                ~segmento();                               //Destructor
};
#endif
```

Segmento.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <typeinfo.h>
```

```
#include <graphics.h>
#include "figura.h"
#include "rectangulo.h"
#include "circulo.h"
#include "elipse.h"
#include "objeto.h"
#include "bonificacion.h"
#include "estatua.h"
#include "vasija.h"
#include "bache.h"
#include "accidente.h"
#include "lista.h"
#include "segmento.h"

segmento::segmento(int xs,int ys, int col, int xi, int yi):rectangulo(xs,ys,col,xi,yi)
{
        //cout<<"Constructor parametrizado de segmento"<<endl;
        carretera=new rectangulo(xs,ys,col,xi,yi);
        lista_objetos=new lista<objeto>();
}

segmento::segmento(const segmento&original):rectangulo(original)
{
        //cout<<"Ha entrado en el constructor copia de segmento ...";
        //getchar();
        carretera = new rectangulo(*((rectangulo *)original.carretera));
        lista_objetos = new lista<objeto>();

        while (original.lista_objetos->noesfin())
          {
                p_obj_aux=original.lista_objetos->leer();
                if (typeid(*(p_obj_aux))==typeid(accidente))
                  {
                        copia_obj=new accidente(*((accidente *)p_obj_aux));
                        //copiando objeto accidente
                        lista_objetos->insertar(copia_obj);
                        //insertando en la lista de la copia
                  }
                if (typeid(*(p_obj_aux))==typeid(vasija))
                  {
                        copia_obj=new vasija(*((vasija *)p_obj_aux));
                        //copiando objeto vasija
                        lista_objetos->insertar(copia_obj);
                        //insertando en la lista de la copia
                  }
                if (typeid(*(p_obj_aux))==typeid(bache))
                  {
                        copia_obj=new bache(*((bache *)p_obj_aux));
                        lista_objetos->insertar(copia_obj);
                  }
```

```
                if (typeid(*(p_obj_aux))==typeid(estatua))
                  {
                        copia_obj=new estatua(*((estatua *)p_obj_aux));
                        //copiando objeto estatua
                        lista_objetos->insertar(copia_obj);
                        //insertando en la lista de la copia
                  }

                original.lista_objetos->avanzar();
        }
}

lista<objeto> * segmento::darlista()
{
        lista<objeto> *p_list;
        p_list=lista_objetos;
        return p_list;
}

figura * segmento::darcarretera()
{
        figura *p_carretera;
        p_carretera=carretera;
        return p_carretera;
}

segmento::~segmento()
{
        //cout<<"Ha entrado en el destructor segmento"<<endl;
        delete lista_objetos;
        delete carretera;
}
```

Turistas.h

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "lista.h"
#ifndef turista_h
#define turista_h
#define MAX 6
```

```
class turista
{
        private:

                        struct date visita;
                        lista<char> *visitados;
                        char * nombre_turista;
                        char * nacionalidad;
                        int puntos;

        public:

                        turista();                              //constructor sin parametrizar
                        turista(char *nombre, char *nacion);//contructor parametrizado
                        bool operator ==(const turista &turist);
                        char * get_nombre();                    //da nombre turista
                        int get_puntos();               //da puntos acumulados por turista
                        void set_puntos(int p);         //actualiza puntos
                        void pilla_fecha();             //pone fecha visita hoy
                        void set_fecha(int d,int m,int a);              //modifica fecha
                        void get_fecha(int &d,int &m,int &a);       //da fecha
                        void imprime_fecha();           //da la ultima visita del turista
                        char * get_nacionalidad();      //da nacionalidad del turista
                        void mostrar_monumentos();
                        //da monumento de la posicion i del vector de mon. visitados
                        char * encadenar(); //genera la cadena que se guarda en el fichero
                        void set_monumento(char * m); //inserta  monumento
                        char * get_monumento();    //obtiene un monumento de la lista
                        lista<char> * darlista();       //Devuelve la lista de monumentos v
                        ~turista();                     //destructor
};

#endif
```

Turistas.cpp

```
#include <iostream.h>
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>
#include "lista.h"
#include "turista.h"
#define MAX 6
```

```
turista::turista()
{
	cout<<"Ha entrado en el contructor turista sin parametrizar";
	getchar();
}

turista::turista(char *nombre, char *nacion)
{
	nombre_turista=new char[strlen(nombre)+1];
	strcpy(nombre_turista,nombre);
	nombre_turista[strlen(nombre)]='\0';

	nacionalidad=new char[strlen(nacion)+1];
	strcpy(nacionalidad,nacion);
	nacionalidad[strlen(nacion)]='\0';

	visitados=new lista<char>();

	char * inicio;

	//EL VISITANTE COMIENZA SIEMPRE EN LA OFICINA DE TURISMO

	inicio=new char[strlen("Oficina_de_Turismo\0")];   //Reserva mem xa la cadena
	strcpy (inicio,"Oficina_de_Turismo\0");
	set_monumento(inicio);
}

bool turista::operator == (const turista &turist)
{
	cout<<"Operador sobrecargado == "<<endl;
	if (strcmp (nombre_turista,turist.nombre_turista)==0)
		if (strcmp (nacionalidad,turist.nacionalidad)==0)
			{
				cout<<"son iguales"<<endl;
				getchar();
				return true ;
			}
		else
			{
				cout <<"son diferentes"<<endl;
				getchar();
				return false;
			}

	else	return false;
}
```

```
void turista::mostrar_monumentos()
{
	visitados->iniciar();
	while (visitados->noesfin())  //Recorremos la lista de monumentos visitados
	{
		cout<<"Monumento visitado: "<<visitados->leer()<<endl;
		//Leemos monumento
		visitados->avanzar();
	}
}

char * turista::encadenar()
{
	char * cadm;
	cadm=new char[180];
	cadm[0]='\0';
	visitados->iniciar();
	while (visitados->noesfin())  //Recorremos la lista de monumentos visitados
	{
		strcat(cadm,visitados->leer());   //Lee monumento y lo concatenamos
		strcat(cadm,"#");                 //Insertamos fin de campo
		visitados->avanzar();
	}
	cadm[strlen(cadm)-1]='\0';              //Quitamos el último fin de campo #
	return cadm;
}

void turista::set_monumento(char *m)
{
	visitados->insertar(m);
}

char * turista::get_monumento()
{
	char * mv;
	mv=new char[strlen(visitados->leer())+1];
	//Se reserva memoria para la long. de cadena + 1 (fin de línea)
	strcpy(mv,visitados->leer());
	//Copiamos la cadena
	mv[strlen(visitados->leer())]='\0';
	//Insertamos el fin de linea en la última posición de la cadena

	return  mv;
}

lista<char> * turista::darlista()
{
	return  visitados;
}
```

```
int turista::get_puntos()
{
        return puntos;
}

void turista::set_puntos(int p)
{
        puntos=p;
}

char * turista::get_nombre()
{
        return nombre_turista;
}

char * turista::get_nacionalidad()
{
        return nacionalidad;
}

void turista::pilla_fecha()
{
        getdate(&visita);                       //Tomamos la fecha
}

void turista::set_fecha(int d,int m,int a)
{
        visita.da_year = a;
        visita.da_day = d;
        visita.da_mon = m;
}

void turista::get_fecha(int &d,int &m,int &a)
{
        a=visita.da_year;
        d=visita.da_day;
        m=visita.da_mon;
}

void turista::imprime_fecha()
{
  printf("Fecha: %d ", visita.da_day);
  printf("%d ", visita.da_mon);
  printf("%d \n", visita.da_year);
}

turista::~turista()
{
        cout<<"Ha entrado en el destructor turista";
}
```

Árbol.h

```
#ifndef arbol_h
#define arbol_h
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "turista.h"

class arbol
{
        private:
                struct nodo                     //Definición de nodo
                        {
                                turista *giri;
                                struct nodo * izq;
                                struct nodo * der;
                        };

                typedef nodo *pnodo;
                pnodo arbol, padre, actual, nuevo;  //Punteros de tipo nodo

         public:
                arbol ();                       //Constructor sin parametrizar
                arbol (pnodo aux);              //Constructor
                bool insertar (turista *t);     //Inserta un turista en el árbol
                bool borrar (turista *t);       //Borra un turista del árbol
                arbol hijoizq ();
                arbol hijoder ();
                bool vacio (pnodo aux);         //Averigua si el árbol está vacío
                void inorden (pnodo aux);       //Ordenación inorden

                void guardar(char * fichet);
                void apertura(pnodo auxi, char *fturi);
                void volcado (pnodo aux, ofstream &turistas);

                void busqueda (pnodo aux, char *mm, char *aa);    //Busca por fecha
                void mostrar ();                        //Muestra el árbol inorden
                void buscar (char *mm, char *aa);
                turista* raiz (pnodo aux);
                bool hoja (pnodo aux);
                int altura (pnodo aux);
                void muestra_hojas (pnodo aux);
                int maximo (int n, int m);
                bool buscanodo (pnodo aux, int dis);
                void mostrar_nivel (int n);
                void mostrar_nivel_inv (int n);
                ~arbol();
};
#endif
```

Árbol.cpp

```
#include <fstream.h>
#include <string.h>
#include <stdio.h>
#include "arbol.h"

arbol::arbol()
{
  //cout<<"constructor arbol sin parmetrizar"<<endl;
  //getchar();
  arbol=NULL;
}

arbol::arbol(pnodo aux)
{
        cout<<"constructor arbol parmetrizado"<<endl;
        getchar();
        arbol=aux;
}

bool arbol::insertar (turista *t)
/* Recorremos el arbol para asegurarnos que el turista no está repetido.
  Si está, no lo insertaremos y finaliza, si no está lo introducimos */
{
        cout<<"InSerTandO TurIstA "<<t->get_nombre();
        cout<<" de "<<t->get_nacionalidad()<<" en El aRbol"<<endl;

        padre=NULL;
        actual=arbol;

        if ((nuevo = new nodo) == NULL)
          {
                cout <<"Error en la asignacion de memoria en un nuevo turista";
                getchar();
                exit (0);
          }

        nuevo->giri=t;
        nuevo->izq=NULL;
        nuevo->der=NULL;

        if (vacio(arbol))
          {
                //cout<<"Insertando primer elemento del arbol..."<<endl;
                arbol=nuevo;
          }
        Else
```

```
        {

    while( (!vacio (actual)) && (strcmp(t->get_nombre(),actual->giri->get_nombre())!=0) )
    /* Recorremos el arbol hasta el final o hasta que encontremos el turista. */
            {
                padre=actual;
                if     (strcmp ( t->get_nombre(), actual->giri->get_nombre() ) < 0)
                    actual=actual->izq;
                else if(strcmp ( t->get_nombre(), actual->giri->get_nombre() ) > 0)
                    actual=actual->der;
            }

    if ((!vacio (actual)) && (strcmp(t->get_nombre(),actual->giri->get_nombre())==0))
    /* Hemos parado antes de llegar al final. Se encontró turista */
            {
                cout<<"El turista "<<t->get_nombre()<<" ya existe."<<endl;
                return false;
            }

    else if (strcmp ( t->get_nombre(), padre->giri->get_nombre() ) < 0)
    /* El elemento nuevo será el hijo izquierdo */
                {
                    //cout<<"Es hijo izquierdo..."<<endl;
                    padre->izq=nuevo;
                    return true;
                }

    else if (strcmp ( t->get_nombre(), padre->giri->get_nombre() ) > 0)
    /* El elemento nuevo será el hijo derecho */
                {
                    //cout<<"Es hijo derecho..."<<endl;
                    padre->der=nuevo;
                    return true;
                }

    else if (strcmp(t->get_nombre(),actual->giri->get_nombre())==0)
            {
                cout<<"tienen = nombre y distinta nacionalidad"<<endl;
                getchar;
            }
    }
}
```

```
bool arbol::borrar (turista *t)
/* Sustituye el nodo borrado por la hoja más a la derecha del subarbol izquierdo */
  {
        cout<<"Borrando turista "<<t->get_nombre()<<" del arbol ..."<<endl;
        getchar();
        pnodo aux=arbol;
        pnodo c, ant;
        cout<<t->get_nombre()<<endl;
        cout<<aux->giri->get_nombre()<<endl;
        cout<<strcmp (t->get_nombre(),aux->giri->get_nombre())<<endl;
        getchar();
        if (aux == NULL)      return false;    //arbol vacío
                else if (strcmp (t->get_nombre(),aux->giri->get_nombre())<1)
                    hijoizq().borrar (t);
                        else if (strcmp (t->get_nombre(),aux->giri->get_nombre())>1)
                            hijoder().borrar (t);
        else
        {
                c=aux;
                if (aux->izq == NULL)                   aux=aux->der;
                        else if (aux->der == NULL) aux=aux->izq;
                else
                {
                        ant=aux;
                        c=aux->izq;
                        while (c->der != NULL)
                        {
                                ant=c;
                                c=c->der;
                        }
                        strcpy(c->giri->get_nombre(),aux->giri->get_nombre());
                        if (ant == aux)aux->izq = c->izq;
                        else ant->der = c->izq;
                }

                delete c;
                return true;
        }
    return false;
  }

arbol arbol::hijoizq ()
{
  return (arbol->izq);
}

arbol arbol::hijoder ()
{
        return (arbol->der);
}
```

```
bool arbol::vacio (pnodo aux)
// Devuelve cierto si es el nodo está vacio
{
  return (aux == NULL);
}

void arbol::mostrar ()
/* Muestra el recorrido En Orden del arbol  */
{
  inorden (arbol);
}

void arbol::buscar (char *mm, char *aa)
/* Muestra el recorrido En Orden del arbol  */
{
  busqueda (arbol,mm,aa);
}

turista* arbol::raiz (pnodo aux)
// Devuelve la raiz del arbol
{
        return (aux->giri);
}

void arbol::inorden (pnodo aux)
/* Realiza un recorrido En Orden del arbol, es decir, primero el subarbol
  izquierdo, luego muestra la raiz y a continuación el subarbol derecho */
{
        if (!vacio(aux))
        {
                inorden (aux->izq);
                cout<<"TURISTA: "<<aux->giri->get_nombre()<<endl;
                cout<<"Nacionalidad: "<<aux->giri->get_nacionalidad()<<endl;
                aux->giri->mostrar_monumentos();
                aux->giri->imprime_fecha();
                getchar();
                inorden (aux->der);
        }
}
```

```
void arbol::busqueda (pnodo aux, char *mm, char *aa)
/* Realiza un recorrido En Orden del arbol, es decir, primero el subarbol
   izquierdo, luego muestra la raiz y a continuación el subarbol derecho
   mostrando aquellos elementos que coinciden con los valores pasados */
{

        int d,m,a;
        char *mes,*anio;
        mes=new char[2];
        anio=new char[4];

        if (!vacio(aux))
        {
                busqueda (aux->izq,mm,aa);
                aux->giri->get_fecha(d,m,a);
                itoa (m,mes,10);
                itoa (a,anio,10);

        if (((strcmp(mm,mes))==0) && ((strcmp(aa,anio))==0))
        {
          cout<<endl<<"El turista nos visito el mes "<<mm<<" de "<<aa<<" :"<<endl;

          cout<<"TURISTA: "<<aux->giri->get_nombre()<<endl;
          cout<<"Nacionalidad: "<<aux->giri->get_nacionalidad()<<endl;
          aux->giri->mostrar_monumentos();
          aux->giri->imprime_fecha();
          getchar();
        }

        busqueda (aux->der,mm,aa);
   }
}

void arbol::guardar (char *fichet)
{
   apertura (arbol,fichet);
}

void arbol::apertura (pnodo auxi, char * fturi)
{
        ofstream turis(fturi);              //Abrimos el fichero para lectura
        if (!turis)
                {
                        cout<<"Error al abrir el fichero de turistas "<<fturi<<endl;
                        getchar();
                        exit(1);
                }
   volcado(auxi,turis);
   turis.close();
}
```

```
void arbol::volcado (pnodo aux, ofstream &turistas)
/* Realiza un recorrido En Orden del arbol, es decir, primero el subarbol
   izquierdo, luego muestra la raiz y a continuación el subarbol derecho */
{

        char * fila, *filam, *dia, *mes, *anio;
        fila=new char[1250];
        filam=new char[180];
        dia=new char[2];
        mes=new char[2];
        anio=new char[4];
        fila[0]='\0';
        filam[0]='\0';
        int d,m,a;

        if (!vacio(aux))
                {
                        volcado (aux->izq,turistas);
                        strcpy(fila,"\n");
                        strcat(fila,aux->giri->get_nombre());
                        strcat(fila,"#");
                        strcat(fila,aux->giri->get_nacionalidad());
                        strcat(fila,"#");

                        aux->giri->get_fecha(d,m,a);
                        itoa (d,dia,10);
                        itoa (m,mes,10);
                        itoa (a,anio,10);
                        strcat(fila,dia);
                        strcat(fila,"#");
                        strcat(fila,mes);
                        strcat(fila,"#");
                        strcat(fila,anio);
                        strcat(fila,"#");

                        strcpy(filam,aux->giri->encadenar());
                        strcat(fila,filam);

                        turistas<<fila;

                        volcado (aux->der,turistas);
  }
}
```

```
bool arbol::hoja (pnodo aux)
/* Devuelve cierto si los subarboles derecho e izquierdo del árbol estan vacios */
{
        if ((aux -> der == NULL) && (aux -> izq == NULL)) return true;
        else return false;
}

int arbol::altura (pnodo aux)  /* Devuelve la altura del arbol pasado por valor */
{
        int prof;
        if (vacio(aux))          prof=0;
        else
                {
                        prof = maximo (altura (aux -> izq), altura (aux -> der)) + 1;
                }
        return prof;
}

int arbol::maximo (int n, int m)        /* Devuelve el máx entre dos enteros pasados */
{
        if (n > m) return n;
        else return  m;
}

bool arbol::buscanodo (pnodo aux, int dis)
/* Muestro los valores de los nodos situados a una distancia 'dis' de su
   hoja más lejana. Devuelve falso si 'dis' es mayor que la altura del arbol */
{
        int aux1, aux2;
        if (!vacio(aux))
        {
          aux1 = altura (aux->izq);
          aux2 = altura (aux->der);

          if (maximo (aux1,aux2) < dis) return false;
          else
            {
                if ((aux1==aux2) && (aux1==dis)) cout<<aux->giri ->get_nombre();
                else
                  {
                    if (aux1 == dis) cout<<aux -> giri ->get_nombre()<<" ";
                    else buscanodo(aux->izq,dis);
                    if (aux2 == dis) cout<<aux -> giri ->get_nombre()<<" ";
                    else buscanodo(aux->der,dis);
                  }
                return true;
            }
        }
        else     return false;
}
```

```
void arbol::muestra_hojas (pnodo aux)
/* Realiza un recorrido en Post Orden hasta llegar a la hoja y la muesra por pantalla */
{
	if (!hoja (aux))
		{
			muestra_hojas (aux -> izq);
			muestra_hojas (aux -> der);
		}
	else cout<<aux -> giri ->get_nombre()<<" ";
}

/*
Método que escriba por pantalla los valores de todos los nodos que
tengan una distancia dada desde la raíz (la distancia de la raíz así misma es 0).
*/

void arbol::mostrar_nivel (int n)
/* Muestra el nivel del arbol pasado por valor */
{
	if (!vacio(arbol))
		{
			if (n == 0)	cout<<raiz(arbol)<<" "<<endl;
			else
				{
					hijoizq ().mostrar_nivel (n-1);
					hijoder ().mostrar_nivel (n-1);
				}
		}
}

/*
Implementa un método que escriba por pantalla los valores de los nodos tales que
la distancia a sus hojas más lejanas sea igual a un valor dado.
*/

void arbol::mostrar_nivel_inv (int n)
{
	pnodo tree = arbol;
	buscanodo (tree,n);
}

arbol::~arbol()
{
	//cout<<"Ha entrado en el destructor arbol."<<endl;
}
```

Lista.h

```
#ifndef lista_h
#define lista_h

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

template <class datos_lista>
class lista
{
        private:

                struct nodo
                {
                        datos_lista *info;
                        struct nodo *siguiente;
                };

                typedef struct nodo *ptrnodo;

                ptrnodo  cabeza, fin, p_actual, anterior, aux;

        public:

                lista();
                void crear_lista();
                void insertar(datos_lista *m);
                bool eliminar_actual();
                void eliminar(datos_lista *p);
                void iniciar();
                datos_lista* leer();
                bool noesfin();
                void avanzar();
                bool vacia();
                void borradototal();
                ~lista();
  };
```

```
template <class datos_lista>
lista <datos_lista>::lista()
{
        cabeza=NULL;
        fin=NULL;
        p_actual=NULL;
        anterior=NULL;
}


template <class datos_lista>
void lista <datos_lista>::crear_lista()
{
        //cout<<"Creando lista ..."<<endl;
        anterior=NULL;
        cabeza = NULL;
        fin = NULL;
        p_actual = cabeza;
}

template <class datos_lista>
void lista <datos_lista>::insertar (datos_lista *m)
{
        ptrnodo nuevo;
        if ((nuevo = new nodo) == NULL)
                {
                        cout <<"Error en la asignacion de memoria";
                        exit (0);
                }
        (*nuevo).info = m;
       (*nuevo).siguiente=NULL;
        if (vacia ())
                {
                        //cout<<"Insertando el primer elemento de la lista ..."<<endl;
                        p_actual = nuevo;
                        cabeza = nuevo;
                        fin = nuevo;
                }
        else
                {
                        //cout<<"Insertando elemento en la lista ..."<<endl;
                        fin->siguiente=nuevo;
                        fin=nuevo;
                }
}
```

```
template <class datos_lista>
bool lista <datos_lista>::eliminar_actual()
{
	//cout<<"eliminando p_actual..."<<endl;
	if (p_actual==cabeza)
	  {
		cabeza=cabeza->siguiente;
		delete p_actual->info;
		delete p_actual;
		p_actual=cabeza;
		anterior=NULL;
		return true;
	  }
	else
	  {
		if (p_actual==fin)
		  {
			fin=anterior;
			fin->siguiente=NULL;
			delete p_actual->info;
			delete p_actual;
			iniciar();
			while (p_actual!=fin)
			  {
				anterior=p_actual;
				p_actual=p_actual->siguiente;
			  }
			return true;
		  }
		else
		  {
			anterior->siguiente=p_actual->siguiente;
			delete p_actual->info;
			delete p_actual;
			p_actual=anterior->siguiente;
			return true;
		  }
	}
}
```

```
//No funciona si se le pasa un objeto identico a un elemento de la lista
//Tendriamos que sobrecargar el operador == para q comparase todos los campos
template <class datos_lista>
void lista <datos_lista>::eliminar (datos_lista *p)
{
        p_actual=cabeza->siguiente;
        anterior=cabeza;
        while (p_actual!=NULL)
        {
                if (p==(cabeza->info))
                {
                        //cout<<"Eliminanado nodo..."<<endl;
                        cabeza=cabeza->siguiente;
                        delete anterior;
                        anterior=cabeza;
                }
                if ((p_actual->info)==p)
                {
                        //cout<<"Eliminanado nodo..."<<endl;
                        aux=p_actual->siguiente;
                        delete p_actual;
                        anterior->siguiente=aux;
                }
                p_actual=p_actual->siguiente;
                anterior=anterior->siguiente;
        }
        if ((p_actual==NULL)&&(anterior!=NULL))
        {
                //cout<<"Eliminando ultimo nodo ... Lista vacia"<<endl;
                delete cabeza;
                cabeza=NULL;
                fin=NULL;
                p_actual=NULL;

          }
}

template <class datos_lista>
void lista <datos_lista>::iniciar ()
{
        //cout<<"Inicializando la lista al primer elemento ..."<<endl;
        p_actual = cabeza;
       anterior=NULL;
}

template <class datos_lista>
datos_lista* lista <datos_lista>::leer ()
{
        return p_actual->info;
}
```

```
template <class datos_lista>
bool lista <datos_lista>::noesfin ()
{
        if (p_actual != NULL)          return 1;
        else                           return 0;
}

template <class datos_lista>
void lista <datos_lista>::avanzar()
{
        anterior=p_actual;
        p_actual=p_actual->siguiente;
}

template <class datos_lista>
bool lista <datos_lista> :: vacia ()
{
        return (cabeza == NULL);
}

template <class datos_lista>
void lista <datos_lista>::borradototal ()
{
        //cout << "Destruyendo lista...\n";
        ptrnodo aux;
        while (!vacia ())
                {
                        aux = cabeza -> siguiente;
                        delete cabeza->info;
                        delete cabeza;
                        cabeza = aux;
                }
        cabeza = NULL;
        p_actual = NULL;
        anterior=NULL;
        fin = NULL;
        //cout <<"Lista destruida..."<<endl;
        //getchar ();
  }

template <class datos_lista>
lista <datos_lista>::~lista()
        {
                while (!vacia ())              borradototal();
        }

# endif
```

Cola.h

```
#ifndef cola_h
#define cola_h

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

template <class datos_cola>
class cola
{
        private:

                struct nodo
                        {
                                datos_cola *info;
                                struct nodo *siguiente;
                        };

                typedef struct nodo *ptrnodo;

                ptrnodo  inicio, fin;

        public:

                cola ();
                void encolar (datos_cola *d);
                datos_cola* primero ();
                void desencolar ();
                bool cola_vacia ();
                void borradototal ();
                ~cola ();
};

template <class datos_cola>
cola <datos_cola> :: cola ()
{
          inicio = NULL;
          fin = NULL;
}
```

```
template <class datos_cola>
void cola <datos_cola> :: encolar (datos_cola *d)
{
        ptrnodo nuevo;
        if ((nuevo = new nodo) == NULL)
                {
                        cout <<"error en la asignacion de memoria";
                        exit (0);
                }
        (*nuevo).info = d;
        if (cola_vacia ())
                {
                        inicio = nuevo;
                        fin = nuevo;
                }
        else
                {
                        fin -> siguiente = nuevo;
                        fin = nuevo;
                        fin -> siguiente = NULL;
                }
}

template <class datos_cola>
datos_cola* cola <datos_cola> :: primero ()
{
        return inicio -> info;
}

template <class datos_cola>
void cola <datos_cola> :: desencolar ()
{
        ptrnodo primero = NULL;
        primero = inicio;
        inicio = inicio -> siguiente;
        delete primero;
}

template <class datos_cola>
bool cola <datos_cola> :: cola_vacia ()
{
        return (inicio == NULL);
}
```

```
template <class datos_cola>
void cola <datos_cola> :: borradototal ()
{
	//cout << "destruyendo cola...\n";
	ptrnodo aux;
	while (!cola_vacia ())
		{
			aux = inicio -> siguiente;
			delete inicio;
			inicio = aux;
		}
	inicio = NULL;
	fin = NULL;
	//cout <<"cola destruida..."<<endl;
	//getchar ();
}

template <class datos_cola>
cola <datos_cola>::~cola()
{
	borradototal();
}

#endif
```