Nombre: ***Carné:***

tecnun

CAMPUS TECNOLÓGICO DE LA UNIVERSIDAD DE NAVARRA
NAFARROAKO UNIBERTSITATEKO CAMPUS TEKNOLOGIKOA
Escuela Superior de Ingenieros · Ingeniarien Goi Mailako Eskola

Examen Mayo'24 C++

Informática II

Prof. Paul Bustamante

Índice

1. Introducción

Recuerde que debe dejar los ejercicios en la carpeta "**C:\Temp\ExamInfor**". Si no los deja allí, no se le podrá corregir el examen. Trate de realizar primero el ejercicio que crea que es más sencillo y deje el complicado para el final.

2. Ejercicio 1: Empresa videojuegos (3.0 Pts.)

Una empresa de videojuegos te ha seleccionado para que implementes un nuevo algoritmo que será vendido en una nueva consola.

El juego, básicamente, consiste en lo siguiente:

- El programa debe generar al inicio un número aleatorio de 5 cifras. Para jugar, debe pedirle al usuario que introduzca un número también de 5 cifras, el cual va a comparar con el número generado por el ordenador y decirle al usuario los dígitos que ha acertado y luego los dígitos que están en la posición correcta y aumentar una variable que es el número de intentos. Por ejemplo, si el número generado es **21456** y el usuario introduce **32357**, el programa le dirá que ha acertado con los dígitos 2 y 5 pero que el único que está en la posición correcta es el 5. Si ya ha acertado todos, debe sacar un aviso por la consola diciendo que ha adivinado el número y en cuántos intentos.
- Además, el juego tendrá otra opción que es la de ver cómo va el juego. En esta opción le sacará por la consola el número, pero sólo los dígitos que va adivinando, el resto le aparecerá con una "X". Siguiendo con nuestro ejemplo, aparecerá: XXX5X, lo cual le da cierta ventaja para no volver a introducir otro número en esa posición. También aparecerá el número de intentos que lleva acumulado.

A continuación, una vez descrito el algoritmo del juego, te voy a describir las funciones y variables del programa:

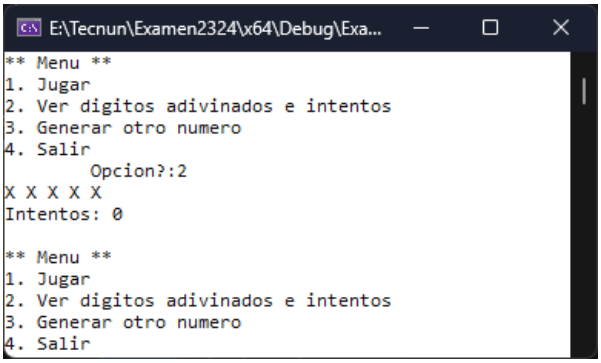
- **Variables**: Debes usar dos array's, uno para los dígitos y otro para ir marcando con 0 o 1 en la posición del dígito que ya ha sido adivinado. Además, necesitas otra variable para ir registrando los intentos realizados.
- **Funciones**: a éstas se accede a través del Menú.
    - Tiene que haber una función **Menu**() que devuelva la opción elegida por el usuario. El menú tendrá 4 opciones: 1-Jugar 2-Ver 3-Generar 4-Salir.
    - Función **Jugar**(..): permite hacer un intento del juego. Ganará el que adivine el número en un número de intentos menor.
    - Función **Genera**(..): deberá generar un nuevo número aleatorio de 5 cifras y poner la variable de intentos a cero. La función **rand**() permite generar un numero aleatorio entre 0 y 32767.
    - Función **Ver**(..): permite ver cómo va el juego (según la descripción anterior).

El esqueleto del programa principal se muestra a continuación:

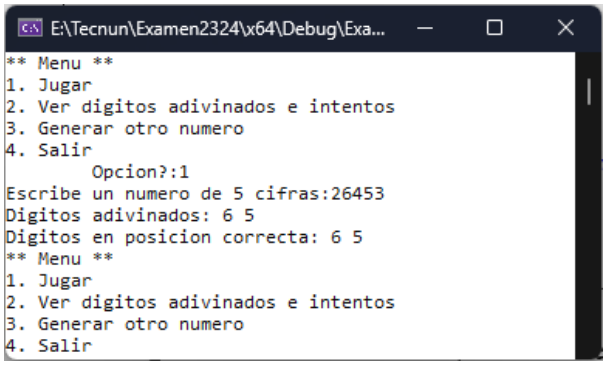
```
// definicion de main()
void main()
{
	char Adivina[] = { 0,0,0,0,0 };
	int Intentos = 0;
	//la primera vez genera el número de 5 cifras
	char *Numero = Genera(. . .);

	while (true) {
		int ret = Menu();
		switch (ret) {
		case 1: Jugar(. . .); break;
		case 2: Ver(. . .); break;
		case 3: Numero=Genera(..); break;
		case 4: return;
		}
	}
}
```

Los argumentos de las funciones quedan a criterio del alumno. Para convertir un número entero en una cadena de caracteres se usa la función `_itoa_s(num, buf ,len_buf, 10)`, donde **num** es la variable entera a convertir, **buf** es el array tipo char que recibirá la conversión, **len_buf** es el tamaño de buf y **10** es la base decimal.

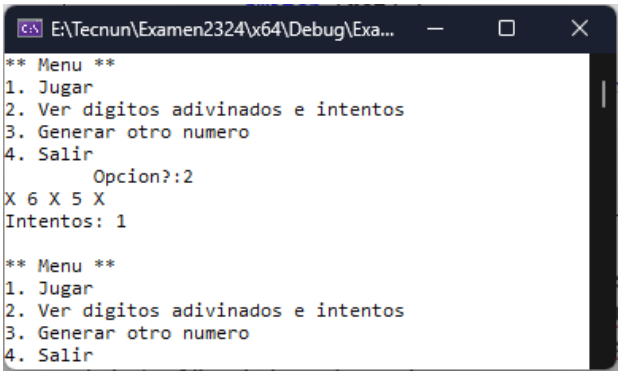
En las siguientes figuras se muestran las salidas por consola para clarificar mejor el ejercicio



Opción 2: Ningún digito adivinado aún y 0 intentos.



Opción 1 **Jugar**: suponiendo que el número generado es el "**76850**" y yo introduzco el "**26453**". He adivinado 2 dígitos (el 6 y el 5) y los dos están en la posición correcta.



Opción 2 nuevamente: ya tengo dos dígitos y 1 intento.

3. Ejercicio 2: Gestión de taller con clases (2.0 Pts.)

A Ud. le han pedido que haga un programa para la gestión de un taller de reparación y mantenimiento de coches, usando la POO. Dicho programa deberá tener las siguientes opciones:

- **Opcion1 (0.4 Pts.)**: Agregar coche: deberá pedir los datos de matrícula, la hora de ingreso y el número de servicios que se hará (para cada servicio debe pedir la descripción y el precio). Ver Figura 1.
- **Opcion2 (1.0 Pts.)**: Pagar servicio: deberá pedir la matrícula del coche (si esta no existe, debe sacar un mensaje). También deberá pedir la hora de finalización (para el coste de la mano de obra). Finalmente deberá sacar detallado el costo total del servicio (materiales más mano de obra). Ver Figura 3. Al final, cuando un coche paga, hay que **borrarlo de la lista**.
- **Opcion3 (0.4 Pts.)**: Listado de coches en servicio. Ver Figura 2.
- **Opcion4**: Salir.

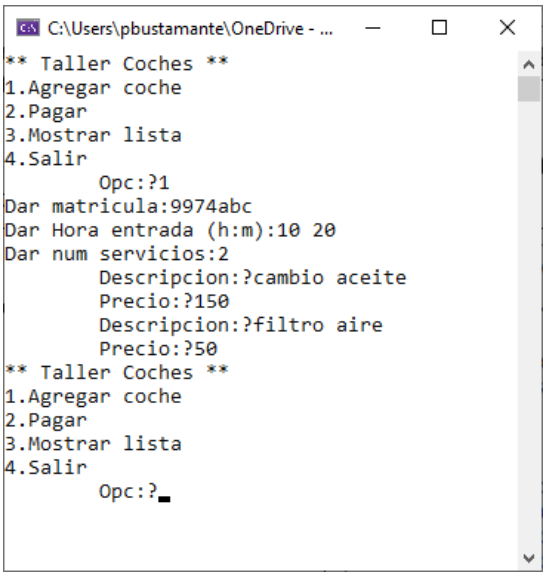


Figura 1 Agregar coche

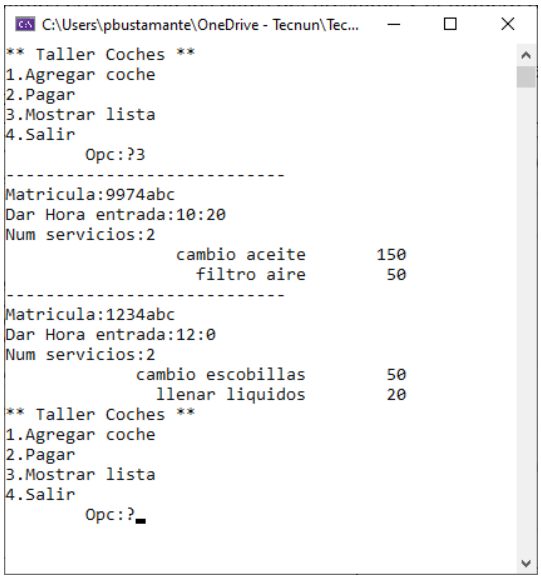


Figura 2 Listado de coches

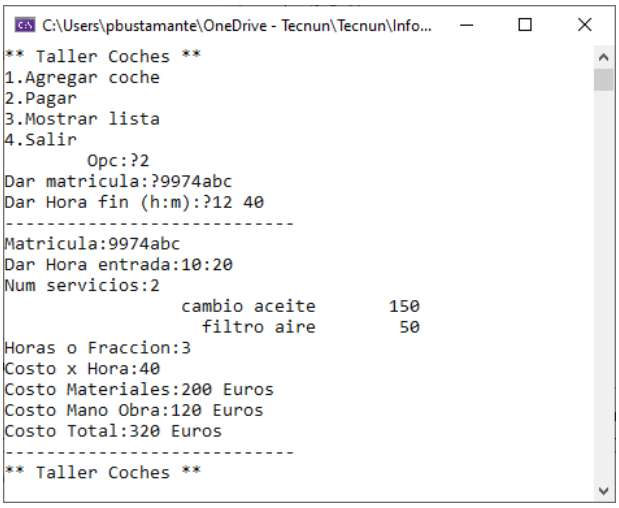


Figura 3 Pago del servicio

Programa:

Para la realización del programa, deberá crear las siguientes clases (con las variables y **SIN** cambiar el especificador de acceso, público o **privado**). Podrá poner los constructores que necesite y agregar más funciones.

```
class servicio
{
        char descripcion[30];
        double precio;
public:
        servicio();
        //agregar más funciones

};
class Time
{
        int hr, min;
public:
        Time(int h=0, int m=0);
        //agregar mas funciones

};
```

```
class mantenimento
{
        static double costo_hora;    //hora o fracción
        char matricula[20];
        servicio lstServ[10]; //máximo 10 servicios
        int numServ=0;    //número de servicios

        Time tini, tfin;
public:
        mantenimento();       //constructor
        //agregar las funciones necesarias

};
```

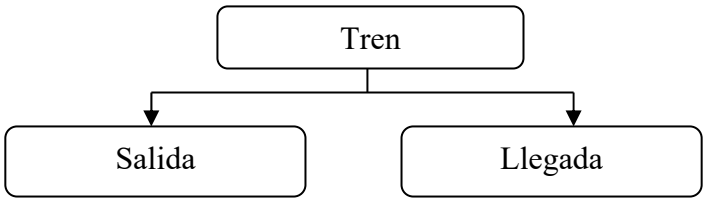
En cuanto a **main**, a continuación, se da un esqueleto de cómo podría ser:

```
#define MAX  100
double mantenimento::costo_hora = 40.0;

void main()
{
        int ns = 0;
        mantenimento *lista[MAX];


        int opc=0;
        while (1) {

            // mostrar el menú y hacer el resto del código

        }
}
```

Función principal (**main**) más optimización del código y comprobación de errores: **0.2** Pts.

4. Ejercicio 4: Gestión de Trenes con POO (2.5 Pts.)

A Ud. le ha contratado Renfe para que haga un programa que le permita llevar mejor la gestión de sus trenes de cercanías. Uno de los requerimientos es que lo haga utilizando sus conceptos adquiridos en esta asignatura de clases y herencia, las cuales se presentan en el siguiente árbol de clases:



En los siguientes apartados se presentan las clases y sus variables miembros y algunas de sus funciones. Usted debe evaluar si necesita más funciones (virtuales o no), constructores o destructores. Lo que no debe cambiar es el carácter de las variables: todas son **privadas**.

- **Clase Tren**: es la clase base del proyecto y tiene los siguientes miembros (privados): Linea, Via y Hora, la cual es una clase que tiene sólo 2 miembros enteros (tipo int): Hora y Minutos.

```
class Tren
```

```
{
        char *Linea; //C4, C9B por ejemplo
        int Via;            //Via: 1, 2, 5, etc
        Hora h;             //Hora de salida o de llegada
public:
        Tren();             //constr por defecto. Si necesita, haga otro
        virtual void Prt();
        // agregar + funciones si hace falta
};
```

- **Clase Salida**: es la clase derivada de **Tren,** que se usará para crear los trenes de salida. Tiene los siguientes miembros:

```
class Salida : public Tren
{
        char *Destino;            //Lugar de destino: Guadalajara, Parla, etc
public:
        Salida();           //constr por defecto. Si necesita, haga otro
        void Prt();
        // agregar + funciones si hace falta

};
```

- **Clase Llegada**: también es otra clase derivada de **Vuelo**, que se encargará de crear los vuelos de llegada. A continuación puede ver sus datos miembro:

```
class Llegada : public Tren
{
        char *Origen;//Lugar de Origen: Madrid, Parla, aeropuerto, etc.
public:
        Llegada();          //constr por defecto. Si necesita, haga otro
        void Prt();
        // agregar + funciones si hace falta

};
```

- **Clase Hora**: clase que no pertenece a la jerarquía, pero permitirá gestionar mejor la hora de los vuelos (Hora y Minutos). Vea a continuación sus miembros:

```
class Hora
{
        int hora, min;
public:
        Hora();       //constr por defecto. Si necesita, haga otro

};
```

Programa Principal

A continuación se describirá la funcionalidad del programa principal (**main**).

Debe crear en primer lugar un "**contenedor**" para almacenar los objetos que se van a ir creando (vuelos de salida o llegada). Esto puede hacerlo así: **Tren *lista[100];**

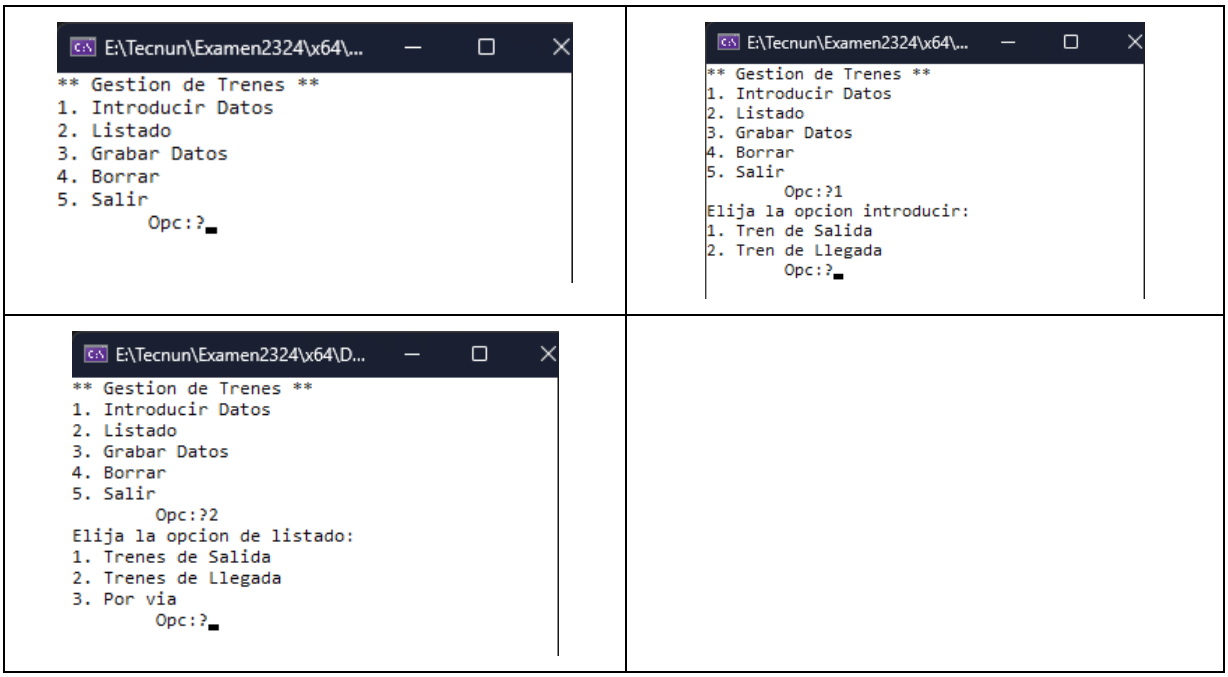
1. Debe crear un menú con las siguientes opciones (5):
    a. **Introducir Datos (0.4 pts)**: debe pedir si es de **Salida** o **Llegada**, tras lo cual debe pedir todos los datos necesarios: Linea, Via, Hora (de salida o llegada), Lugar (Destino u Origen). Debe crear el objeto de forma dinámica, con el operador **new**. La entrada de datos puede ser de forma desordenada (en el tiempo).
    b. **Listado de los Trenes (1.2 pts)**: con las siguientes opciones (preguntar al usuario):
        i. Los Trenes de Salida, **ordenados por la hora**, es decir desde las 00:00 hasta las 23:59.
        ii. Los Trenes de Llegada: **ordenados por la hora**, es decir desde las 00:00 hasta las 23:59.
        iii. Por la via (es decir si quiero que muestre los de la via "3", tanto de salida como de llegada).

c. **Grabar los datos (0.3 pts)** en un fichero, para guardar el trabajo realizado (debe pedir el nombre del fichero).

d. **Borrar un registro (0.4 pts):** En caso de haberse equivocado, que liste "**todos los trenes**" y pida al usuario qué registro borrar.

e. **Salir**.

2. Al terminar el programa debe liberar la memoria que ha reservado.



Resto del programa: función menú, código optimizado, liberar memoria, comprobación de errores, etc. **0.2 pts.**

¡Suerte!!