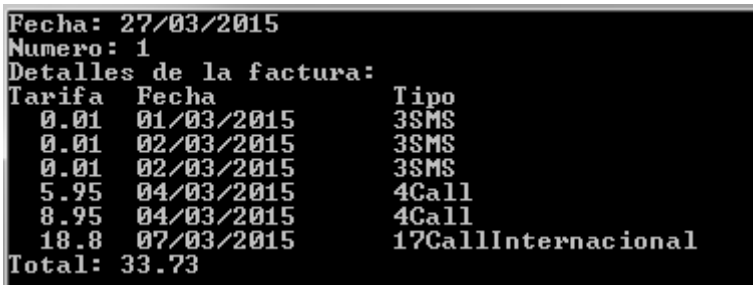
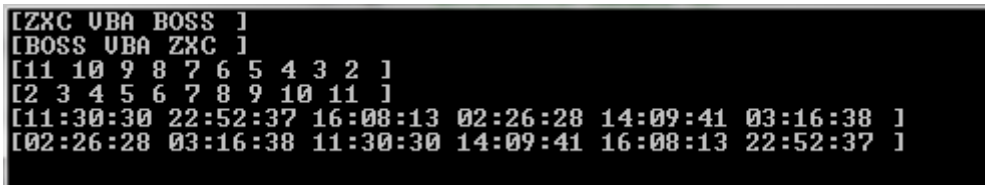
PRACTICAS C++ AVANZADO

1) HERENCIA / COMPOSICIÓN: Diseñar una factura para un operador de telefonía móvil para que se imprima como esta:

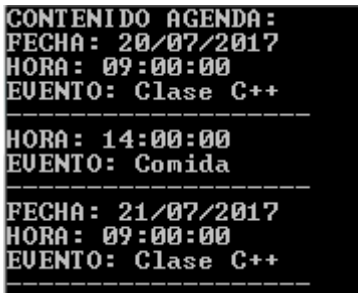


La factura tiene 3 tipos de servicios: SMS, Llamada nacional e internacional.
El SMS tiene fecha, hora, número destino y tarifa que va por unidad.
La llamada tiene una tarifa que va por segundo.
Y la llamada internacional a parte de la facturación por segundo añade una cuota fija de roamming.

2) TEMPLATE / MEM. DINAMICA: Implementar una clase vector con memoria dinámica y con un template. Debe de tener los métodos añadir, imprimir, ordenar. Probarlo con la clase string, con int y con una clase propia.
Pruebas con string, int y hora.



3) LIBRERÍA STL: Utilizando la STL implementar una clase Agenda que se puedan añadir eventos con una determinada fecha, hora y descripción del evento. Utilizar mapas. Podemos tener más de un evento en el mismo día y hay capacidad para distintos días. Implementar operaciones para imprimir los eventos de un día, de la agenda completa. Si añadimos un evento en el mismo día y hora, se machaca.



4) Diseñar una clase idioma que permita internacionalizar una aplicación. Se adjuntan dos ficheros de idioma ya preparados. Indicar el código del país: es, en, it, fr, etc.
Se puede utilizar la función Split para obtener las claves y los valores del fichero:

```
void split(const string &s, char delim, vector<string> &elems) {
   stringstream ss;
   ss.str(s);
   string item;
   while (getline(ss, item, delim)) {
      elems.push_back(item);
   }
}
```

El idioma se puede indicar por los argumentos de main. Si no existe se puede cargar uno por defecto.

5) Ahorcado (String): Implementar el juego del ahorcado. El ordenador genera una palabra aleatoria utilizando el alfabeto de 4 caracteres (será una consonante, vocal, consonante y vocal). El usuario intenta adivinarla, para ello utiliza el método adivina que le pasamos la secuencia que teclea el usuario y nos devuelve una cadena de la siguiente manera. Serán 4 caracteres: con X marca letras acertadas y bien colocadas, 0 una letra que está bien pero no está bien colocada y un guion medio para indicar que esa letra no pertenece a la palabra. ***Para la generación de número aleatorios utilizar la función rand() devuelve un entero aleatorio, del fichero <cstdlib>, y para inicializar el generador de números aleatorios tenemos: srand(time(NULL)); include de ctime.***
De forma opcional se puede ir pintando el ahorcado. Pintando tantas líneas como intentos lleve el usuario. Hay un fichero en la carpeta de prácticas.

6) **ListaEnteros** (**Punteros, POO**) Implementar una clase que represente una lista enlazada, los datos que manejemos dentro pueden ser números. Añadir los siguientes métodos:

```
Lista();
void insertar(int);
bool eliminar(int);
bool existe(int);
bool vacia();
int get(int);
void set(int, int);
int numeroElementos();
void borrarTodos();
```

```
void imprime();
virtual ~Lista();
```

7) Template: La lista anterior se puede convertir en una plantilla para almacenar cualquier tipo.
8) **Properties (Streams / Map / String)**: Implementar una clase Properties se cargará con un fichero de Texto con la siguiente sintaxis: clave=valor. Cada par se distribuye en una línea del fichero. La idea es poder almacenar dentro de nuestra clase parámetros de configuración que luego recuperaremos desde otra clase. La clase Properties dispondrá de un constructor que recibirá un fichero de propiedades, un método string getString(string clave). Se podrán modificar los valores de las propiedades para ello implementar un método putString y un método save que actualice el fichero original.

9) Implementar la funcionalidad de un cajero automático. En la parte de fichero se puede almacenar la información del cajero.
10) **Figuras** (**Métodos Virtuales, POO**) Implementar una jerarquía de clases que nos permita representar figuras en 2D y en 3D, las operaciones que queremos tener son sencillas. Para 2D cálculo de áreas y para 3D cálculo de volúmenes. Todas las clases dispondrán de un método visualizar() que mostrará los datos de cada figura. Trabajar con Circulo, Cuadrado, Triangulo, Cubo y Cilindro. Trabajar a distintos niveles definiendo arrays de Figuras, de Figuras2D y Figuras3D.

11) **HerenciaMúltiple**: Utilizar la clase Time antes generada y crear una nueva clase Date que represente una fecha con su hora. Escribir constructores y el método toString() en la clase DateTime. Utilizar herencia múltiple.

12) **Banco** (**POO, vector, Stream, Herencia**) Implementar una clase Cuenta con nombre, numero, saldo y tipo de interés. Con los métodos: ingresar, retirar, consultar saldo. Controlar las cantidades a ingresar y retirar. Crear una cuenta Vivienda con un tope de años y una cantidad máxima a retirar. Agregar a la clase cuenta un método que el usuario pueda imprimir los movimientos de la cuenta (fecha, tipo movimiento [INGRESO, REINTEGRO], cantidad (indicar si es o no negativa), y Saldo actual. Los movimientos se imprimirán del último al primero (clase vector de la STL). En main, crear una cuenta y mediante un menú ofrecer las distintas operaciones. Después probar a grabar la cuenta en un fichero de

texto. Al iniciar main se carga la cuenta del fichero y al terminar grabar la cuenta en el fichero. Mirar los posibles métodos que tenemos que añadir.

13) (**Map, String**) A partir de la clase Properties creada en el ejercicio 4, reutilizarla para poder internacionalizar una aplicación. El usuario indica el idioma y los mensajes de la aplicación y saldrán en el idioma elegido. Español / inglés.

14) **Excepcion**: Definir una exception propia, implementar algún método que la lance y luego probar a capturarla. Por ejemplo en la clase persona, lanzar una excepcion si la edad de la persona no está entre 0 y 100. Implementar la clase de la excepción, asociarla al método setEdad y probarla desde main.