EJERCICIOS DE LA ASIGNATURA PROGRAMACIÓN DE SISTEMAS DE TIEMPO-REAL

GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

industriales
etsii UPCT

Universidad Politécnica de Cartagena

ENUNCIADOS DE EJERCICIOS DE PROGRAMACIÓN BASADA EN OBJETOS

En este documento se presentan algunos ejercicios que hacen uso de la librería estándar del lenguaje C++, como `<vector>`, `<string>`, `<map>`, `<set>`, `<sstream>`, `<random>`. Se presentan en primer lugar los enunciados, mientras que las soluciones de los mismos se encuentran al final de este documento.

EJERCICIO 1

Haga una función que calcule y retorne el resultado de aplicar la función matemática *unión* (un nuevo conjunto que contiene todos los elementos no repetidos de los dos parámetros de entrada). La función recibe como parámetros los dos conjuntos de entrada (modelados como `set<int>`). Supondremos que los parámetros de entrada **NO** están vacíos. Haga también una función `main` con, al menos, un ejemplo de utilización, donde además se impriman los conjuntos de entrada y salida. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 2

Haga una función que calcule y retorne el resultado de aplicar la función matemática *intersección* (un nuevo conjunto que contiene solo los elementos que aparecen en ambos parámetros de entrada). La función recibe como parámetros los dos conjuntos de entrada (modelados como `set<int>`). Supondremos que los parámetros de entrada **NO** están vacíos. Haga también una función `main` con, al menos, un ejemplo de utilización, donde además se impriman los conjuntos de entrada y salida. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 3

Haga una función que calcule y retorne el resultado de aplicar la función matemática *diferencia* (un nuevo conjunto que contiene solo los elementos que aparecen en un conjunto, pero no en el otro). La función recibe como parámetros los dos conjuntos de entrada (modelados como `set<int>`). Supondremos que los parámetros de entrada **NO** están vacíos. Haga también una función `main` con, al menos, un ejemplo de utilización, donde además se impriman los conjuntos de entrada y salida. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 4

Haga una función que compruebe si dos conjuntos son *subconjuntos* (si todos los elementos de uno están también contenidos en el otro). La función recibe como parámetros los dos conjuntos de entrada (modelados como `set<int>`) y retorna un valor `bool`. Haga también una función `main` con dos ejemplos de utilización (uno con respuesta `true` y otro con respuesta `false`), donde además se impriman los conjuntos de entrada y salida. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 5

Haga una función que reciba un valor de tipo `string` y devuelva un valor **`bool`** que indique si el parámetro contiene caracteres repetidos. Si el parámetro está vacío, debe retornar **`false`**. Obligatoriamente, debe utilizar la colección `set<T>` para implementar la solución. Haga una función `main` con, al menos, tres ejemplos de utilización: parámetro vacío, parámetro con caracteres repetidos y parámetro sin caracteres repetidos. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 6

Haga una función que reciba un valor de tipo `string` y devuelva un valor **`bool`** que indique si el parámetro contiene caracteres repetidos. Si el parámetro está vacío, debe retornar **`false`**. Obligatoriamente, debe utilizar la colección `vector<T>` para implementar la solución. Haga una función `main` con, al menos, tres ejemplos de utilización: parámetro vacío, parámetro con caracteres repetidos y parámetro sin caracteres repetidos. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 7

Haga una función que reciba un valor de tipo `string` y devuelva un valor **`bool`** que indique si el parámetro contiene caracteres repetidos. Si el parámetro está vacío, debe retornar **`false`**. Obligatoriamente, debe utilizar la colección `map<T>` para implementar la solución. Haga una función `main` con, al menos, tres ejemplos de utilización: parámetro vacío, parámetro con caracteres repetidos y parámetro sin caracteres repetidos. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 8

Haga una función que reciba un valor de tipo `string` y devuelva un valor **`bool`** que indique si el parámetro contiene caracteres repetidos. Si el parámetro está vacío, debe retornar **`false`**. No puede utilizar ninguna colección (`vector<T>`, `set<T>`, `map<T>`, `array<T,N>`, ni array de C []) para implementar la solución. Haga una función `main` con, al menos, tres ejemplos de utilización: parámetro vacío, parámetro con caracteres repetidos y parámetro sin caracteres repetidos. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 9

Se pide que haga una función que acepte un string y retorne un nuevo string que contenga la secuencia más larga de vocales consecutivas contenidas en el string. Por ejemplo, si la entrada fuera *"abcdeaiefghija"*, la función debe retornar *"eaie"*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 10

Se pide que haga una función que acepte un vector de enteros y retorne un nuevo vector que contenga la secuencia más larga de números pares consecutivos. Por ejemplo, si la entrada fuera *{1,2,4,2,8,3,4,8,5,4,2,7}*, la función debe retornar *{2,4,2,8}*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 11

Se pide que haga una función que acepte un string y un número entero, y que retorne un string que contenga únicamente las palabras cuya longitud es superior al indicado por el parámetro numérico. Se supone que las palabras están separadas por un espacio en blanco. No es necesario eliminar los signos de puntuación (punto, coma, etc.) que puedan aparecer. Por ejemplo, si la entrada fuera *"uno dos tiempo. Real "* y el número fuera 4, la función debe retornar *"tiempo. Real "*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 12

Se pide que haga una función que acepte un vector de enteros y retorne un nuevo vector que contenga únicamente los números primos (aquellos que son únicamente divisibles por sí mismos y por la unidad) almacenados en el parámetro. Haga una función `main` con un ejemplo de utilización en el que genere el vector de entrada de forma aleatoria, mediante una distribución uniforme (entre 0 y 200) de la librería `<random>`. Añada todas las funciones auxiliares que pudiera necesitar.
Se deja como ejercicio.

EJERCICIO 13

Se pide que haga una función que acepte un string y un vector de enteros, ambos del mismo tamaño. La función debe retornar un nuevo string formado por tantas repeticiones de cada uno de los caracteres como indica los valores almacenados en el vector. Si no tienen el mismo tamaño, retornará el string vacío. Por ejemplo, si la entrada fuera *"abcd"* y *{1,2,3,2}*, la salida será "abbcccdd". Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 14

Se pide que haga una función que acepte un string y un vector de enteros (de tamaño menor que el string; si no, retorne string vacío), y que retorne un nuevo string resultado de eliminar del string que se pasa como parámetro las posiciones indicadas en el vector de enteros. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 15

Se pide que haga una función que acepte un vector de enteros y retorne un nuevo vector que contenga únicamente aquellas secuencias de tres números tales que el tercero es la suma de los dos anteriores. Por ejemplo, si el vector de entrada es *{1,2,3,2,2,3,5,8,5,4,2,7}*, el de salida tiene que ser *{1,2,3,2,3,5,3,5,8}*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 16

Se pide que haga una función que acepte un vector de string y retorne otro vector que contiene solo los strings que comienzan o acaban en una letra mayúscula. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar.

EJERCICIO 17

El sensor de distancia de la marca ACME genera una cadena de texto que tiene 181 valores de tipo double, separados por comas y con el siguiente formato: "*v1,v2,v3,...v181\n*". El sensor cubre un ángulo de 180º y mide, en cada punto de la cadena de texto, la distancia hasta el objeto que se encuentra a dicho ángulo (empezando por -90º). Haga una función que reciba un string como el descrito y retorne un valor que indique el ángulo y la distancia a que se encuentra el obstáculo más cercano (utilizando un **`struct`**). Haga una función `main` con un ejemplo de utilización en el que genere el string de entrada de forma aleatoria, mediante una distribución uniforme (entre 0 y 100) de la librería `<random>`. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 18

El sensor de potencia de la marca ACME devuelve una cadena de texto con el siguiente formato "*Amp= 'valorA' Volt= 'valorV'\n*", donde los valores son de tipo **`double`**. Se pide que haga un programa que acepte un string formado por varias cadenas como la anterior (siempre acabadas en *'\n'*), y devuelva un `vector<double>` que contenga el valor de la potencia (calculada como V*A). Un ejemplo del string de entrada es `string pot {"Amp= 1.1 Volt= 220.1\n Amp= 1.7 Volt= 220.2\n"`. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 19

Haga una función que reciba una `string` y un caracter de separación, y devuelva un `vector<string>` en el que se ha partido el string suministrado según el caracter de separación. El caracter de separación puede dejarlo o no, según le convenga. No olvide añadir el resto del `string` que pueda quedar, aunque no esté delimitado por el caracter de separación. Por ejemplo, si la entrada es *"hola. ADIOS. Tiempo*

33 2342@#¢..LALA", y el caracter de separación es el '.', el vector devuelto es: *{"hola", " ADIOS", " Tiempo 33 2342@#¢", "", "LALA"}*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 20

Haga una función que reciba una `string` y un numero entero positivo, y devuelva un `vector<string>` en el que se ha partido el string suministrado según la cantidad de palabras especificada por el número entero. Se supone que las palabras están separadas por un espacio en blanco. No es necesario eliminar los signos de puntuación (punto, coma, etc.) que puedan aparecer. No olvide añadir el resto del `string` que pueda quedar, aunque no esté delimitado por el caracter de separación. Por ejemplo, si la entrada es *"hola. ADIOS. Tiempo 33 2342@#¢..LALA"*, y el número es 2, el vector devuelto es: *{"hola. ADIOS.", " Tiempo 33", " 2342@#¢..LALA"}*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 21

El sensor GPS de la marca ACME devuelve una cadena de texto con el siguiente formato "*X: 'valorX' Y: 'valorY' Z: 'valorZ'\n*". Se pide que haga un programa que acepte un string formado por varias cadenas como la anterior (siempre acabadas en *'\n'*), y devuelva un vector de 3 valores de tipo **`double`**, que contenga la media de cada una de las tres coordenadas. Por ejemplo, si el string de entrada es `string gps {"X: 1 Y: 2 Z: 3\n X: 10 Y: 20 Z: 30\nX: 100 Y: 200 Z: 300\n"`, la función debe devolver el vector con los valores *{37 , 74, 111}*. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 22

El sistema de control de acceso de un colegio almacena el nombre del alumno y el retraso que ha tenido al llegar en una base de datos. Al final del mes, se genera un string con el formato (separado por comas) *nombre y apellidos, retraso en minutos\n*, siendo el retraso un valor entero. Una fila por alumno. El mismo alumno puede aparecer más de una vez. Un ejemplo de datos de entrada puede ser `string alus {"Aitor Menta, 12, \nArmando Casitas, 20\n Aitor Menta, 5\n"}`. Se pide que haga una función que retorne un mapa de nombres y total de minutos de retraso acumulados durante el mes. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 23

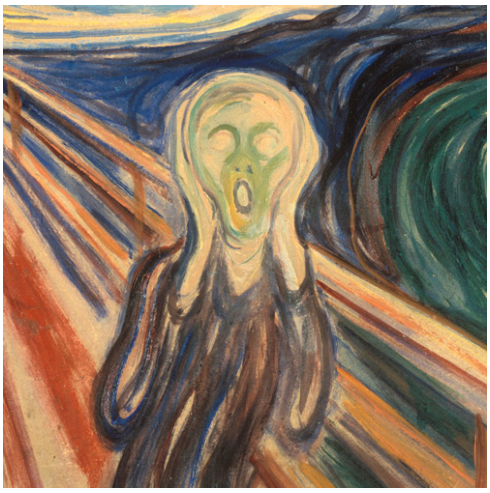
El sistema de pedidos de un supermercado recibe un string con la compra que solicita un cliente, con el formato (separado por comas) *producto, precio unitario, cantidad\n*. Un producto por fila. Se pide

que haga una función que reciba este string y retorne el precio total de la compra. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

EJERCICIO 24

La base de datos de un colegio retorna una cadena de texto con la siguiente información por cada alumno (formato separado por comas): *"'nombre', 'apellido', 'dia', 'mes', 'año nacimiento' \n"*. Se pide que haga un programa que acepte un string formado por varias cadenas como la anterior (siempre acabadas en *'\n'*) y que devuelva un vector que contenga la información de cada alumno en un **`struct`**. Suponga que los dos primeros campos de la cadena de entrada son de tipo `string`, y los tres siguiente son enteros. No se pide comprobar si la fecha de nacimiento es correcta, tan solo procesar la entrada. Un ejemplo de datos de entrada es `string alus {"Aitor, Menta Menta, 1, 12, 2010\nArmando, Casitas, 2,3, 2210\n"};`. Haga una función `main` con un ejemplo de utilización. Añada todas las funciones auxiliares que pudiera necesitar. Puede utilizar otras librerías de C++, como `<sstream>`, para procesar el string de entrada.

Las soluciones vienen a continuación.
Consultar solo si ya está un poco desesperad@, o para comparar soluciones



SOLUCIONES A LOS EJERCICIOS DE PROGRAMACIÓN BASADA EN OBJETOS

EJERCICIO 1

Esta función es muy sencilla, ya que la propiedad fundamental de un set<> es que no puede haber elementos repetidos. La función operator<< es cómoda para imprimir directamente todo el set<>. En la solución al ejercicio 2 pondré otra versión, más larga, pero que imprime el set<> sin la última ‘,’. He marcado los parámetros de la función en rojo para resaltar su uso e importancia. s1se pasa por copia mientras que s2 se pasa por referencia constante. ¿Por qué esta asimetría?. La función tiene que devolver un nuevo set<> (la unión), y este set<> contiene todos los valores de ambos sets<>. Pues bien, con esta forma de pasar los parámetros obtengo una copia de s1, y posteriormente le voy copiando todos los valores de s2. Por último, retorno s1 (que comenzó la función como una copia del set<> original).

```
#include <iostream>
#include <set>

using namespace std;

ostream& operator<< (ostream &os, const set<int> &s) {
    for (const auto &i : s)
        os << i << ", ";
    return os;
}

set<int> union_sets (set<int> s1, const set<int> &s2) {
    for (const auto &ss : s2)
        s1.insert(ss);
    return s1;
}

int main() {
    set<int> s1 {1,2,3,4,5}, s2 {10, 11, 12, 13}, s3 {10, 11,12, 5};
    set<int> setTMP;
    cout << s1 << endl;
    setTMP = union_sets(s1, s2);
    cout << "s1 U s2 => " << setTMP << '\n';
    setTMP = union_sets(s3, s2);
    cout << "s3 U s2 => " <<  setTMP<< '\n';
    return 0;
}
```

Si quisiera una versión en la que ambos set<> se pasan por referencia constante, tendría que ser:

```
set<int> union_sets (const set<int> &s1, const set<int> &s2) {
    set<int> tmp {s1}; // copia explícita
    for (const auto &ss : s2)
        tmp.insert(ss);
    return tmp;
}
```

No puedo añadir los elementos directamente a s1 porque entonces estaría modificando ¡el set<> original! (menos mal que lo paso por referencia constante a la función, y entonces al menos el compilador me avisaría de mi error).

EJERCICIO 2

La intersección entre dos set<> debe retornar los elementos que están en los dos conjuntos. En este caso, la estrategia es comprobar si los elementos de un conjunto ya están en el otro. Para ello, se utilizará la función miembro count(). Si el elemento no está (count() retorna cero), se elimina de un set<>. Se utiliza también el mismo "truco" respecto al paso de parámetros. Se presenta una función operator<< que imprime los sets<> sin la ',' final. El tipo real del iterador es `set<int>::iterator`, pero `auto` facilita la vida del programador.

```
#include <iostream>
#include <set>
#include <iterator>

using namespace std;

ostream& operator<< (ostream &os, const set<int> &s) {
    auto it {begin(s)}; // tipo real → set<int>::iterator
    os << *it;
    const auto fin {end(s)};
    for (it=next(it); it!=fin; it=next(it)) {
        os << ", " << *it;
    }
    return os;
}

set<int> interseccion_sets (set<int> s1, const set<int> &s2) {
    for (const auto &ss : s1) {
        if (s2.count(ss) == 0) {
            s1.erase(ss);
        }
    }
    return s1;
}

int main() {
    set<int> s1 {1,2,3,4,5}, s2 {10, 11, 12, 13}, s3 {10, 11,12, 5},
    set<int> setTMP;
    cout << s1 << endl;
    setTMP = interseccion_sets(s1, s2);
    cout << "s1 INT s2 => " << setTMP << '\n';
    setTMP = interseccion_sets(s3, s2);
    cout << "s3 INT s2 => " << setTMP << '\n';
    return 0;
}
```

EJERCICIO 3

La diferencia entre dos set<> debe retornar los elementos que están en un set<> pero no en el otro. En este caso, la estrategia seguida es eliminar de cada set<> aquellos elementos que aparecen también en el otro. Para ello, se hará uso de las funciones miembro count() (detectar si un elemento aparece en un set<>) y erase() (borrar una clave). Una vez se termine, tan solo resta por fusionar ambos set<>.

```
#include <iostream>
#include <set>

using namespace std;

ostream& operator<< (ostream &os, const set<int> &s) {
    for (const auto &i : s)
        os << i << ", ";
    return os;
}

set<int> diferencia_sets (set<int> s1, set<int> s2) {
    for (const auto &ss : s1) {
        if (s2.count(ss) != 0) {
            s1.erase(ss);
            s2.erase(ss);
        }
    }
    for (const auto &ss : s2)
        s1.insert(ss);
    return s1;
}

int main() {
    set<int> s1 {1,2,3,4,5}, s2 {10, 11, 12, 13}, s3 {10, 11,12, 5},
    set<int> setTMP;
    cout << s1 << endl;
    setTMP = diferencia_sets(s1, s2);
    cout << "s1 DIF s2 => " << setTMP << '\n';
    setTMP = diferencia_sets(s3, s2);
    cout << "s3 DIF s2 => " <<  setTMP << '\n';
    return 0;
}
```

EJERCICIO 4

En esta solución asumo que el orden de los parámetros es irrelevante, y por tanto busco en primer lugar el set<> con más elementos y le hago una referencia constante que se llama mayor, y el que tiene menos elementos y lo meto en una copia que se llama menor. Esta diferencia se debe a que la estrategia va a ser eliminar del set<> más pequeño todos los elementos del más grande, y finalmente comprobar si el set<> menor está vacío o no. En este caso, la utilización del operador ternario (?:) es muy útil para inicializar los set<> de forma más sencilla (aunque la sintaxis parezca decir que no es así).

```
#include <iostream>
#include <set>

using namespace std;

ostream& operator<< (ostream &os, const set<int> &s) {
    for (const auto &i : s)
        os << i << ", ";
    return os;
}

bool son_subconjunto (const set<int> &s1, const set<int> &s2) {
    set<int> menor { s1.size()>s2.size()? s2 : s1  };
    const set<int> &mayor { s1.size()>s2.size()? s1 : s2  };
    for (const auto &ss : mayor) {
        if (menor.count(ss) != 0) {
            menor.erase(ss);
        }
    }
    return menor.empty();
}

int main() {
    set<int> s1 {1,2,3,4,5}, s2 {10, 11, 12, 13}, s3 {10, 11,12};
    cout << std::boolalpha;
    cout << "s1 SUBC s2 => " << son_subconjunto (s1,s2) << '\n';
    cout << "s2 SUBC s1 => " << son_subconjunto (s2,s1) << '\n';
    cout << "s3 SUBC s2 => " << son_subconjunto (s3,s2) << '\n';
    cout << "s2 SUBC s3 => " << son_subconjunto (s2,s3) << '\n';
    return 0;
}
```

EJERCICIOS 1-4

Otra solución, más flexible y potente porque:

1. El parámetro genérico del set se respeta en todas las funciones, que ahora son genéricas.
2. Se implementa una función operator<< genérica para imprimir cualquier colección, desde un iterador inicial (inclusive) hasta uno final (exclusive). La función auxiliar, también genérica, colección_completa devuelve una pareja de iteradores [begin, end). Es una función que facilita el uso de operator<<

```
#include <iostream>
#include <set>
#include <iterator>

using namespace std;

template <typename T>
ostream& operator<< (ostream &os, pair<T,T> its) {
    if (its.first != its.second) { // necesario para evitar el set<> vacío
        T it = its.first;
        os << *it;
        it = next(it);
        for (; it != its.second; it = next(it)) {
            os << ", " << *it;
```

```
        }
    }
    return os;
}

template <typename T>
auto coleccion_completa (const T &c) {
    return make_pair(begin(c), end(c));
}

template <typename T>
set<T> union_sets (const set<T> &s1, const set<T> &s2) {
    set<T> tmp {s1};
    for (const auto &ss : s2)
        tmp.insert(ss);
    return tmp;
}

template <typename T>
set<T> interseccion_sets (set<T> s1, const set<T> &s2) {
    for (const auto &ss : s1) {
        if (s2.count(ss) == 0) {
            s1.erase(ss);
        }
    }
    return s1;
}

template <typename T>
set<T> diferencia_sets (set<T> s1, set<T> s2) {
    for (const auto &ss : s1) {
        if (s2.count(ss) != 0) {
            s1.erase(ss);
            s2.erase(ss);
        }
    }
    for (const auto &ss : s2) s1.insert(ss);
    return s1;
}

int main() {
    set<int> s1 {1,2,3,4,5}, s2 {10, 11, 12, 13}, s3 {10, 11,12, 5}, setTMP;
    cout <<  coleccion_completa (s1) << '\n';
    setTMP = union_sets(s1, s2);
    cout << "s1 U s2 => " << coleccion_completa (setTMP) << '\n';
    setTMP = union_sets(s3, s2);
    cout << "s3 U s2 => " <<  coleccion_completa (setTMP) << '\n';
    setTMP = interseccion_sets(s1, s2);
    cout << "s1 INT s2 => " << coleccion_completa (setTMP) << '\n';
    setTMP = interseccion_sets(s3, s2);
    cout << "s3 INT s2 => " <<  coleccion_completa (setTMP) << '\n';
    setTMP = diferencia_sets(s1, s2);
    cout << "s1 DIF s2 => " << coleccion_completa (setTMP) << '\n';
    setTMP = diferencia_sets(s3, s2);
    cout << "s3 DIF s2 => " <<  coleccion_completa (setTMP) << '\n';
    return 0;
}
```

EJERCICIO 5

Este es un ejemplo de problema con contraejemplo: no podemos concluir que no hay letras repetidas hasta que recorramos todo el string y no encontremos ninguna. Pero, en cuanto encontremos una letra repetida, ya no hace falta que terminemos de recorrerlo y podemos concluir que hay letras repetidas. En este caso, hacemos uso de la propiedad del set<> de que no admite duplicados. Por tanto, podemos utilizar el segundo valor del pair devuelto por la función miembro insert() para saber si la letra estaba ya en el set<> o no, y actuar en consecuencia.

```
#include <iostream>
#include <set>
#include <string>

using namespace std;

bool no_hay_repe_set (const string &str) {
    if (str.empty())
         return false;
    set<char> s;
    for (const auto &c : str) {
        auto res = s.insert(c); //C++17 → auto [it, nuevo] = s.insert(c);
        if (!res.second)        //C++17 → if (!nuevo)
            return false;
    }
    return true;
}

int main() {
    string sin_repes {"abced1234ABCD"}, con_repes{"abcd1234ABCDA"};
    string por_defecto, vacio{""};
    cout<< std::boolalpha; // para imprimir por consola 'true/false'
    cout << "por defecto -> " << no_hay_repe_set(por_defecto)  << '\n';
    cout << "vacio -> "       << no_hay_repe_set(vacio)        << '\n';
    cout << "sin repes -> "   << no_hay_repe_set(sin_repes)    << '\n';
    cout << "con repes -> "   << no_hay_repe_set(con_repes)    << '\n';
    return 0;
}
```

EJERCICIO 6

La estrategia con un vector<> es la misma, solo que en este caso hay que hacer un bucle for para buscar en el vector<> la letra. Si no está, se añade. Si está, se retorna el valor false.

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

using namespace std;
```

```
bool no_hay_repe_vector (const string &str) {
    if (str.empty())
        return false;
    vector<char> v;
    for (const auto &c : str) {
        for (const auto &vv : v) {
            if (c==vv)
               return false;
        }
        v.push_back(c);
    }
    return true;
}

int main() {
    string sin_repes {"abced1234ABCD"}, con_repes{"abcd1234ABCDA"};
    string por_defecto, vacio{""};
    cout<< std::boolalpha;
    cout << "por defecto -> " << no_hay_repe_vector(por_defecto) << '\n';
    cout << "vacio -> "       << no_hay_repe_vector (vacio)      << '\n';
    cout << "sin repes -> "   << no_hay_repe_vector (sin_repes)  << '\n';
    cout << "con repes -> "   << no_hay_repe_vector (con_repes)  << '\n';
    return 0;
}
```

EJERCICIO 7

Con el map<> es también muy sencillo, ya que se utiliza la letra como clave y se almacena un entero, que indica si se ha visto previamente la letra. El resto del código es similar a lo que se ha mostrado en las soluciones anteriores.

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

bool no_hay_repe_map (const string &str) {
    if (str.empty())
         return false;
    map<char,int> m;
    for (const auto &c : str) {
        if (m[c] == 1)
             return false;
        m[c] = 1;
    }
    return true;
}

int main() {
    string sin_repes {"abced1234ABCD"}, con_repes{"abcd1234ABCDA"};
    string por_defecto, vacio{""};
    cout<< std::boolalpha;
```

```
    cout << "por defecto -> " << no_hay_repe_map (por_defecto) << '\n';
    cout << "vacio -> "       << no_hay_repe_map (vacio)      << '\n';
    cout << "sin repes -> "   << no_hay_repe_map (sin_repes)  << '\n';
    cout << "con repes -> "   << no_hay_repe_map (con_repes)  << '\n';
    return 0;
}
```

EJERCICIO 8

En este ultimo caso hay que comparar una letra con todas las demás para detectar letras repetidas. Lo único que hay que tener en cuenta es que la última comparación que se tiene que realizar es la penúltima letra con la última. Por eso el bucle for de la i llega hasta tam-1, y el bucle for anidado de la j empieza en i+1.

```
#include <iostream>
#include <string>
using namespace std;
bool no_hay_repe (const string &str) {
    if (str.empty())
         return false;
    const auto tam = str.size(), tam_1 = tam -1;
    for (int i=0; i<tam_1; ++i) {
        for (int j=i+1; j<tam; ++j)
            if (str[i] == str[j])
                return false;
    }
    return true;
}

int main() {
    string sin_repes {"abced1234ABCD"}, con_repes{"abcd1234ABCDA"};
    string por_defecto, vacio{""};
    cout<< std::boolalpha;
    cout << "por defecto -> " << no_hay_repe (por_defecto) << '\n';
    cout << "vacio -> "       << no_hay_repe (vacio)      << '\n';
    cout << "sin repes -> "   << no_hay_repe (sin_repes)  << '\n';
    cout << "con repes -> "   << no_hay_repe (con_repes)  << '\n';
    return 0;
}
```

EJERCICIOS 9 Y 10

Estos dos ejercicios son muy parecidos, puesto que ambos piden encentrar la secuencia de valores consecutivos que cumplen una determinada propiedad. Se presenta la solución para vocales consecutivas, y se deja como ejercicio encontrar los números pares consecutivos. El código presenta dos soluciones:

- Versión 1: se recorre el string, buscando la secuencia más larga. Se guarda el índice donde termina y la cantidad de elementos que tiene. Se devuelve un substring del parámetro, utilizando la función miembro substr()
- Versión 2: se recorre el string que se pasa como parámetro, almacenando en un string auxiliar todas las vocales seguidas que se encuentran. Cada vez que se encuentra un string más largo que el más largo encontrado hasta el momento, se actualizan las variables max_long y res.

```
#include <iostream>
#include <string>
using namespace std;
bool es_vocal (const char c) {
    if ( c=='a' || c=='e' || c=='i' || c=='o' || c=='u') { return true; }
    else { return false; }
}

string secuencia_vocales1 (const string &str) {
    int fin=0, longitud=0, max_long=0;
    const int tam = str.size();
    for (int i=0; i<tam; ++i) {
        if (es_vocal(str[i])) {
            ++longitud;
        } else {
            if (longitud>max_long) {
                fin=i;
                max_long=longitud;
            }
            longitud=0;
        }
    }
    if (longitud>max_long) {
        fin = tam;
        max_long = longitud;
    }
    return str.substr(fin-max_long,max_long);
}
string secuencia_vocales2 (const string &str) {
    string tmp, res;
    int max_long=0;
    const int tam = str.size();
    for (const auto & c : str) {
        if (es_vocal(c)) {
            tmp.push_back(c);
        } else {
            if (tmp.size()>max_long) {
                res = tmp;
                max_long = tmp.size();
            }
            tmp.clear();
        }
    }
    if (tmp.size()>max_long)
      { res = tmp;  }
    return res;
}
```

```
int main() {
    string ppio {"aeeeeeabcdeaiefghija"};
    string mitad {"abcdeaijaeeeeeakjlkhsdf"};
    string final {"abcdeaiefghijaeeeeea"};
    cout << "Version 1 ppio -> " << secuencia_vocales1 (ppio) << '\n';
    cout << "Version 2 ppio -> " << secuencia_vocales2 (ppio) << '\n';
    cout << "Version 1 mitad -> " << secuencia_vocales1 (mitad) << '\n';
    cout << "Version 2 mitad -> " << secuencia_vocales2 (mitad) << '\n';
    cout << "Version 1 final -> " << secuencia_vocales1 (final) << '\n';
    cout << "Version 2 fina -> " << secuencia_vocales2 (final) << '\n';
    return 0;
}
```

EJERCICIO 11

Tres soluciones se muestran en el código: dos basadas en streams y una realizando el procesamiento del string a mano. Las dos soluciones basadas en stream son similares, diferenciándose únicamente en la forma de extraer la información: con formato (operator>>) o sin formato (getline). En este caso, son iguales, ya que se saca únicamente información en forma de cadena de caracteres. La transformación de un string en un stream es común y ayuda a resolver fácilmente muchas situaciones, como se mostrará en otros ejercicios posteriores.
Pero la segunda solución, el procesamiento a mano, me gusta más. Muestra, además, otra estrategia típica (como la búsqueda de un contraejemplo) que se puede utilizar en programación. El bucle principal, en lugar de recorrer el string, se encarga de encontrar palabras. Dentro está organizado en: encontrar inicio de la palabra (primer while), encontrar punto de finalización (segundo while), y posteriormente copia esa sección en el string que retorna la función si cumple la condición. A pesar de que parece más lógico poner un bucle for que recorra uno a uno los caracteres del string, realmente el bucle while que se muestra en la solución modela mejor la forma, y las fases, de la solución. Es, además, una estrategia aplicable en multitud de problemas.

```
#include <iostream>
#include <string>
#include <sstream>
#include <cctype>

using namespace std;

string palabras_mas_grandes_sstream (const string &str, const int tam) {
    istringstream ss {str};
    string tmp, res;
    while (!ss.eof()) {
        ss >> tmp;
        if (tmp.size() >= tam)
            res += tmp;
        tmp.clear();
    }
    return res;
}

string palabras_mas_grandes_sstream2 (const string &str, const int tam) {
    istringstream ss {str};
    string tmp, res;
    while (!ss.eof()) {
        getline(ss, tmp, ' ');
        if (tmp.size() >= tam)
```

```
            res += tmp;
    }
    return res;
}

string palabras_mas_grandes (const string &str, const int tam) {
    int ini, fin, i=0;
    const int size_str = str.size();
    string res;
    do {
        while (isspace(str[i])) ++i;
        ini=i;
        while (!isspace(str[i])) ++i;
        fin=i;
        if (fin-ini>=tam)
            res +=str.substr(ini, fin-ini);
    } while (i<size_str);
    return res;
}

int main() {
    string prueba {"uno dos tiempo.   Real  "};
    cout << "prueba stream1 => "
         << palabras_mas_grandes_sstream (prueba,4) << '\n';
    cout << "prueba stream2 => "
         << palabras_mas_grandes_sstream2 (prueba,4) << '\n';
    cout << "prueba sin stream => "
         << palabras_mas_grandes (prueba,4) << '\n';
    return 0;
}
```

EJERCICIO 13

Se presentan dos soluciones a este ejercicio. La primera función es la que aparece en un primer impulso, mientras que la segunda, más corta, se obtiene al mirar con detenimiento los constructores que proporciona la clase string.

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

using namespace std;

string repetir_caracteres_1 (const string &str, const vector<int> &reps) {
    const int str_tam = str.size();
    if (str_tam != reps.size()) return {};
    string res;
    for (int i=0; i<str_tam; ++i) {
        const int tmp = reps[i];
        for (int j=0; j<tmp; ++j) {
            res+=str[i];
        }
    }
    return res;
}

string repetir_caracteres_2 (const string &str, const vector<int> &reps) {
```

```
    const int str_tam = str.size();
    if (str_tam != reps.size()) return {};
    string res;
    for (int i=0; i<str_tam; ++i) {
        string tmp (reps[i], str[i]);
        res += tmp;
    }
    return res;
}

int main () {
    string str {"abcd"}, malo{"Tiempo real"};
    vector<int> reps {1,2,3,2};
    cout << '|' << repetir_caracteres_1(str, reps) << '|' <<endl;
    cout << '|' << repetir_caracteres_1(malo, reps) << '|' <<endl;
    cout << '|' << repetir_caracteres_2(str, reps) << '|' <<endl;
    cout << '|' << repetir_caracteres_2(malo, reps) << '|' <<endl;
    return 0;
}
```

EJERCICIO 14

Se presentan tres soluciones: una errónea (además, la más obvia; ¿por qué?) y dos correctas. La primera se basa en eliminar los caracteres del string (que, además, se pasa por copia), mientras que la segunda construye un nuevo string a partir de los caracteres que no se quieren eliminar. Para ello, la segunda solución hace uso de la función miembro set<>::count(). En la primera solución correcta se utiliza un iterador inverso para recorrer al revés el set<>, que asegura que no hay elementos repetidos.

```
#include <iostream>
#include <string>
#include <set>
#include <iterator>

using namespace std;

string eliminar_caracteres_1_erroneo (string str, const set<int> &reps) {
    const int str_tam = str.size();
    if (str_tam <= reps.size()) return {};
    for (const auto &i : reps)
        str.erase(i,1);
    return str;
}

string eliminar_caracteres_1 (string str, const set<int> &reps) {
    const int str_tam = str.size();
    if (str_tam <= reps.size()) return {};
    auto fin = rend(reps);
    for (auto it=rbegin(reps); it!=fin; it = next(it))
        str.erase(*it,1);
    return str;
}
```

```
string eliminar_caracteres_2 (const string &str, const set<int> &reps) {
    const int str_tam = str.size();
    if (str_tam <= reps.size()) return {};
    string res;
    for (int i=0; i<str_tam; ++i) {
        if (reps.count(i)==0)
            res += str[i];
    }
    return res;

}

int main () {
    string ej{"Tiempo real"};
    set<int> eliminar {2,4,8,9};
    //cout << '|' << eliminar_caracteres_1_erroneo(ej, eliminar) << '|' <<endl;
    cout << '|' << eliminar_caracteres_1(ej, eliminar) << '|' <<endl;
    cout << '|' << eliminar_caracteres_2(ej, eliminar) << '|' <<endl;
    return 0;
}
```

EJERCICIO 15

En este ejercicio hay que tener cuidado de no salirse de los límites del vector al operar con tres valores en posiciones consecutivas del vector.

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

using namespace std;

vector<int> suma_de_2_anteriores (const vector<int> &v) {
    vector<int> res;
    const int v_tam = v.size();
    for (int i=2; i<v_tam; ++i) {
        if (v[i] == v[i-1] + v[i-2]) {
            res.push_back(v[i-2]);
            res.push_back(v[i-1]);
            res.push_back(v[i]);
        }
    }
    return res;
}

ostream& operator<< (ostream& os, const vector<int> &v) {
    for (const auto vv : v)
        os << vv << ", ";
    return os;
}

int main () {
    vector<int> entrada {1,2,3,2,2,3,5,8,5,4,2,7};
    vector<int> salida {1,2,3,2,3,5,3,5,8};
    vector<int> res {suma_de_2_anteriores (entrada)};
```

```
    cout << res << (res==salida? " OK " : " KO ") << endl;
    return 0;
}
```

EJERCICIO 16

Se deja como ejercicio para resolver, pero se dan las siguientes indicaciones:

1. Existe un función int isupper(char c), definida en <cctype>, que indica si el carácter que se le pasa como parámetro está en mayúsculas (retorna un valor distinto de cero).
2. La clase string define dos funciones miembro, front() y back(), que acceden directamente al primer y último carácter del string.

EJERCICIO 17

Resolución habitual, no tiene mucho secreto. Quizás utilizar stringstream para procesar más fácilmente el string de entrada, aprovechando la opción que ofrece getline() para especificar el carácter hasta el que leer la entrada.

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

using namespace std;

template <class T>
ostream& operator<< (ostream& os, const vector<T> &v) {
    for (const auto vv : v)
        os << vv << ", ";
    return os;
}

struct datos {
    double distancia;
    int angulo;
};

vector<double> procesar (const string &acme) {
    vector<double> res;
    istringstream ss {acme};
    do {
        string tmp;
        getline(ss, tmp, ',');
        res.push_back(stod(tmp));
    } while (!ss.eof());
    return res;
}

datos obstaculo_cercano (const string &acme) {
    datos tmp;
```

```
    vector<double> dists = procesar (acme);
    const int tam_dists = dists.size();
    int ang=0;
    double dist=dists[0];
    for (int i=1; i<tam_dists; ++i) {
        if (dist>dists[i]) {
            ang=i;
            dist=dists[i];
        }
    }
    tmp.distancia = dist;
    tmp.angulo = ang - 90;
    return tmp;
}

int main () {
    std::random_device rd;
    std::default_random_engine gen(rd());
    uniform_int_distribution<> dists (0, 100);
    string acme {to_string(dists(gen))};
    for (int i=0; i<180; ++i)
        acme += ", " + to_string(dists(gen));
    acme.push_back('\n');
    cout << acme << endl;
    datos res = obstaculo_cercano(acme);
    cout << '@' << res.angulo << " -> " << res.distancia << endl;
    return 0;
}
```

EJERCICIO 18

En este caso no se ha utilizado stringstream, sino que se ha procesado el string aprovechando la estructura que tiene.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
template <class T>
ostream& operator<< (ostream& os, const vector<T> &v) {
    for (const auto vv : v)
        os << vv << ", ";
    return os;
}
vector<double> potencias (const string &str) {
    vector<double> pot;
    double a, v;
    const int tam_str = str.size();
    int p=0, u;
    do {
        p = str.find("p=", p)+2;
        u = str.find("V", p);
        a = stod(str.substr(p, u-p));
        p = str.find("t=", p)+2;
```

```
        u = str.find("\n", p);
        v = stod(str.substr(p, u-p));
        u+=2;
        pot.push_back(a*v);
        //cout << "A " << a << ", V " << v << endl;
    } while (u<tam_str);
    return pot;
}
int main () {
    string pot {"Amp= 1.1 Volt= 220.1\n Amp= 1.7 Volt= 220.2\n Amp= 1.1 Volt= 220.1\n"};
    vector<double> pots = potencias(pot);
    cout << pots << endl;
    return 0;
}
```

EJERCICIO 19

La solución está basada en lo que se ha comentado en ejercicios anteriores.

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

using namespace std;

vector<string> separar_por_char (string &str, const char c) {
    vector<string> tmp;
    int p=0, u=0;
    bool continuar = true;
    do {
        while (str[u] != c && u<str.size()) ++u;
        tmp.push_back(str.substr(p, u-p));
        if (u<str.size()) {
            ++u;
            p=u;
        } else
           continuar = false;
    } while (continuar);
    return tmp;
}

void imprimir(const vector<string> &v) {
    for (const auto &vv : v)
        cout << '|' << vv << "|\n";
}

int main() {
    string str = "hola. ADIOS. Tiempo 33 2342@#¢..LALA";
    auto res = separar_por_char(str, '.');
    imprimir(res);
    return 0;
}
```

EJERCICIO 20

La solución está basada en lo que se ha comentado en ejercicios anteriores, y es particularmente parecida a la del ejercicio anterior.

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

using namespace std;

void imprimir(const vector<string> &v) {
    for (const auto &vv : v)
        cout << '|' << vv << "|\n";
}

vector<string> separar_por_num_palabras (string &str, const int num) {
    vector<string> tmp;
    auto p = begin(str), u = begin(str);
    int n=0;
    for (u; u!=end(str); u=next(u)) {
        if(isblank(*u)) {
            ++n;
            if (n==num) {
                n=0;
                tmp.emplace_back(p,u);
                p=u;
            }
        }
    }
    if(p!=end(str))
        tmp.emplace_back(p,u);
    return tmp;
}

int main() {
    string str = "hola. ADIOS. Tiempo 33 2342@#¢..LALA";
    auto res = separar_por_num_palabras(str, 2);
    imprimir(res);
    return 0;
}
```

EJERCICIO 21

La solución es similar a la de ejercicios anteriores.

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

```
using namespace std;

template <class T>
ostream& operator<< (ostream& os, const vector<T> &v) {
    for (const auto vv : v)
        os << vv << ", ";
    return os;
}

vector<double> valor_medio_gps (const string &str) {
    vector<double> tmp {0,0,0};
    int p=0, u, num=0;
    do {
        p = str.find("X: ", p) + 3;
        u = str.find("Y: ", p);
        tmp[0] += stod(str.substr(p, u-p));
        p=u+3;
        u = str.find("Z: ", p);
        tmp[1] += stod(str.substr(p, u-p));
        p=u+3;
        u = str.find("\n", p);
        tmp[2] += stod(str.substr(p, u-p));
        ++num;
    } while (u!=str.size()-1);
    tmp[0] /= num;
    tmp[1] /= num;
    tmp[2] /= num;
    return tmp;
}

int main() {
    string gps {"X: 1 Y: 2 Z: 3\n X: 10 Y: 20 Z: 30\n
                 X: 100 Y: 200 Z: 300\n"};
    auto res = valor_medio_gps(gps);
    cout << res << endl;
    return 0;
}
```

EJERCICIO 22

```
#include <iostream>
#include <string>
#include <map>

using namespace std;

ostream& operator<< (ostream& os, const map<string,int> &m) {
    for (const auto mm : m)
        os << mm.first << " -> " << mm.second << " minutos\n";
    return os;
}

map<string,int> retrasos_mes (const string &str) {
    map<string,int> res;
```

```
    const int tam_str =  str.size();
    int p=0,u;
    do {
        u = str.find(",", p);
        string nombre = str.substr(p,u-p);
        p=u+1;
        u = str.find("\n", p);
        int retraso = stoi (str.substr(p,u-p));
        res[nombre] += retraso;
        p=u+1;
     } while (p<tam_str);
    return res;
}

int main() {
    string alus {"Aitor Menta,12\nArmando Casitas,20\nAitor Menta,50\n"};
    auto res = retrasos_mes(alus);
    cout << res << endl;
    return 0;
}
```

EJERCICIOS 23 Y 24

Se deja como ejercicio. Con todo lo que se ha mostrado sobre procesamiento de strings, no deberían ser complejos.

Haga una función que reciba dos strings y devuelva si su contenido es igual o se encuentra a 1 "edición de distancia" (es decir, si hay que eliminar o añadir un carácter para que ambos strings sean iguales). Por ejemplo, "tiempo" y "timpo" están a 1 edición de distancia, al igual que "tiempo" y "tiempoX", pero "timpo" y "tiempoX" no lo están. Haga una función main con un ejemplo de utilización (uno correcto, otro incorrecto). Añada todas las funciones auxiliares que pudiera necesitar.

```
bool string_a_una_edicion (const string &str1, const string &str2) {
    if (str1 == str2) return true;
    const int tam1 = str1.size(), tam2 = str2.size();
    if (abs (tam1-tam2) >=2) return false;
    bool una_diferencia=false;
    if (tam1 == tam2) {
        return false;
    } else {
        const string &corto = (tam1<tam2? str1 : str2);
        const string &largo = (tam1<tam2? str2 : str1);
        int p=0;
        for (const auto &c : largo) {
            if (c!=corto[p]) {
                if (una_diferencia) return false;
                una_diferencia=true;
            } else ++p;
        }
    }
    return true;
}
```

Haga una función que reciba dos strings y devuelva un valor booleano que indique si uno es una permutación del otro. Si está vacío, debe retornar false. Por ejemplo, "telorta" y "artolet" son permutaciones. Haga una función main con un ejemplo de utilización (uno correcto, otro incorrecto). Añada todas las funciones auxiliares que pudiera necesitar.
2 soluciones: utilizando vector, map

```
map<char,int> letras (const string &str) {
    map<char,int> m;
    for (const auto &c : str) {
        if (m.count(c) == 0) m[c]=1;
        else ++m[c];
    }
    return m;
}

bool string_es_permutacion_map (const string &str1, const string &str2) {
    if (str1.empty() or str2.empty()) return false;
    if (str1.size() != str2.size()) return false;
    if ((&str1 == &str2) || (str1 == str2)) return true;
    map<char, int> m1 = letras (str1), m2 = letras(str2);
    if (m1 == m2) return true;
    return false;
}
```

```
struct c_datos {
  char l;
  int num;
};

vector<c_datos> letras_v (const string &str) {
    vector<c_datos> v;
    for (const auto &c : str) {
        bool repe = false;
        for (auto &vv : v) {
            if (vv.l == c) {
                repe = !repe;
                ++vv.num;
                break;
            }
        }
        if (!repe) v.push_back({c, 1});
    }
    return v;
}

bool string_es_permutacion_vector (const string &str1, const string &str2) {
    if (str1.empty() or str2.empty()) return false;
    if (str1.size() != str2.size()) return false;
    if ((&str1 == &str2) || (str1 == str2)) return true;
    vector<c_datos> v1 = letras_v (str1), v2 = letras_v(str2);
```

```
    for (const auto &vv1 : v1) {
        for (const auto &vv2 : v2) {
            if ((vv1.l == vv2.l) and (vv1.num != vv2.num)) return false;
        }
    }
    return true;
}
```

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {

  map<string, string> diccionario;
   string dia;

  diccionario["Monday"]    = "Lunes";
  diccionario["Tuesday"]   = "Martes";
  diccionario["Wednesday"] = "Miércoles";
  diccionario["Thursday"]  = "Jueves";
  diccionario["Friday"]    = "Viernes";
  diccionario["Saturday"]  = "Sábado";
  diccionario["Sunday"]    = "Domingo";

   cout << "Introduzca el día de la semana: ";
   cin >> dia;

   auto it = diccionario.find(dia);
   if (it != diccionario.end()) cout << diccionario[dia] << endl;
   else cout << "Día desconocido." << endl;
}
```

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main()
{

   map<int, string> passwords;
   int user;

   passwords[1] = "patata";
   passwords[3402] = "coles";
   passwords[501] = "deBruselas";

   cout << "User: ";
   cin >> user;

   auto it = passwords.find(user);
   if (it == passwords.end()) cout << "Unknown user." << endl;
   else
   {
      string password;
      cout << "password: ";
      cin >> password;
      if (password == passwords[user]) cout << "Access granted." << endl;
      else cout << "Access denied." << endl;
   }
}
```

Algoritmos de ordenar diferentes (describir cómo hacerlo pero no dar su nombre)

Algoritmos para desordenar vectores y strings, utilizando números aleatorios de <random> (2 versiones). In-place o que retornen nuevos vectores/strings
Hacer que el DNI afecte a alguno de los ejercicios anteriores. Por ejemplo: 1. Los valores que sean mayores o menores que el último dígito del DNI 2. Si el DNI es par, ordenar primero números pares y luego impares (da igual el orden relativo entre ellos)
Buscar en strings y sacar información. Del estilo "*Coord-x: XXX, Coord-y: XXX, Coord-z:XXX*" y lo mismo con Velocidades, u otras magnitudes físicas. Y guardarlos en un vector para luego hacer algún cálculo. O el string tiene formato CSV.
Separar un string grande en un vector de strings, mirando el '.' o un 'n' (Por líneas) O cada 'n' palabras
Sacar subconjunto de subarrays/substrings cuyos elementos consecutivos cumplen una propiedad

Union interseccion y diferencia / ejs string fáciles
Buscar en strings y sacar información. Del estilo "Coord-x: XXX, Coord-y: XXX, Coord-z:XXX" y lo mismo con Velocidades, u otras magnitudes físicas. Y guardarlos en un vector para luego hacer algún cálculo. O el string tiene formato CSV. Separar un string grande en un vector de strings, mirando el '.' o un 'n' (Por líneas) O cada 'n' palabras Hacer U/I/D pero con funciones que dependan de su DNI (si son divisibles por su último dígito, si son mayores que
Password según los criterios de PSP (letras mayusculas y minusculas y número) y una bbdd de passwrds Los de los sobcontjuntos

Separar un string grande en un vector de strings, suministrando el carácter de separación que se va a utilizar (exclusive). Los restos van a una última línea (aunque no cumplan)

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

```
using namespace std;

vector<string> separar_por_char (string &str, const char c) {
    vector<string> tmp;
    int p=0, u=0;
    bool continuar = true;
    do {
        while (str[u] != c && u<str.size()) ++u;
        tmp.push_back(str.substr(p, u-p));
        if (u<str.size()) {
            ++u;
            p=u;
        } else continuar = false;

    } while (continuar);

    return tmp;
}

void imprimir(const vector<string> &v) {
    for (const auto &vv : v)
        cout << '|' << vv << "|\n";
}

int main()
{
    string str = "hola. ADIOS. Tiempo 33 2342@#¢..LALA";
    auto res = separar_por_char(str, '.');
    imprimir(res);

    return 0;
}
```

Separar un string grande en un vector de strings, suministrando un número que indica la cantidad de palabras que deben guardarse en cada línea. Se supone que las palabras están separadas por un espacio en blanco. No es necesario eliminar los signos de puntuación (punto, coma, etc.). Los restos van a una última línea (aunque no cumplan). No se eliminan los espacios en blanco

```
vector<string> separar_por_num_palabras (string &str, const int num) {
  vector<string> tmp;
  auto p = begin(str), u = begin(str);
  int n=0;
  for (u; u!=end(str); u=next(u)) {
    if(isblank(*u)) {
      ++n;
      if (n==num) {
        n=0;
```

```
            tmp.emplace_back(p,u);
            p=u;
          }
        }
    }
    if(p!=end(str))
      tmp.emplace_back(p,u);
    return tmp;
}
```

El sensor GPS de la marca ACME devuelve una cadena de texto con el siguiente formato “X: <valor> Y: <valor> Z: <valor> \n". Se pide que haga un programa que acepte un string formado por varias cadenas como la anterior (siempre acabadas en ‘\n’), y devuelva un vector de 3 valores, que contenga la media de cada una de las tres coordenadas. Por ejemplo, si el string es string gps = "X: 1 Y: 2 Z: 3\n X: 10 Y: 20 Z: 30\nX: 100 Y: 200 Z: 300\n", la función debe devolver el vector con los valores {37 , 74, 111 }

```
vector<double> valor_medio_no_csv (const string &str) {
    vector<double> tmp {0,0,0};
    int p=0, u, num=0;
    do {
      p = str.find("X: ", p);
      p+=3;
      u = str.find("Y: ", p);
      tmp[0] += stod(str.substr(p, u-p));
      p=u+3;
      u = str.find("Z: ", p);
      tmp[1] += stod(str.substr(p, u-p));
      p=u+3;
      u = str.find("\n", p);
      tmp[2] += stod(str.substr(p, u-p));
      ++num;
    } while (u!=str.size()-1);
    tmp[0] /= num;
    tmp[1] /= num;
    tmp[2] /= num;
    return tmp;
}
```

Ahora con CSV

```
struct alumno {
    string nombre, ape;
    int d, m, a;
};
```

```
vector<alumno> procesar_csv (string str) {
   vector<alumno> res;
   istringstream iss {str};
   do{
      alumno alu;
      getline(iss,alu.nombre,',');
      getline(iss,alu.ape,',');
      string tmp;
      iss >> alu.d;
      iss.ignore(999,',');
      iss >> alu.m;
      iss.ignore(999,',');
      iss >> alu.a;
      res.push_back(alu);
   }while (!iss.eof());


   return res;
}
```