LENGUAJES DE PROGRAMACIÓN

Trabajo práctico de la convocatoria ordinaria (2023)

Autor
Yaiza Arnáiz Alcácer
16/04/23

Índice

1. Ejercicios

1.1. Ejercicio 1

El programa pedido en el ejercicio 1 debe ser capaz de convertir la temperatura en cualquiera de las tres unidades disponibles: K, C, F. Para ello lo primero que se debe de hacer es definir las funciones requeridas para realizar estas conversiones. En este caso necesitamos un minimo de 4 conversiones:

- Conversión de Celsius a Fahrenheit
- Conversión de Celsius a Kelvin
- Conversión de Fahrenheit a Celsius
- Conversión de Kelvin a Celsius

No es necesario definir 6 conversiones porque una vez convertimos Fahrenheit o Kelvin a Celsius, podemos utilizar las funciones definidas para convertir Celsius a la temperatura restante.

A continuación el código que he utilizado para realizar las conversiones:

Source Code 1: Funciones para conversión de unidades

```
#include <iostream>
using namespace std;

/* Primero defino cuatro funciones para realizar las diferentes conversiones*/
// conversion de celsius a fahrenheit
double conversion_c_f(double p) {
    double a;
    a = (p * 1.8) + 32 ;
    return a;
}
// conversion de celsius a kelvin
double conversion_c_k(double q) {
    double b;
    b = q + 273.15 ;
    return b;
}
// conversion de fahrenheit a celsius
double conversion_f_c(double r) {
    double c;
    c = (r / 1,8) - 32 ;
    return c;
}
//conversion de kelvin a celsius
double conversion_k_c(double s) {
    double d;
    d = s - 273.15 ;
    return d;
}
```

Después de definir estas 1 funciones básicas , podemos escribir el programa principal en *main*:

Source Code 2: Main

```
/* Ahora el programa principal usa las funciones definidas
con anterioridad para
realizar la funcion pedida de convertir cualquier temperatura
en cualquiera de las unidades C, F, K*/
int main () {
    // declaracion de variables
    double x;
    char y;
    // pedir al usuario la temperatura
    std::cout << "Escriba la temperatura con sus unidades (C, F, K):\t";
    std::cin >> x >> y;
    switch (y)
    {
    case 'C':  // si la temperatura es celsius
        if (x < -273.15 ){ // primero comprobar que el valor es valido para celsius
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Celsius la temperatura por debajo de -273.15 C no existe. ";
            std::cout << "Por favor intentelo de nuevo." ;
            break;
        }
        double fahrenheit_1;
        double kelvin_1;
        fahrenheit_1 = conversion_c_f(x); // realizar las conversiones
        kelvin_1 = conversion_c_k(x);
        std::cout << x << " C, " << fahrenheit_1 << " F, " << kelvin_1 << " K"; // output
        break;
    case 'F': // si la temperatura es fahrenheit
        if (x < -459.67){ // primero comprobar que el valor es valido para fahrenheit
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Fahrenheit la temperatura por debajo de -459.67 F no existe. ";
            std::cout << "Por favor intentelo de nuevo.";
            break;
        }
        double celsius_1;
        double kelvin_2;
        celsius_1 = conversion_f_c(x); // realizar la conversiones
        kelvin_2 = conversion_c_k(celsius_1);
        std::cout << celsius_1 << " C, " << x << " F, " << kelvin_2 << " K"; // output
        break;
    case 'K': // si la temperatura es kelvin
        if (x < 0){ // primero comprobar que el valor es valido para kelvin
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Kelvin la temperatura por debajo de 0 K no existe. ";
            std::cout <<  "Por favor intentelo de nuevo.";
            break;
        }
        double celsius_2;
        double fahrenheit_2;
        celsius_2 = conversion_k_c(x); // realizar la conversiones
        fahrenheit_2 = conversion_c_f(celsius_2);
        std::cout << celsius_2 << " C, " << fahrenheit_2 << " F, " << x << " K"; //output
```

```
52          break;
53      default:
54      /* Si el usuario no introdujo la temperatura en el formato correcto,
55       ya sea porque no introdujo un numero, se le olvidaron las unidades
56      o puso unidades incorrectas,
57      darle feedback al usuario para que use de nuevo el programa de forma correcta*/
58          std::cout << "Error! La temperatura introducida,";
59          std::cout << "no tiene el formato correcto (valor + unidades en C, F o K).\n";
60          std::cout << "Un ejemplo de uso seria: 24.2 C. \n";
61          std::cout << "Por favor, intentelo de nuevo.\n";
62          break;
63      }
64      return 0;
65  }
```

El primer paso a seguir es declarar las variables (líneas 7 y 8 2) y solicitar al usuario la temperatura.

Para decidir que conversiones hacer comenzamos un *switch* (línea 12), y definimos en cada caso:

- Comprobación de que la temperatura introducida tiene un valor válido desde el punto de vista de la Física.
- Realizamos las conversiones necesarias dependiendo de la unidad dada y escribimos el resultado en la consola.

De esta manera si un usuario decide introducir una temperatura de kelvin negativa, obtenemos el siguiente output en la consola:

```
Escriba la temperatura con sus unidades (C, F, K):      -90 K
Esta temperatura es fisicamente imposible.
En Kelvin la temperatura por debajo de 0 K no existe, por favor intentelo de
    nuevo.
```

Listing 1: Sálida de consola cuando se introduce una temperatura imposible

Para realizar esta comprobación y escribir el error en la consola utilizo un condicional *if* dentro de cada caso. Soy consciente de que se podría haber utilizado un solo condicional con *if* y dos *else if*, antes del *switch* para comprobar directamente el valor de la temperatura después del *input*. Sin embargo, creo que es más fácil de entender el código si se comprueba en cada caso. De esta manera, se entiende muy fácil que si es por ejemplo la temperatura es celsius, primero existe una comprobación y luego la conversión, y cada comprobación es a su vez específica para esa unidad e imprime un texto específico al valor de esa unidad.

A parte de realizar esta comprobación en cada caso, también se evitan errores de *input* en el default del *switch* (línea 54). Funciona tanto si se introduce el valor de las unidades erróneo o en minúsculas como si se introduce el valor de temperatura erróneo. Este es el ejemplo de salida en consola de algunos tests:

```
Escriba la temperatura con sus unidades (C, F, K):      hola C
Error! La temperatura introducida,no tiene el formato correcto (valor +
    unidades en C, F o K).
Un ejemplo de uso seria: 24.2 C.
```

```
Por favor, intentelo de nuevo.
```

Listing 2: Sálida de consola por error de input (test 1)

```
Escriba la temperatura con sus unidades (C, F, K):      54 y
Error! La temperatura introducida,no tiene el formato correcto (valor +
    unidades en C, F o K).
Un ejemplo de uso seria: 24.2 C.
Por favor, intentelo de nuevo.
```

Listing 3: Sálida de consola por error de input (test 2)

```
Escriba la temperatura con sus unidades (C, F, K):      98 c
Error! La temperatura introducida,no tiene el formato correcto (valor +
    unidades en C, F o K).
Un ejemplo de uso seria: 24.2 C.
Por favor, intentelo de nuevo.
```

Listing 4: Sálida de consola por error de input (test 3)

Por último, para comprobar que el programa funciona y produce correctamente las conversiones, utilice la temperatura ejemplo (25.4 C) dada en el enunciado del ejercicio. En la siguiente salida de consola podemos ver que las conversiones fueron realizadas de forma correcta:

```
Escriba la temperatura con sus unidades (C, F, K):      25.4 C
25.4 C, 77.72 F, 298.55 K
```

Listing 5: Sálida de consola por ejecución correcta del programa

1.2. Ejercicio 2

En este ejercicio, el resultado del programa debe ser la suma y multiplicación de las matrices respectivas de dos numeros complejos provistos por el usuario.

Para este programa he definido mútiples funciones 4:

- Función para sumar ambas matrices (línea 16, 4)
- Función para multiplicar ambas matrices (línea 25, 4)
- Función para imprimir en consola las matrices (línea 2, 4)
- Función para imprimir en consola los números complejos (línea 1, 4)

Las dos primeras funciones deben ser capaces de devolver una matriz, esto se puede hacer bien usando punteros o estructuras. En mi caso decidi usar estructuras. Por eso, en primer lugar definí la estructura de un *array* 3:

Source Code 3: Definición de estructura para el *array*

```
#include <iostream>
#include <complex>
using namespace std;

/* Definir constante. En este caso estamos trabajando con numeros complejos.
```

```
Esto significa que las matrices siempre seran de 2x2*/
const int N = 2;

// definir la estructura del array
struct Struct_array {
    double arr[N][N];
};
```

En línea 7 3 definé también la constante para crear la estructura teniendo en cuenta que un número complejo tendrá siempre una matriz 2x2.

Una vez definida esta estructura, se pueden crear las funciones ya mencionadas para las diferentes operaciones 4:

Source Code 4: Funciones para la suma, multiplicación e impresión en consola de las matrices

```
//crear funcion para imprimir complejo
void print_complex(double real, double imaginaria, string desc){
    std::cout << desc << real;
    if (imaginaria >= 0) {
        std::cout << " + ";
    } else {
        std::cout << " - ";
    }
    std::cout << std::abs(imaginaria) << "i" << std::endl;
}

// crear funcion para imprimir matriz
void print_matrix(Struct_array array, double real, double imaginaria){
    if (imaginaria >= 0){
    cout << "\t--Matrix de " << real << "+" << imaginaria << "i--\n";
    }
    else{cout << "\t--Matrix de " << real << imaginaria << "i--\n";
    }
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            cout<<"\t"<<array.arr[i][j] <<"\t";
        }
        cout<<endl;
    }
    cout<<endl;
}

// crear funcion para sumar las dos matrices
Struct_array sum_matrix(Struct_array p, Struct_array q){
    Struct_array result_sum;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result_sum.arr[i][j] = p.arr[i][j] + q.arr[i][j];
        }
    }
    return result_sum;
}

```

```
// crear funcion para multiplicar las dos matrices
Struct_array product_matrix(Struct_array n, Struct_array m){
    Struct_array result_product;
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            result_product.arr[i][j] = 0;
            for (int k = 0; k < N; k++){
                result_product.arr[i][j] += n.arr[i][k] * m.arr[k][j];

            }

        }

    }
    return result_product;
}
```

Después de estas definiciones, podemos escribir el programa 5 en el que seguí los siguientes pasos:

- Pedir al usuario los dos numeros complejos (líneas 8-11, 5).
- Comprobar *input* del usuario y devolver error en caso de input incorrecto (líneas 7-18, 5).
- Declarar los complejos como numeros complejos usando la librería *complex* de C++ (línea 19, 5).
- Calcular la suma y producto usando la librería *complex* de C++ (líneas 23-25, 5).
- Declarar las matrices de los complejos (líneas 29-30, 5).
- Realizar la suma y producto usando las matrices (líneas 32-33, 5).
- Imprimir en consola los resultados (líneas 21-27 / 34-43, 5).

En el código:

Source Code 5: Main

```
int main () {
    std::cin.exceptions(std::ios_base::failbit);
    // declaracion de variables
    double a, b, c, d; /*vease que se piden dos doubles por cada numero
    complejo por ser mas intuitivo que pedir el complejo con su formato correcto
    (entre parentesis y separando parte real e imaginaria con una coma)*/
    try{ // comprobar que el usuario introduce correctamente los numeros complejos
    std::cout << "Introduzca el primer numero complejo (valor real + valor imaginario):\t";
    std::cin >> a >> b;
    std::cout << "Introduzca el segundo numero complejo (valor real + valor imaginario):\t";
    std::cin >> c >> d;
    } // si no lo hace, lanzar error y finalizar
    catch(std::ios_base::failure &error){
```

```
14          std::cout << "Error! El numero complejo no tiene el formato correcto\n";
15          std::cout << "Ejemplo de entrada: 2.0 4.55 \n";
16          std::cout << "Intentelo de nuevo.";
17          throw;
18      }
19      std::complex<double> x(a,b), y(c,d); // ahora declarar los complejos
20      // Hacer las operaciones y escribir el resultado en consola
21      print_complex(x.real(), x.imag(), "El primer numero complejo introducido es: ");
22      print_complex(y.real(), y.imag(), "El segundo numero complejo introducido es: " );
23      std::complex<double> sum = x + y;
24      print_complex(sum.real(), sum.imag(), "La suma de ambos numeros complejos es: " );
25      std::complex<double> product = x * y;
26      print_complex(product.real(), product.imag()
27      , "El producto de ambos numeros complejos es: " );
28      // Declarar matrices por cada uno de los numeros complejos
29      Struct_array matrix_1 = {real(x), -imag(x), imag(x), real(x)};
30      Struct_array matrix_2 = {real(y), -imag(y), imag(y), real(y)};
31      // Realizar las operaciones con las matrices y escribit el resultado en consola
32      Struct_array suma = sum_matrix(matrix_1, matrix_2);
33      Struct_array producto = product_matrix(matrix_1, matrix_2);
34      std::cout << "La suma y el producto de dos numeros complejos puede calcularse sumando";
35      std::cout << "y multiplicando sus respectivas matrices\n";
36      std::cout << "La matriz correspondiente al primer numero complejo es:\n";
37      print_matrix(matrix_1, real(x), imag(x));
38      std::cout << "La matriz correspondiente al segundo numero complejo es:\n";
39      print_matrix(matrix_2, real(y), imag(y));
40      std::cout << "La matriz resultado de esta suma es:\n";
41      print_matrix(suma, suma.arr[0][0], suma.arr[1][0]);
42      std::cout << "La matriz resultado del producto es:\n";
43      print_matrix(producto, producto.arr[0][0], producto.arr[1][0]);
44    return 0;
45  }
```

Cuando pido al usuario los números complejos, soy consciente de que se podrían haber pedidos directamente declarados como complejos, pero la razón por la que decidí definir 4 *doubles* es porque considero que es más intuitivo para un usuario introducir dos números seguidos por un espacio para determinar la parte real y la parte imaginaria de su número que introducirlos entre paréntesis y separados por una coma. Puede que un usuario que sepa C++, sabrá como introducir el número en el formato de complejo en la consola, pero en general habrá personas que no lo sepan.

Sin embargo, al ser más intuitivo también es ligeramente peor para la memoria ya que se necesistan declarar 6 variables en lugar de dos. En primer lugar se declaran las cuatro variables *double*, para los dos números con su parte real y su parte imaginaria y en segundo lugar se declaran dos números complejos usando los cuatro double dados por el usuario. Aún así, son solo seis variables y el programa es muy pequeño, por lo que creo que es totalmente correcto en este caso ocupar más memoria pero tener un programa más intuitivo.

En cuánto a la comprobación del *input* o entrada de los números en consola he usado un *try, catch and throw* en líneas 7-18, 5. He utilizado en este caso `ios_base::failbit`, porque este *flag* se activa cuando esperamos un double pero recibimos una letra por ejemplo, en el flujo `cin`. Luego en el *catch*, podemos emitir una excepción del tipo `ios_base::failure`

en el caso de la activación del `ios_base::failbit` en el flujo `cin`. De esta forma puedo ver que el usuario ha introducido correctamente los números antes de declarar los números complejos.

Un ejemplo de la salida por consola cuando la entrada es incorrecta sería:

```
Introduzca el primer numero complejo (valor real + valor imaginario):   4.56
    0.5
Introduzca el segundo numero complejo (valor real + valor imaginario):  3.1 h
Error! El numero complejo no tiene el formato correcto
Ejemplo de entrada: 2.0 4.55
Intentelo de nuevo.
```

Listing 6: Sálida de consola por error en entrada

Como se puede ver, la parte imaginaria del segundo numero es una letra, lo que activa la excepción y devuelve el error.

Finalmente, en cuánto a las operaciones decidía realizarlas con la librería *complex* aparte de con las matrices para tener una forma de comprobar que el resultado con las matrices es correcto.

Si ejecutamos el programa con las matrices ejemplo dadas en el enunciado del ejercicio, la salida sería la siguiente:

```
Introduzca el primer numero complejo (valor real + valor imaginario):   5 7
Introduzca el segundo numero complejo (valor real + valor imaginario):  1 -8
El primer numero complejo introducido es: 5 + 7i
El segundo numero complejo introducido es: 1 - 8i
La suma de ambos numeros complejos es: 6 - 1i
El producto de ambos numeros complejos es: 61 - 33i
La suma y el producto de dos numeros complejos puede calcularse sumandoy
    multiplicando sus respectivas matrices
La matriz correspondiente al primer numero complejo es:
        --Matrix de 5+7i--
        5               -7
        7               5

La matriz correspondiente al segundo numero complejo es:
        --Matrix de 1-8i--
        1               8
        -8              1

La matriz resultado de esta suma es:
        --Matrix de 6-1i--
        6               1
        -1              6

La matriz resultado del producto es:
        --Matrix de 61-33i--
        61              33
        -33             61
```

Listing 7: Sálida de consola

Como se puede observar los resultados son correctos.

Además el programa incluye condicionales en las funciones para imprimir la salida en consola correctamente. Por ejemplo, en el caso de dar un numero imaginario negativo como es el caso, no imprime $1 + -8i$ sino $1 - 8i$.

Una de mis preocupaciones al respecto de la salida en consola es cuando el usuario da como numero imaginario el 0. Al principio pensé en añadir otro condicional para eliminar de la salida en consola la parte imaginaria, sin embargo determiné que esto al final no era lo que quería ya que quiero que se vea claramente que sigue siendo un número complejo con su parte imaginaria. En otras palabras, quiero que se siga representando como número complejo. Así, utilizando un 0 en la parte imaginaria, el output se vería:

```
1  Introduzca el primer numero complejo (valor real + valor imaginario):   5 7
2  Introduzca el segundo numero complejo (valor real + valor imaginario):  1 0
3  El primer numero complejo introducido es: 5 + 7i
4  El segundo numero complejo introducido es: 1 + 0i
5  La suma de ambos numeros complejos es: 6 + 7i
6  El producto de ambos numeros complejos es: 5 + 7i
7  La suma y el producto de dos numeros complejos puede calcularse sumandoy
       multiplicando sus respectivas matrices
8  La matriz correspondiente al primer numero complejo es:
9          --Matrix de 5+7i--
10         5               -7
11         7               5
12
13 La matriz correspondiente al segundo numero complejo es:
14         --Matrix de 1+0i--
15         1               -0
16         0               1
17
18 La matriz resultado de esta suma es:
19         --Matrix de 6+7i--
20         6               -7
21         7               6
22
23 La matriz resultado del producto es:
24         --Matrix de 5+7i--
25         5               -7
26         7               5
```

Listing 8: Sálida de consola (ejemplo 2)

1.3. Ejercicio 3

En este ejercicio se pedía leer un fichero con información de la actividad de unas máquinas, recuperar esta información, realizar el cálculo del número de actividades y número de minutos y devolver en la consola la información en orden de menor a mayor según el número de indentificación de la máquina.

Para ello en primer lugar declare dos constantes globales (líneas 12-13, 6), que son el nombre del fichero y el prefijo al número identificador de la máquina, en este caso “M”. Así mismo para recuperar la información y realizar las operaciones quiero usar un mapa de estructuras. De esta manera definí las variables correspondientes a los diferentes datos del fichero así como la estructura (líneas 23-29, 6) que voy a utilizar más adelante.

Source Code 6: Definición de variables

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <chrono>
#include <iomanip>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

//Definir las constantes globales
const string filename = "diarioActividad.txt";
const string prefijo = "M";

// Crear variables para obtener los diferentes datos del fichero
string id;
string inicio;
string final;
string tipo;

/*Quiero trabajar con un diccionario (mapa) de estructuras, para ello
primero debo de definir la estructura que usaré*/
struct maquina{
    int id; // el id sera un numero entero para poder usarlo luego para sortear por orden
    int auto_n;
    int manual_n;
    int error_n;
    int tiempo_auto;
};
```

La idea de utilizar un mapa de estructuras, viene de la posibilidad de no repetir las estructuras, ya que cada clave es única. De esta manera, se facilita la posibilidad de hacer los cálculos leyendo el archivo línea por línea una sola vez. Sin embargo, la dificultad que encontré es el hecho de que las claves de los mapas deben ser tipo *string*. Al ser así, aunque el mapa se imprime en consola directamente ordenado lo hará de comparando las claves, lo que da lugar por ejemplo a imprimir primero "M10" y segundo "M5", cuando 5 es menor que 10.

Para evitarlo necesitaba definir una función que actuará como comparador de ids de las máquinas 7:

Source Code 7: Función comparador

```cpp
struct comparador {
    bool operator()(const pair<string, maquina>& x, const pair<string, maquina>& y) const {
        return x.second.id < y.second.id;
    }
}; //gracias a esta función podremos ordenar por id
```

Esta función usa el identificador de cada estructura que en este caso sera un entero, puesto que eliminó la M y lo transformó más adelante en *main*.

Por otro lado, también necesitaba una función que fuera capaz de calcular la duración en minutos utilizando como argumentos los tiempos en formato HH:MM 8. Para esto utilicé la librería *chrono*.

Source Code 8: Función para calcular la duración en minutos

```
// Definir una función para calcular la duración:
int calcular_duracion(string tiempo_1, string tiempo_2){
    tm t_1{}, t_2{};

    istringstream stream_1(tiempo_1);
    istringstream stream_2(tiempo_2);

    stream_1>> get_time(&t_1, "%H:%M");
    stream_2 >> get_time(&t_2, "%H:%M");

    auto empieza = chrono::hours(t_1.tm_hour) + chrono::minutes(t_1.tm_min);
    auto acaba = chrono::hours(t_2.tm_hour) + chrono::minutes(t_2.tm_min);

    int duracion = (acaba - empieza).count();

    return duracion;
}
```

Una vez definidas estas funciones, el programa *main* 9 sigue los siguientes pasos:

- Abrir el archivo (líneas 4-6, 9).
- Comprobar que el archivo no esta corrupto y se puede leer (líneas 8-9, 9).
- Crear el mapa de estructuras (línea 13, 9).
- Crear *loop* para leer línea por línea y recoger la información en las variables tipo *string* (línea 17, 9).
- Recoger información del número identificador como entero dentro de la estructura (líneas 23-25, 9).
- Utilizar condicional para recoger la información del número de veces en cada actividad y calcular los minutos en auto con la función para calcular la duración (líneas 29-37, 9).
- Introducir la estructura en el diccionario (línea 39, 9).
- Crear vector a partir del diccionario para sortearlo usando la función que compara las estructuras por id (líneas 52-54, 9).
- Imprimir en consola la información (líneas 56-64, 9).

Podemos ver el código aquí:

Source Code 9: Main

```
int main(){
    /*Leemos el archivo y utilizamos una condicion para comprobar que el fichero
    puede ser leido y en caso de no ser asi devolver el error*/
    ifstream input;

    input.open(filename);

    if(!input.is_open()){
        cout << "Error, no es posible leer el fichero " << filename << std::endl;
        return 1;
    }

    map<string, maquina> maquinas; //Crear el mapa de estructuras
    int count = 0;
    /* definir constante para comprobar que el codigo funciona
     sobre todas las lineas del archivo*/
    while(input>>id>>inicio>>final>>tipo){ //loop para leer linea por linea

        struct maquina m = maquinas[id]; //declarar la estructura en el mapa

        input.get();

        string numero;
        numero = id.substr(prefijo.length());
        m.id = stoi(numero); // obtener el id en forma de entero

        /*Condiciones para obtener la duracion en auto
        y el numero total en auto, manual y error*/
        if (tipo == "AUTO"){
            m.tiempo_auto += calcular_duracion(inicio, final);
            m.auto_n += 1;
        }
        else if (tipo == "MANUAL"){
            m.manual_n +=1;
        }
        else if (tipo == "ERROR") { m.error_n +=1;
        }

       maquinas[id] = m; //introducir la estructura en el diccionario (mapa)

       count++;

       if(!input){ // comprobar que no ha habido errores en la lectura
            break;
        }


    }

    cout << "Numero de lineas leidas en el fichero " << count << endl;
    // para sortear el diccionario creamos un vector
    vector<pair<string, maquina>> v(maquinas.begin(), maquinas.end());
    // utilizar el comparador de estructuras para ordenar por id
    sort(v.begin(), v.end(), comparador());
    size_t size = v.size();
    cout << "Numero de maquinas: " << size << endl;
```

```
57      // Imprimir en consola toda la informacion obtenida
58      for (const auto& key : v) {
59          cout << "Nombre: " << key.first << ", id: " << key.second.id
60                   << ", total en auto: " << key.second.auto_n  << ", total en manual: "
61                   << key.second.manual_n << ", total en error " << key.second.error_n
62                   << ", tiempo en auto: " << key.second.tiempo_auto
63                   << " min" << endl;
64      }
65
66      input.close(); // cerrar el archivo
67      return 0;
68  }
```

La lógica del programa es hacer uso del mapa para incrementar el valor del número de actividades y cantidad en minutos cada vez que leemos una misma máquina e introducir una nueva estructura cada vez que leemos una nueva máquina, ya que los elementos del mapa no pueden repetirse. Luego para imprimir la información obtenida en el orden pedido, se convierte el mapa en un vector y se sortea.

Soy consciente de que al crear un mapa y un vector, se esta almacenando dos veces la información y esto podría ser contraproducente para la memoria. Pero en este caso, el número máximo de estructuras es 100, por lo que la memoria en este caso no es un problema.

Además el código también usa una variable contador llamada `count` (líneas 14 y 41, 9) que me permite comprobar que el programa ha leído todas las líneas del fichero. A parte, también uso un condicional `if(!input)` (línea 43, 9)para comprobar que se ha alcanzado el final del fichero y evitar que el bucle continúe usando líneas en blanco.

Finalmente este sería el resultado del programa, usando el fichero ejemplo dado en el enunciado:

```
1 Numero de lineas leidas en el fichero 22
2 Numero de maquinas: 5
3 Nombre: M2, id: 2, total en auto: 2, total en manual: 2, total en error 0,
      tiempo en auto: 369 min
4 Nombre: M15, id: 15, total en auto: 2, total en manual: 0, total en error 0,
      tiempo en auto: 105 min
5 Nombre: M53, id: 53, total en auto: 4, total en manual: 3, total en error 2,
      tiempo en auto: 253 min
6 Nombre: M62, id: 62, total en auto: 0, total en manual: 0, total en error 1,
      tiempo en auto: 0 min
7 Nombre: M64, id: 64, total en auto: 2, total en manual: 4, total en error 0,
      tiempo en auto: 77 min
```

Listing 9: Sálida de consola

1.4. Ejercicio 4

Siguiendo las instrucciones del ejercicio, el primer paso fue definir las constantes globales `MODO_DEBUG` y `N` 10.

Después, cree un array de estructuras para almacenar los estados y el número entre 0 y 1 que funciona como criterio para el paso de un estado a otro.

Asimismo también definí una estructura para almacenar los datos que se quieren imprimir.

Source Code 10: Definición de variables y estructuras

```
#include <iostream>
#include <random>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

// definir las constantes globales
const bool MODO_DEBUG = true;
const int N = 10;

// crear estructura para recoger los estados y el numero entre 0 y 1
struct Tabla_de_estados
{
    int estado_S_1;
    int random_numero;
    int estado_S_2;
};
// crear array de estructuras con los datos de los estados y sus transiciones
static const Tabla_de_estados t[]=
{
    { 0, 0, 2},
    { 0, 1, 1},
    { 1, 0, 1},
    { 1, 1, 4},
    { 2, 0, 5},
    { 2, 1, 3},
    { 3, 0, 6},
    { 3, 1, 7},
    { 4, 0, 5},
    { 4, 1, 7},
    { 5, 0, 5},
    { 5, 1, 1},
    { 6, 0, 6},
    { 6, 1, 7},
    { 7, 0, 7},
    { 7, 1, 7},
};

// crear estructura de los datos que queremos imprimir en consola
struct informe{
    int n_transicion;
    int replicas = 0;
};
```

Mi idea principal era usar un array de estructuras para almacenar la información del diagrama proporcionada en el enunciado y luego usar una doble condición *if* para encontrar en el array, la estructura que coincida con el estado inicial y el número obtenido de forma pseudorandom. Sin embargo, me di cuenta de que no es necesario utilizar un doble condi-

cional porque se puede acceder al indice de la estructura correspondiente multiplicando el estado inicial por dos y sumandole el número obtenido de forma pseudorandom.

Permiteme explicar esto con un ejemplo:

En el array de estructuras, cada estructura contiene el número de estado inicial, el número random y el número del segundo estado (líneas 20-38, 10). Así si estamos en el estado 1, y el número random es 1, pasaríamos al estado 4, esto representa la estructura de índice 3 en el *array*. Es tan sencillo como multiplicar en este caso el estado $(1 * 2) + 1 = 3$, y luego seleccionar el segundo estado correspondiente a la estructura almacenada en el *array* en el índice 3, lo que nos daría el estado 4.

Para resolverlo con condicionales, podríamos usar el doble condicional:

```
if(estado==estado_S_1 & random==random_numero)
```

,pero esto consume mucho más tiempo.

En este sentido, una vez que me di cuenta de que se podía calcular directamente el índice podría haber cambiado el *array* de estructuras a un *array* unidimensional que solo almacene los números correspondientes al segundo estado, sin embargo, creo que el código es mucho más fácil de entender con el array de estructuras, asique decidí no cambiarlo.

Así para calcular el índice cree una función y recicle la función comparador del ejercicio 3, ya que para imprimir las transiciones de estado voy a utilizar la misma idea que use en el ejercicio 3 con el diccionario o mapa de estructuras. Se puede ver el código aquí 11:

Source Code 11: Funciones

```
1  // crear funcion para calcular el index al que ir y hacer el codigo más rápido y eficiente
2  int index(int a, int b){
3      int result = a*2 + b;
4      return result;
5  }
6
7
8  // crear función para comparar y sortear por numero de transiciones
9  struct comparador {
10     bool operator()(const pair<string, informe>& x, const pair<string, informe>& y) const {
11         return x.second.n_transicion< y.second.n_transicion;
12     }
13 };
```

Teniendo en cuenta lo anterior, mi programa en *main* sigue los siguientes pasos:

- Definir las distintas variables necesarias para obtener el número pseudorandom, el *array* para almacenar las transiciones y el mapa de estruturas (líneas 6-10, 12) .
- Crear un bucle `for` que me permita realizar cada experimento N veces (línea 12, 12).
- Inicializar el estado y la transicion a 0 en dos distintas variables (líneas 13-14, 12).
- Crear un bucle `while` para realizar el experimento hasta que se alcance el estado 7 (líneas 18, 12).
- Obtener el número pseudorandom entre 0 y 1 y redondearlo para obtener 0 o 1 (líneas 19-20, 12).

- Calcular el índice (línea 21, 12).
- Acceder y actualizar el estado utilizando el índice (línea 22, 12).
- Almacenar transicion (líneas 23, 12).
- Salir del bucle y añadir la transición al *array* de transiciones (línea 29, 12).
- Salir del bucle y usar otro para leer el *array* de transiciones (línea 35, 12).
- Añadir a la estructura el número de transiciones y contar el número de replicas de cada experimento con las mismas transiciones (líneas 37-38, 12).
- Añadir la estructura al diccionario (línea 39, 12).
- Salir del bucle y convertir el diccionario a vector y sortearlo usando la función *Comparador* (líneas 44-47, 12).
- Imprimir en consola los datos (líneas 48-51, 12).

En código 11:

Source Code 12: main

```
1  int main(){
2      if (MODO_DEBUG){cout << "Evolucion del automata en " << N << " replicas" << endl;}
3      /* Las condiciones
4      ayudaran a imprimir en caso de de estar en modo debug TRUE*/
5      // Definir las distintas variables:
6      random_device rd;
7      mt19937 gen(rd());
8      uniform_real_distribution<double> dis(0.0, 1.0);
9      int T[N];
10     map<string, informe> informes;
11     // loop para realizar N veces el experimento
12     for(int i= 0; i< N; i++){
13         int estado = 0;
14         int transicion = 0;
15         if (MODO_DEBUG){cout << "S" << estado;}
16
17
18         while(estado != 7){ // loop para realizar el experimento
19             double random = dis(gen);
20             int number = round(random); // definir numero pseudorandom entre 0 y 1
21             int n = index(estado, number); // calcular el index del array
22             estado = t[n].estado_S_2; // update el estado
23             transicion += 1; // recoger numero de transiciones
24             if (MODO_DEBUG){cout << ", " << number << ", " << "S" << estado;}
25
26
27
28         }
29         T[i] = transicion; // recoger transiciones en array
30         if (MODO_DEBUG){cout << endl;}
31
32     }
```

```
33
34      if (MODO_DEBUG){cout << "\n" << "Contenido del vector T:" << endl;}
35      for (int t: T){
36          struct informe z = informes[to_string(t)];
37          z.n_transicion = t;
38          z.replicas += 1; // popular las estructuras del diccionario
39          informes[to_string(t)] = z; // popular el diccionario
40          if (MODO_DEBUG){cout << t << " ";}
41      }
42      if (MODO_DEBUG){cout << "\n\n";}
43      cout << "Informe sobre el numero de transiciones:" << endl;
44      vector<pair<string, informe>> v(informes.begin(), informes.end());
45      /*crear vector para sortear
46      por numero de transiciones*/
47      sort(v.begin(), v.end(), comparador());
48      for (const auto& key : v) { // imprimir en consola los datos recogidos.
49          cout << key.second.n_transicion << " transiciones en "
50               << key.second.replicas << " replicas" << endl;
51      }
52
53      return 0;
54  }
```

Además, el código también incluye tres condicionales más para imprimir los datos adicionales requeridos cuando `MODO_DEBUG = true`.

Tál y como se pide en el enunciado, esta sería la salida en consola para `MODO_DEBUG=true, N=10`:

```
Evolucion del automata en 10 replicas
S0, 1, S1, 1, S4, 0, S5, 0, S5, 1, S1, 0, S1, 0, S1, 1, S4, 1, S7
S0, 1, S1, 1, S4, 1, S7
S0, 0, S2, 0, S5, 0, S5, 0, S5, 0, S5, 1, S1, 1, S4, 0, S5, 0, S5, 0, S5, 1,
    S1, 1, S4, 0, S5, 0, S5, 1, S1, 1, S4, 0, S5, 0, S5, 0, S5, 1, S1, 1, S4,
    0, S5, 1, S1, 1, S4, 0, S5, 1, S1, 1, S4, 0, S5, 1, S1, 0, S1, 0, S1, 1,
    S4, 1, S7
S0, 1, S1, 0, S1, 1, S4, 1, S7
S0, 1, S1, 0, S1, 0, S1, 1, S4, 0, S5, 0, S5, 0, S5, 1, S1, 1, S4, 0, S5, 1,
    S1, 0, S1, 1, S4, 1, S7
S0, 0, S2, 1, S3, 0, S6, 1, S7
S0, 1, S1, 0, S1, 0, S1, 0, S1, 0, S1, 0, S1, 0, S1, 0, S1, 1, S4, 0, S5, 0,
    S5, 1, S1, 1, S4, 0, S5, 0, S5, 0, S5, 0, S5, 0, S5, 1, S1, 1, S4, 1, S7
S0, 1, S1, 1, S4, 1, S7
S0, 1, S1, 1, S4, 0, S5, 1, S1, 1, S4, 0, S5, 0, S5, 0, S5, 0, S5, 1, S1, 0,
    S1, 0, S1, 1, S4, 0, S5, 1, S1, 0, S1, 0, S1, 0, S1, 0, S1, 1, S4, 1, S7
S0, 1, S1, 0, S1, 0, S1, 0, S1, 1, S4, 1, S7

Contenido del vector T:
9 3 33 4 14 4 21 3 21 6

Informe sobre el numero de transiciones:
3 transiciones en 2 replicas
4 transiciones en 2 replicas
6 transiciones en 1 replicas
9 transiciones en 1 replicas
14 transiciones en 1 replicas
21 transiciones en 2 replicas
```

```
33 transiciones en 1 replicas
```

Listing 10: Sálida de consola

y este sería para `MODO_DEBUG=false, N=10000`:

```
Informe sobre el numero de transiciones:
3 transiciones en 2488 replicas
4 transiciones en 1266 replicas
5 transiciones en 942 replicas
6 transiciones en 847 replicas
7 transiciones en 618 replicas
8 transiciones en 502 replicas
9 transiciones en 406 replicas
10 transiciones en 369 replicas
11 transiciones en 328 replicas
12 transiciones en 277 replicas
13 transiciones en 258 replicas
14 transiciones en 213 replicas
15 transiciones en 164 replicas
16 transiciones en 176 replicas
17 transiciones en 155 replicas
18 transiciones en 109 replicas
19 transiciones en 108 replicas
20 transiciones en 85 replicas
21 transiciones en 89 replicas
22 transiciones en 69 replicas
23 transiciones en 74 replicas
24 transiciones en 55 replicas
25 transiciones en 46 replicas
26 transiciones en 43 replicas
27 transiciones en 33 replicas
28 transiciones en 32 replicas
29 transiciones en 30 replicas
30 transiciones en 35 replicas
31 transiciones en 18 replicas
32 transiciones en 20 replicas
33 transiciones en 22 replicas
34 transiciones en 14 replicas
35 transiciones en 12 replicas
36 transiciones en 14 replicas
37 transiciones en 12 replicas
38 transiciones en 7 replicas
39 transiciones en 7 replicas
40 transiciones en 9 replicas
41 transiciones en 8 replicas
42 transiciones en 6 replicas
43 transiciones en 2 replicas
44 transiciones en 2 replicas
45 transiciones en 2 replicas
46 transiciones en 5 replicas
47 transiciones en 2 replicas
48 transiciones en 2 replicas
49 transiciones en 1 replicas
50 transiciones en 2 replicas
51 transiciones en 7 replicas
52 transiciones en 1 replicas
53 transiciones en 1 replicas
```

```
54 transiciones en 1 replicas
58 transiciones en 1 replicas
59 transiciones en 1 replicas
61 transiciones en 1 replicas
65 transiciones en 1 replicas
66 transiciones en 1 replicas
73 transiciones en 1 replicas
```

Listing 11: Sálida de consola

Me parece interesante llamar la atención al hecho de que el número más probable de transiciones necesarias para llegar al estado siete es 3. Esto tiene mucho sentido ya que de acuerdo al diagrama de estado, en dos ocasiones es posible llegar al estado 7 solo con tres transiciones.

1.5. Appendix

1.5.1. Código ejercicio 1

```
#include <iostream>
using namespace std;

/* Primero defino cuatro funciones para realizar las diferentes conversiones*/
// conversion de celsius a fahrenheit
double conversion_c_f(double p) {
    double a;
    a = (p * 1.8) + 32 ;
    return a;
}
// conversion de celsius a kelvin
double conversion_c_k(double q) {
    double b;
    b = q + 273.15 ;
    return b;
}
// conversion de fahrenheit a celsius
double conversion_f_c(double r) {
    double c;
    c = (r / 1,8) - 32 ;
    return c;
}
//conversion de kelvin a celsius
double conversion_k_c(double s) {
    double d;
    d = s - 273.15 ;
    return d;
}


/* Ahora el programa principal usa las funciones definidas
con anterioridad para
realizar la funcion pedida de convertir cualquier temperatura
en cualquiera de las unidades C, F, K*/
int main () {
    // declaracion de variables
    double x;
    char y;
    // pedir al usuario la temperatura
    std::cout << "Escriba la temperatura con sus unidades (C, F, K):\t";
    std::cin >> x >> y;
    switch (y)
    {
    case 'C':  // si la temperatura es celsius
        if (x < -273.15 ){ // primero comprobar que el valor es valido para celsius
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Celsius la temperatura por debajo de -273.15 C no existe. ";
            std::cout << "Por favor intentelo de nuevo." ;
            break;
        }
        double fahrenheit_1;
        double kelvin_1;
```

```
        fahrenheit_1 = conversion_c_f(x); // realizar las conversiones
        kelvin_1 = conversion_c_k(x);
        std::cout << x << " C, " << fahrenheit_1 << " F, " << kelvin_1 << " K"; // output
        break;
    case 'F': // si la temperatura es fahrenheit
        if (x < -459.67){ // primero comprobar que el valor es valido para fahrenheit
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Fahrenheit la temperatura por debajo de -459.67 F no existe. ";
            std::cout << "Por favor intentelo de nuevo.";
            break;
        }
        double celsius_1;
        double kelvin_2;
        celsius_1 = conversion_f_c(x); // realizar la conversiones
        kelvin_2 = conversion_c_k(celsius_1);
        std::cout << celsius_1 << " C, " << x << " F, " << kelvin_2 << " K"; // output
        break;
    case 'K': // si la temperatura es kelvin
        if (x < 0){ // primero comprobar que el valor es valido para kelvin
            std::cout << "Esta temperatura es fisicamente imposible. \n";
            std::cout << "En Kelvin la temperatura por debajo de 0 K no existe. ";
            std::cout <<  "Por favor intentelo de nuevo.";
            break;
        }
        double celsius_2;
        double fahrenheit_2;
        celsius_2 = conversion_k_c(x); // realizar la conversiones
        fahrenheit_2 = conversion_c_f(celsius_2);
        std::cout << celsius_2 << " C, " << fahrenheit_2 << " F, " << x << " K"; //output
        break;
    default:
    /* Si el usuario no introdujo la temperatura en el formato correcto,
     ya sea porque no introdujo un numero, se le olvidaron las unidades
    o puso unidades incorrectas,
    darle feedback al usuario para que use de nuevo el programa de forma correcta*/
        std::cout << "Error! La temperatura introducida,";
        std::cout << "no tiene el formato correcto (valor + unidades en C, F o K).\n";
        std::cout << "Un ejemplo de uso seria: 24.2 C. \n";
        std::cout << "Por favor, intentelo de nuevo.\n";
        break;
    }
    return 0;
}
```

1.5.2. Código ejercicio 2

```
#include <iostream>
#include <complex>
using namespace std;

/* Definir constante. En este caso estamos trabajando con numeros complejos.
Esto significa que las matrices siempre seran de 2x2*/
```

```
const int N = 2;

// definir la estructura del array
struct Struct_array {
    double arr[N][N];
};

//crear funcion para imprimir complejo
void print_complex(double real, double imaginaria, string desc){
    std::cout << desc << real;
    if (imaginaria >= 0) {
        std::cout << " + ";
    } else {
        std::cout << " - ";
    }
    std::cout << std::abs(imaginaria) << "i" << std::endl;
}

// crear funcion para imprimir matriz
void print_matrix(Struct_array array, double real, double imaginaria){
    if (imaginaria >= 0){
    cout << "\t--Matrix de " << real << "+" << imaginaria << "i--\n";
    }
    else{cout << "\t--Matrix de " << real << imaginaria << "i--\n";
    }
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            cout<<"\t"<<array.arr[i][j] <<"\t";
        }
        cout<<endl;
    }
    cout<<endl;
}

// crear funcion para sumar las dos matrices
Struct_array sum_matrix(Struct_array p, Struct_array q){
    Struct_array result_sum;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            result_sum.arr[i][j] = p.arr[i][j] + q.arr[i][j];
        }
    }
    return result_sum;
}

// crear funcion para multiplicar las dos matrices
Struct_array product_matrix(Struct_array n, Struct_array m){
    Struct_array result_product;
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            result_product.arr[i][j] = 0;
            for (int k = 0; k < N; k++){
                result_product.arr[i][j] += n.arr[i][k] * m.arr[k][j];

            }

```

```
        }

    }
    return result_product;
}

int main () {
    std::cin.exceptions(std::ios_base::failbit);
    // declaracion de variables
    double a, b, c, d; /*vease que se piden dos doubles por cada numero
    complejo por ser mas intuitivo que pedir el complejo con su formato correcto
    (entre parentesis y separando parte real e imaginaria con una coma)*/
    try{ // comprobar que el usuario introduce correctamente los numeros complejos
    std::cout << "Introduzca el primer numero complejo (valor real + valor imaginario):\t";
    std::cin >> a >> b;
    std::cout << "Introduzca el segundo numero complejo (valor real + valor imaginario):\t";
    std::cin >> c >> d;
    } // si no lo hace, lanzar error y finalizar
    catch(std::ios_base::failure &error){
        std::cout << "Error! El numero complejo no tiene el formato correcto\n";
        std::cout << "Ejemplo de entrada: 2.0 4.55 \n";
        std::cout << "Intentelo de nuevo.";
        throw;
    }
    std::complex<double> x(a,b), y(c,d); // ahora declarar los complejos
    // Hacer las operaciones y escribir el resultado en consola
    print_complex(x.real(), x.imag(), "El primer numero complejo introducido es: ");
    print_complex(y.real(), y.imag(), "El segundo numero complejo introducido es: " );
    std::complex<double> sum = x + y;
    print_complex(sum.real(), sum.imag(), "La suma de ambos numeros complejos es: " );
    std::complex<double> product = x * y;
    print_complex(product.real(), product.imag()
    , "El producto de ambos numeros complejos es: " );
    // Declarar matrices por cada uno de los numeros complejos
    Struct_array matrix_1 = {real(x), -imag(x), imag(x), real(x)};
    Struct_array matrix_2 = {real(y), -imag(y), imag(y), real(y)};
    // Realizar las operaciones con las matrices y escribit el resultado en consola
    Struct_array suma = sum_matrix(matrix_1, matrix_2);
    Struct_array producto = product_matrix(matrix_1, matrix_2);
    std::cout << "La suma y el producto de dos numeros complejos puede calcularse sumando";
    std::cout << "y multiplicando sus respectivas matrices\n";
    std::cout << "La matriz correspondiente al primer numero complejo es:\n";
    print_matrix(matrix_1, real(x), imag(x));
    std::cout << "La matriz correspondiente al segundo numero complejo es:\n";
    print_matrix(matrix_2, real(y), imag(y));
    std::cout << "La matriz resultado de esta suma es:\n";
    print_matrix(suma, suma.arr[0][0], suma.arr[1][0]);
    std::cout << "La matriz resultado del producto es:\n";
    print_matrix(producto, producto.arr[0][0], producto.arr[1][0]);
  return 0;
}
```

1.5.3. Código ejercicio 3

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <chrono>
#include <iomanip>
#include <map>
#include <vector>
#include <algorithm>
using namespace std;

//Definir las constantes globales
const string filename = "diarioActividad.txt";
const string prefijo = "M";

// Crear variables para obtener los diferentes datos del fichero
string id;
string inicio;
string final;
string tipo;

/*Quiero trabajar con un diccionario (mapa) de estructuras, para ello
primero debo de definir la estructura que usaré*/
struct maquina{
    int id; // el id sera un numero entero para poder usarlo luego para sortear por orden
    int auto_n;
    int manual_n;
    int error_n;
    int tiempo_auto;
};


struct comparador {
    bool operator()(const pair<string, maquina>& x, const pair<string, maquina>& y) const {
        return x.second.id < y.second.id;
    }
}; //gracias a esta función podremos ordenar por id

// Definir una función para calcular la duración:
int calcular_duracion(string tiempo_1, string tiempo_2){
    tm t_1{}, t_2{};

    istringstream stream_1(tiempo_1);
    istringstream stream_2(tiempo_2);

    stream_1>> get_time(&t_1, "%H:%M");
    stream_2 >> get_time(&t_2, "%H:%M");

    auto empieza = chrono::hours(t_1.tm_hour) + chrono::minutes(t_1.tm_min);
    auto acaba = chrono::hours(t_2.tm_hour) + chrono::minutes(t_2.tm_min);

    int duracion = (acaba - empieza).count();

    return duracion;
```

```
}

int main(){
    /*Leemos el archivo y utilizamos una condicion para comprobar que el fichero
    puede ser leido y en caso de no ser asi devolver el error*/
    ifstream input;

    input.open(filename);

    if(!input.is_open()){
        cout << "Error, no es posible leer el fichero " << filename << std::endl;
        return 1;
    }

    map<string, maquina> maquinas; //Crear el mapa de estructuras
    int count = 0;
    /* definir constante para comprobar que el codigo funciona
     sobre todas las lineas del archivo*/
    while(input>>id>>inicio>>final>>tipo){ //loop para leer linea por linea

        struct maquina m = maquinas[id]; //declarar la estructura en el mapa

        input.get();

        string numero;
        numero = id.substr(prefijo.length());
        m.id = stoi(numero); // obtener el id en forma de entero

        /*Condiciones para obtener la duracion en auto
        y el numero total en auto, manual y error*/
        if (tipo == "AUTO"){
            m.tiempo_auto += calcular_duracion(inicio, final);
            m.auto_n += 1;
        }
        else if (tipo == "MANUAL"){
            m.manual_n +=1;
        }
        else if (tipo == "ERROR") { m.error_n +=1;
        }

       maquinas[id] = m; //introducir la estructura en el diccionario (mapa)

       count++;

       if(!input){ // comprobar que no ha habido errores en la lectura
            break;
        }


    }

    cout << "Numero de lineas leidas en el fichero " << count << endl;
    // para sortear el diccionario creamos un vector
    vector<pair<string, maquina>> v(maquinas.begin(), maquinas.end());
    // utilizar el comparador de estructuras para ordenar por id
    sort(v.begin(), v.end(), comparador());
```

```
    size_t size = v.size();
    cout << "Numero de maquinas: " << size << endl;
    // Imprimir en consola toda la informacion obtenida
    for (const auto& key : v) {
        cout << "Nombre: " << key.first << ", id: " << key.second.id
                  << ", total en auto: " << key.second.auto_n  << ", total en manual: "
                  << key.second.manual_n << ", total en error " << key.second.error_n
                  << ", tiempo en auto: " << key.second.tiempo_auto
                  << " min" << endl;
    }

    input.close(); // cerrar el archivo
    return 0;
}
```

1.5.4. Código ejercicio 4

```
#include <iostream>
#include <random>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

// definir las constantes globales
const bool MODO_DEBUG = true;
const int N = 10;

// crear estructura para recoger los estados y el numero entre 0 y 1
struct Tabla_de_estados
{
    int estado_S_1;
    int random_numero;
    int estado_S_2;
};
// crear array de estructuras con los datos de los estados y sus transiciones
static const Tabla_de_estados t[]=
{
    { 0, 0, 2},
    { 0, 1, 1},
    { 1, 0, 1},
    { 1, 1, 4},
    { 2, 0, 5},
    { 2, 1, 3},
    { 3, 0, 6},
    { 3, 1, 7},
    { 4, 0, 5},
    { 4, 1, 7},
    { 5, 0, 5},
    { 5, 1, 1},
    { 6, 0, 6},
    { 6, 1, 7},
```

```
    { 7, 0, 7},
    { 7, 1, 7},
};

// crear estructura de los datos que queremos imprimir en consola
struct informe{
    int n_transicion;
    int replicas = 0;
};

// crear funcion para calcular el index al que ir y hacer el codigo más rápido y eficiente
int index(int a, int b){
    int result = a*2 + b;
    return result;
}


// crear función para comparar y sortear por numero de transiciones
struct comparador {
    bool operator()(const pair<string, informe>& x, const pair<string, informe>& y) const {
        return x.second.n_transicion< y.second.n_transicion;
    }
};

int main(){
    if (MODO_DEBUG){cout << "Evolucion del automata en " << N << " replicas" << endl;}
    /* Las condiciones
    ayudaran a imprimir en caso de de estar en modo debug TRUE*/
    // Definir las distintas variables:
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<double> dis(0.0, 1.0);
    int T[N];
    map<string, informe> informes;
    // loop para realizar N veces el experimento
    for(int i= 0; i< N; i++){
        int estado = 0;
        int transicion = 0;
        if (MODO_DEBUG){cout << "S" << estado;}


        while(estado != 7){ // loop para realizar el experimento
            double random = dis(gen);
            int number = round(random); // definir numero pseudorandom entre 0 y 1
            int n = index(estado, number); // calcular el index del array
            estado = t[n].estado_S_2; // update el estado
            transicion += 1; // recoger numero de transiciones
            if (MODO_DEBUG){cout << ", " << number << ", " << "S" << estado;}



        }
        T[i] = transicion; // recoger transiciones en array
        if (MODO_DEBUG){cout << endl;}

    }
```

```
92
93      if (MODO_DEBUG){cout << "\n" << "Contenido del vector T:" << endl;}
94      for (int t: T){
95          struct informe z = informes[to_string(t)];
96          z.n_transicion = t;
97          z.replicas += 1; // popular las estructuras del diccionario
98          informes[to_string(t)] = z; // popular el diccionario
99          if (MODO_DEBUG){cout << t << " ";}
100     }
101     if (MODO_DEBUG){cout << "\n\n";}
102     cout << "Informe sobre el numero de transiciones:" << endl;
103     vector<pair<string, informe>> v(informes.begin(), informes.end());
104     /*crear vector para sortear
105     por numero de transiciones*/
106     sort(v.begin(), v.end(), comparador());
107     for (const auto& key : v) { // imprimir en consola los datos recogidos.
108         cout << key.second.n_transicion << " transiciones en "
109              << key.second.replicas << " replicas" << endl;
110     }
111
112     return 0;
113 }
```