Examen de la Convocatoria Extraordinaria

Fundamentos de la Programación I, Grupos A,B,E y F

12 de Junio de 2024

Puntuación máxima: 10 puntos

Duración: 3 horas

Normas de realización del examen

- Pon tu NOMBRE, APELLIDOS, y Grupo (A, B, E o F) al principio de tu fichero `main.cpp`.
- Si el examen tiene errores de compilación, **la calificación será de `0` puntos**.
- Un error en la especificación de los parámetros de una función, bajará la calificación de la pregunta a la mitad.
- Está terminantemente prohibido el uso de la instrucción `break`. Si la función que implementas no cumple esta restricción, *la pregunta será evaluada con `0` puntos*.
- Si una función devuelve `void`, no puede contener ninguna instrucción `return`. Si no devuelve `void`, debe contener únicamente una instrucción `return`. Si la función que implementes no cumple esta restricción, **la pregunta será evaluada con `0` puntos**.
- Utilizar recorrido en lugar de búsqueda o viceversa bajará la calificación de la pregunta a la mitad.
- Se valorará la eficiencia y claridad del código implementado.

Enunciado

La empresa YourBoxDoc ofrece servicios de gestión de documentos. Concretamente ofrece un archivo de documentos para poder almacenar documentos en cajas. Un *documento* consta de un código (`string` sin blancos) y un peso (`int`). Cada *caja* admite un peso máximo de `10` kg y contiene una lista de documentos de tamaño máximo `4`, junto con el peso total de todos los documentos de la lista. El *archivo de documentos* no es más que una lista de exactamente `3` cajas.

El archivo de documentos se carga a partir del fichero de texto `"operaciones.txt"`, que se adjunta con el enunciado. Si abres el fichero, verás que la primera línea contiene el número de operaciones que hay que procesar, y el resto de líneas representan las operaciones. Concretamente existen dos tipos de operaciones:

- *Añadir un documento en una caja concreta del archivo de documentos*. La operación de añadir se representa en el fichero, por ejemplo, como:

  ```
  ADD 2 P2 6
  ```

  que significa que se desea añadir a la caja `2` del archivo de documentos, el documento de código `P2` y peso `6` kg. Para llevar a cabo el almacenamiento en la caja, se debe comprobar que el documento cabe en la caja, es decir, la caja contiene menos de `4` documentos, y que al almacenar el documento la caja no pese más de `10` kilos. Observa que en el fichero las cajas van numeradas de 1 a 3, pero en la implementación, al usar arrays, estarán numeradas de `0` a 2.

- *Eliminar un documento del archivo de documentos*. La operación de eliminar se representa en el fichero, por ejemplo, como:

  ```
  RM N2
  ```

  que significa que se desea eliminar el documento de código `N2` del archivo de documentos. Para eliminar el documento debes buscar en las cajas del archivo de documentos hasta que encuentres el documento con ese código. Una vez encontrado debes eliminarlo de la caja y actualizar el peso actual de la caja. En caso de que no exista ningún documento con ese código, simplemente se informará al usuario.

El programa comenzará procesando las operaciones del archivo `"operaciones.txt"`, una a una. Para cada operación del tipo ADD que no se pueda realizar, por ejemplo, debido a que no quede espacio en la caja que indica la operación, almacenaremos su documento asociado en una *lista de documentos pendientes*, que posteriormente procesaremos. Tened en cuenta que los documentos se añadirán a esta lista únicamente si hay hueco suficiente. En caso contrario, se ignorarán los documentos que no quepan.

Una vez procesadas las operaciones, se mostrará por consola cómo ha quedado el archivo de documentos. A continuación, se procesará la lista de documentos pendientes de almacenar, y se almacenarán aquellos que sea posible, en este caso, buscando la primera caja del archivo de documentos donde se pueda almacenar.

Detalles de implementación

[1 punto] Debes definir los siguientes tipos:

- `tDocumento`: estructura que encapsula la información de un documento, es decir su código (`string`) y su peso (`int`).
- `tListaDocs`: representa una lista de documentos (array parcialmente completo de longitud máxima `4`).
- `tCaja`: encapsula la información de una caja. Una caja está compuesta por la lista de documentos que almacena, y el peso de todos los documentos almacenados en la lista. Una caja soporta como máximo `10` kilos.
- `tArchivo`: es el archivo de documentos, representado a través de un array de tamaño fijo `3`, que almacena *exactamente* `3` cajas.

Implementa las siguientes funciones, de las que sólo te damos el nombre. Los argumentos y el tipo que devuelven, debes ponerlos tú, considerando las explicaciones que te damos:

- **[0.25 puntos]** `iniciar_archivo_docs`: dado un archivo de documentos, inicializa todas sus cajas con `0` documentos y peso `0`.
- **[0.25 punto]** `insertar_final`: dado un documento y una lista de documentos, inserta el documento al final de la lista de documentos. Si la inserción ha sido posible, la función devuelve `true`. En otro caso devuelve `false`.
- **[0.75 puntos]** `colocar_documento_caja`: dado un documento y una caja, intenta colocar el documento en la caja. Si la operación tiene éxito devuelve `true` y coloca el documento al final de la caja, actualizando el peso actual de la misma. Si no es posible realizar la operación, la función devuelve `false`.
- **[1,25 puntos]** `eliminar_documento_caja`: dado el código de un documento y una caja, elimina de la caja dicho documento. Si la eliminación tiene éxito devuelve `true`. En otro caso devuelve `false`.
- **[1,25 puntos]** `eliminar_documento_archivo`: dado el código de un documento y un archivo de documentos, elimina del mismo el documento con ese código. Si la eliminación tiene éxito, devuelve la posición de la caja de la que se ha eliminado el documento. En otro caso devuelve `-1`.
- **[1,25 puntos]** `colocar_documento_archivo`: dado un documento y el archivo de documentos, intenta colocar el documento en la primera caja del archivo de documentos donde sea posible. Si la operación tiene éxito, devuelve la posición de la caja donde ha almacenado el documento. En otro caso devuelve `-1`.
- **[1,25 puntos]** `procesar_operaciones`: dado un fichero (`ifstream&`), un archivo de documentos vacío, y una lista de documentos pendientes vacía, realiza el procesamiento de las operaciones. Al principio del enunciado ya se ha explicado en qué consisten las operaciones ADD y RM. Si alguno de los identificadores de las operaciones no es correcto, ignora la línea y sigue procesando operaciones. Aquellos documentos correspondientes a operaciones ADD que no se hayan podido realizar, se almacenarán en la lista de documentos pendientes. Esta función debe informar al usuario del procesamiento de cada una de las operaciones. Para ello, consulta el ejemplo de ejecución al final del examen.

- **[0,75 puntos]** `procesar_docs_pendientes`: dada una lista de documentos y un archivo de documentos, intenta almacenar cada uno de los documentos en el archivo de documentos. Cada documento se almacenará en la primera caja del archivo de documentos en la que sea posible. Aquellos documentos que no se puedan almacenar, directamente se ignoran y se pasa a procesar el siguiente documento. Esta función debe informar al usuario del procesamiento de cada uno de los documentos. Para ello consulta el ejemplo de ejecución al final del examen.
- **[0,25 puntos]** `mostrar_documento`: dado un documento, muestra su información por consola. Para ver cómo quedaría la salida, consulta el ejemplo de ejecución que aparece al final del examen.
- **[0,25 puntos]** `mostrar_lista_documentos`: dada una lista de documentos, la muestra por consola. Para ver como quedaría la salida, consulta el ejemplo de ejecución que aparece al final del examen.
- **[0,25 puntos]** `mostrar_caja`: dada una caja, muestra por consola su contenido. Para ello muestra el espacio libre disponible en la caja y a continuación la lista de documentos almacenados en la caja. Consulta el formato de salida en el ejemplo de ejecución al final del examen.
- **[0,25 puntos]** `mostrar_archivo_docs`: dado un archivo de documentos, lo muestra por consola. En el ejemplo de ejecución, al final del enunciado, puedes ver cómo queda la salida.

Función `main` [1 punto]

Primero abre el fichero `"operaciones.txt"`. Si la apertura ha sido correcta, inicializa un archivo de documentos, una lista de documentos pendientes, y realiza el procesamiento de las operaciones del fichero, una a una. Después, se muestra por consola el archivo de documentos, junto con los documentos pendientes de almacenar. Finalmente se procesan los documentos pendientes de almacenar y se vuelve a mostrar el archivo de documentos. Si la apertura del fichero `operaciones.txt` no ha sido posible, se escribe un mensaje de error por consola.

Junto con el enunciado te dejamos el fichero `"operaciones.txt"` y `main.cpp`, que contiene un esquema que debes completar respondiendo al examen.

Ejemplo de ejecución

A continuación mostramos cómo es la salida para el fichero `"operaciones.txt"`.

```
 ** PROCESAMIENTO DE OPERACIONES **

DOCUMENTO P2 (6 kg) ALMACENADO EN LA CAJA 2. EL PESO DE LA CAJA ES: 6
DOCUMENTO N1 (4 kg) ALMACENADO EN LA CAJA 1. EL PESO DE LA CAJA ES: 4
DOCUMENTO H1 (4 kg) ALMACENADO EN LA CAJA 3. EL PESO DE LA CAJA ES: 4
EL DOCUMENTO H2 (5 kg)  NO SE HA PODIDO ALMACENAR EN LA CAJA 2. DEJAMOS EL DOCUMENTO PENDIENTE
DOCUMENTO P7 (3 kg) ALMACENADO EN LA CAJA 3. EL PESO DE LA CAJA ES: 7
DOCUMENTO A2 (3 kg) ALMACENADO EN LA CAJA 1. EL PESO DE LA CAJA ES: 7
DOCUMENTO M7 (1 kg) ALMACENADO EN LA CAJA 3. EL PESO DE LA CAJA ES: 8
LA OPERACION AD NO ESTA SOPORTADA POR LA APLICACION
DOCUMENTO H4 (1 kg) ALMACENADO EN LA CAJA 3. EL PESO DE LA CAJA ES: 9
EL DOCUMENTO N2 (3 kg)  NO SE HA PODIDO ALMACENAR EN LA CAJA 3. DEJAMOS EL DOCUMENTO PENDIENTE
EL DOCUMENTO Q1 (2 kg)  NO SE HA PODIDO ALMACENAR EN LA CAJA 3. DEJAMOS EL DOCUMENTO PENDIENTE
```

```
EL DOCUMENTO N2 NO EXISTE EN EL ARCHIVO
DOCUMENTO H3 (2 kg) ALMACENADO EN LA CAJA 1. EL PESO DE LA CAJA ES: 9
EL DOCUMENTO J7 (2 kg)  NO SE HA PODIDO ALMACENAR EN LA CAJA 3. DEJAMOS EL DOCUMENTO PENDIENTE
DOCUMENTO A2 ELIMINADO DE LA CAJA 1EL PESO DE LA CAJA ES: 6
DOCUMENTO P2 ELIMINADO DE LA CAJA 2EL PESO DE LA CAJA ES: 0
DOCUMENTO J1 (4 kg) ALMACENADO EN LA CAJA 2. EL PESO DE LA CAJA ES: 4
EL DOCUMENTO KK NO EXISTE EN EL ARCHIVO

** ESTADO ACTUAL DEL ARCHIVO DE DOCUMENTOS **:

Caja numero 1: Espacio libre: 4 kg.
                N1 :    Peso:  4
                H3 :    Peso:  2

Caja numero 2: Espacio libre: 6 kg.
                J1 :    Peso:  4

Caja numero 3: Espacio libre: 1 kg.
                H1 :    Peso:  4
                P7 :    Peso:  3
                M7 :    Peso:  1
                H4 :    Peso:  1


** DOCUMENTOS PENDIENTES DE ARCHIVAR **

                H2 :    Peso:  5
                N2 :    Peso:  3
                Q1 :    Peso:  2
                J7 :    Peso:  2

** PROCESAMIENTO DE OPERACIONES PENDIENTES **

DOCUMENTO H2 (5 kg.) ALMACENADO EN LA CAJA 2. EL PESO DE LA CAJA ES: 9
DOCUMENTO N2 (3 kg.) ALMACENADO EN LA CAJA 1. EL PESO DE LA CAJA ES: 9
EL DOCUMENTO Q1 NO SE HA PODIDO ALMACENAR.
EL DOCUMENTO J7 NO SE HA PODIDO ALMACENAR.

** ESTADO ACTUAL DEL ARCHIVO DE DOCUMENTOS **:

Caja numero 1: Espacio libre: 1 kg.
                N1 :    Peso:  4
                H3 :    Peso:  2
                N2 :    Peso:  3

Caja numero 2: Espacio libre: 1 kg.
                J1 :    Peso:  4
                H2 :    Peso:  5

Caja numero 3: Espacio libre: 1 kg.
                H1 :    Peso:  4
                P7 :    Peso:  3
                M7 :    Peso:  1
                H4 :    Peso:  1
```

```cpp
//darklight_

#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
using namespace std;

const int MAX_CAJAS = 3;
const int MAX_DOCS = 4;
const double MAX_PESO_CAJA = 10;

typedef tDocumento tArrayDocs[MAX_DOCS];
typedef tCaja tArchivo[MAX_CAJAS];

struct tDocumento {
    string codigo;
    double peso;
};

struct tListaDocs {
    tArrayDocs documentos;
    int cont;
};

struct tCaja {
    int pesoTotal;
    tListaDocs docsCaja;
};


void iniciar_archivo_docs(tArchivo arch_docs);
void procesar_operaciones(ifstream& fichero, tArchivo arch_docs, tListaDocs&
   docs_pend);
void procesar_docs_pendientes(const tListaDocs& docs_pend, tArchivo
  arch_doc);
bool insertar_final(const tDocumento& doc, tListaDocs& docs);
int colocar_documento_archivo(const tDocumento& doc, tArchivo arch_doc);
bool colocar_documento_caja(const tDocumento& doc, tCaja& caja);
int eliminar_documento_archivo(const string& codigo, tArchivo arch_doc);
bool eliminar_documento_caja(const string& codigo, tCaja& caja);
void mostrar_documento(const tDocumento& doc);
void mostrar_caja(const tCaja& caja);
void mostrar_archivo_docs(const tArchivo archivo);
void mostrar_lista_documentos(const tListaDocs& listaDocs);


int main() {
    ifstream fichero;
    fichero.open("operaciones.txt");
    if (fichero.is_open()) {
        tArchivo arch_docs;
```

```
        tListaDocs lista_docs_pend;
        lista_docs_pend.cont = 0;
        /* todas las cajas se ponen vacías */
        iniciar_archivo_docs(arch_docs);

        cout << endl << "** PROCESAMIENTO DE OPERACIONES **" << endl <<
          endl;
        procesar_operaciones(fichero, arch_docs, lista_docs_pend);
        cout << endl << endl << "** ESTADO ACTUAL DEL ARCHIVO DE DOCUMENTOS
          **: " << endl;
        mostrar_archivo_docs(arch_docs);

        cout << endl << "** DOCUMENTOS PENDIENTES DE ARCHIVAR **" << endl <<
           endl;
        mostrar_lista_documentos(lista_docs_pend);

        cout << endl << "** PROCESAMIENTO DE OPERACIONES PENDIENTES **" <<
          endl << endl;
        procesar_docs_pendientes(lista_docs_pend, arch_docs);
        cout << endl << endl << "** ESTADO ACTUAL DEL ARCHIVO DE DOCUMENTOS
          **: " << endl;
        mostrar_archivo_docs(arch_docs);
        fichero.close();
    }
    else cout << "ERROR en la apertura del fichero de operaciones" << endl;
    return 0;
}

void iniciar_archivo_docs(tArchivo archivo) {
    for (int i = 0; i < MAX_CAJAS; i++) {
        archivo[i].pesoTotal = 0;
        archivo[i].docsCaja.cont = 0;
    }
}



void procesar_operaciones(ifstream& fichero, tArchivo arch_doc, tListaDocs&
  docs_pend) {
    int num_oper;
    fichero >> num_oper;
    for (int i = 0; i < num_oper; i++) {

        string tipo_oper;
        fichero >> tipo_oper;
        tDocumento doc;
        int donde;
        if (tipo_oper == "ADD") {
            fichero >> donde;
            fichero >> doc.codigo;
            fichero >> doc.peso;
            if (colocar_documento_caja(doc, arch_doc[donde - 1])) {
```

```
                cout << "DOCUMENTO " << doc.codigo << " (" << doc.peso << " kg) " << "ALMACENADO EN LA CAJA " << donde << ". ";
                cout << "EL PESO DE LA CAJA ES: " << arch_doc[donde - 1].pesoTotal << endl;
            }
            else {
                cout << "EL DOCUMENTO " << doc.codigo << " (" << doc.peso << " kg) " << " NO SE HA PODIDO ALMACENAR EN LA CAJA " << donde;
                if (insertar_final(doc, docs_pend)) {
                    cout << ". DEJAMOS EL DOCUMENTO PENDIENTE" << endl;
                }
                else cout << ". LA LISTA DE DOCUMENTOS PENDIENTES ESTA LLENA" << endl;

            }
        }
        else {
            if (tipo_oper == "RM") {
                fichero >> doc.codigo;
                int pos = eliminar_documento_archivo(doc.codigo, arch_doc);
                if (pos != -1) {
                    cout << "DOCUMENTO " << doc.codigo << " ELIMINADO DE LA CAJA " << pos + 1;
                    cout << "EL PESO DE LA CAJA ES: " << arch_doc[pos].pesoTotal << endl;
                }
                else {
                    cout << "EL DOCUMENTO " << doc.codigo << " NO EXISTE EN EL ARCHIVO " << endl;
                }
            }
            else {
                string basura;
                getline(fichero, basura);
                cout << "LA OPERACION " << tipo_oper << " NO ESTA SOPORTADA POR LA APLICACION" << endl;
            }

        }

    }

}


void procesar_docs_pendientes(const tListaDocs& docs_pend, tArchivo arch_doc) {
    for (int i = 0; i < docs_pend.cont; i++) {
        tDocumento doc = docs_pend.documentos[i];
        int pos = colocar_documento_archivo(doc, arch_doc);
        if (pos != -1) {
            cout << "DOCUMENTO " << doc.codigo << " (" << doc.peso << " kg.)
```

```
                " << "ALMACENADO EN LA CAJA " << pos + 1 << ". ";
            cout << "EL PESO DE LA CAJA ES: " << arch_doc[pos].pesoTotal <<
              endl;
        }
        else {
            cout << "EL DOCUMENTO " << doc.codigo << " NO SE HA PODIDO
              ALMACENAR. " << endl;
        }
    }
}



int colocar_documento_archivo(const tDocumento& doc, tArchivo arch_doc) {
    bool colocado = false;
    int pos = -1;
    int i = 0;
    while (i < MAX_CAJAS && !colocado) {
        colocado = colocar_documento_caja(doc, arch_doc[i]);
        if (!colocado) i++;
    }
    if (colocado) pos = i;
    return pos;
}

bool colocar_documento_caja(const tDocumento& doc, tCaja& caja) {
    bool r = true;
    if (doc.peso > MAX_PESO_CAJA - caja.pesoTotal) {
        r = false;
    }
    else { // colocamos el documento en la caja
        if (insertar_final(doc, caja.docsCaja)) {
            caja.pesoTotal = caja.pesoTotal + doc.peso;
        }
        else r = false;
    }
    return r;
}

bool insertar_final(const tDocumento& doc, tListaDocs& docs) {
    bool r = true;
    if (docs.cont < MAX_DOCS) {
        docs.documentos[docs.cont] = doc;
        docs.cont++;
    }
    else r = false;
    return r;
}

bool eliminar_documento_caja(const string& codigo, tCaja& caja) {
    int i = 0;
    bool encontrado = false;
    while (i < caja.docsCaja.cont && !encontrado) {
```

```
        if (caja.docsCaja.documentos[i].codigo == codigo) encontrado = true;
        else i++;
    }
    if (encontrado) {
        caja.pesoTotal = caja.pesoTotal - caja.docsCaja.documentos[i].peso;
        for (int j = i; j < caja.docsCaja.cont - 1; j++) {
            caja.docsCaja.documentos[j] = caja.docsCaja.documentos[j + 1];
        }
        caja.docsCaja.cont--;
    }
    return encontrado;
}

int eliminar_documento_archivo(const string& codigo, tArchivo arch_doc) {
    int i = 0;
    int pos = -1;
    bool encontrado = false;
    while (i < MAX_CAJAS && !encontrado) {
        encontrado = eliminar_documento_caja(codigo, arch_doc[i]);
        if (!encontrado) i++;
    }
    if (encontrado) pos = i;
    return pos;
}


void mostrar_documento(const tDocumento& doc) {
    cout << setw(18) << right << doc.codigo << " : "
        << setw(10) << left << "   Peso: " << doc.peso << endl;
}

void mostrar_caja(const tCaja& caja) {
    cout << "Espacio libre: " << MAX_PESO_CAJA - caja.pesoTotal << " kg. "
      << endl;
    for (int j = 0; j < caja.docsCaja.cont; j++)
        mostrar_documento(caja.docsCaja.documentos[j]);
}

void mostrar_archivo_docs(const tArchivo archivo) {
    for (int i = 0; i < MAX_CAJAS; i++) {
        cout << endl << "Caja numero " << i + 1 << ": ";
        mostrar_caja(archivo[i]);
    }
    cout << endl;
}

void mostrar_lista_documentos(const tListaDocs& listaDocs) {
    for (int i = 0; i < listaDocs.cont; i++)
        mostrar_documento(listaDocs.documentos[i]);
}
```