La biblioteca popular lo contrata para realizar un software que permita consultar los libros que se encuentran en la biblioteca. Los libros tienen un número de identificación, un nombre y un autor. Además, de un libro puede haber varios ejemplares impresos, los ejemplares tienen un isbn (que permite identificarlo), un año de impresión y una editorial.

Es necesario que el software permita cargar estos datos y guardarlos en uno o varios archivos binarios. Los cuales luego se van a poder consultar.

(30) Realice un modelado que permita guardar los datos de los libros y sus ejemplares

(25) Realice una funcionalidad que dado el nombre de un libro permita saber todos sus ejemplares e imprima en un archivo de texto, todos los datos de los ejemplares.

(30) Realice procedimientos que permitan saber (utilizando stl) :

El o los libros con mayor cantidad de ejemplares.
Los libros que fueron impresos por diferentes editoriales.
La editorial que tiene más cantidad de ejemplares impresos.

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <fstream>
#include <string>
#include <cstring>
using namespace std;

struct datosLibro{
    char nombre[200];
    char autor[200];
    int numeroIdentificacion;
};
struct datosEjemplar{
    char editorial[200];
    int anioImpresion;
    int isbn;
    int numeroIdentificacion;
};
```

```
class ejemplar{
    private:
    string editorial;
    int anioImpresion;
    int isbn;
    int numeroIdentificacion;
    public:
    ejemplar(string edi, int anio, int isbn){
        editorial = edi;
        this->isbn = isbn;
        anioImpresion = anio;
    }
    int getAnioImpresion() const{
        return anioImpresion;
    }
    int getisbn() const{
        return isbn;
    }
    void setNroIdentificacion(int nro){
        numeroIdentificacion = nro;
    }
    int getNroIdentificacion() const{
        return numeroIdentificacion;
    }
    string getEditorial() const{
        return editorial;
    }
    datosEjemplar getDatos(){
        datosEjemplar aux;
        strcpy(aux.editorial,editorial.c_str());
        aux.isbn = isbn;
        aux.anioImpresion = anioImpresion;
        aux.numeroIdentificacion =
numeroIdentificacion;
        return aux;
    }};
```

```
class Libro{
    vector<ejemplar>ejemplares;
    string nombre;
    string autor;
    int numeroIdentificacion;
    public:
    Libro(string nombre, string autor,
int nroIden){
        this->nombre = nombre;
        this->autor = autor;
        numeroIdentificacion = nroIden;
    }
    void addEjemplar(ejemplar&
ejem){

ejem.setNroIdentificacion(numeroIde
ntificacion);
        ejemplares.push_back(ejem);
    }
    datosLibro getDatos(){
        datosLibro aux;

strcpy(aux.nombre,nombre.c_str());
        strcpy(aux.autor,autor.c_str());
        aux.numeroIdentificacion =
numeroIdentificacion;
        return aux;
    }
    int getNroIdentificacion() const{
        return numeroIdentificacion;
    }
    bool TieneVariasEditoriales(){
        if (ejemplares.size() <= 1) return
false;
        string editorialAux =
ejemplares[0].getEditorial();
        for (int i = 1; i <
ejemplares.size(); ++i)
        {
```

```
//—--------------///
si no quiero usar get datos
void Biblioteca::GuardarLibros(){
    vector<LibroEstructura>LibroEstruct;
    LibroEstruct.resize(this->L.size());
transform(this->L.begin(),this->L.end(),Libro
Estruct.begin(),[](Libro& libro){
LibroEstructura le;
strncpy(le.autor,libro.get_autor().c_str(),sizeof
(le.autor));
strncpy(le.nombre,libro.get_nombre().c_str(),s
izeof(le.nombre));
        le.numero_ID=libro.get_numero_ID();
        return le;
    });
ofstream
archivo("C:/Users/nicov/OneDrive/Document
os/untitled5/VectorLibro.bat", std::ios::binary);
            archivo.write(reinterpret_cast<const
char*>(LibroEstruct.data()),
LibroEstruct.size() * sizeof(LibroEstructura));
            archivo.close();}
```

```
            if (editorialAux !=
ejemplares[i].getEditorial()) return
true;
        }
        return false;
    }
    vector<ejemplar> getEjemplares() {
        return ejemplares;
    }
    size_t getCantidadEjemplares()
const{
        return ejemplares.size();
    }
    string getNombre() const{
        return nombre;
    }
    string getAutor() const{
        return autor;
    }};
```

```
class Biblioteca{
    private:
    vector<Libro>Libros;
    public:
    void addLibro(Libro& libro2Aniadir){
        Libros.push_back(libro2Aniadir);
    }
    void guardarLibroYEjemplares(const char* direccionLibros, const char* direccionEjemplares){
        ofstream archivoLibros(direccionLibros, ios::binary);
        ofstream archivoEjemplares(direccionEjemplares, ios::binary);
        if (archivoEjemplares.fail() || archivoLibros.fail()) return;
        for (auto& libro : Libros){
            archivoLibros.write(reinterpret_cast<char*>(&libro.getDatos()),sizeof(datosLibro));
            auto ejemplares = libro.getEjemplares();
            for (ejemplar auxEjemplar : ejemplares){
                archivoEjemplares.write(reinterpret_cast<char*>(&auxEjemplar.getDatos()),sizeof(datosEjemplar));
            }
        }
        archivoLibros.close();
        archivoEjemplares.close();
    }
    /*(25) Realice una funcionalidad que dado el nombre de un libro permita saber todos sus ejemplares e imprima en un
archivo de texto, todos los datos de los ejemplares.*/
    void ConsultarLibro(const char* direccion, string nombreLibro){
        ofstream archivo(direccion);
        if(archivo.fail()) return;
        auto lib = find_if(Libros.begin(), Libros.end(), [nombreLibro](Libro &libro){
            return libro.getNombre() == nombreLibro;
        });
        if(lib == Libros.end()){return;}
        auto ejemplares = lib->getEjemplares();
        archivo << "Ejemplares de " << lib->getNombre() << ": \n";
        for(auto& ejemplar : ejemplares ){
      archivo << ejemplar.getEditorial() << "   " << ejemplar.getisbn() << "   " << ejemplar.getAnioImpresion() << "\n"; }}
    vector<Libro> mayorCantidadDeEjemplares(){
        vector<Libro>MasEjemplares;
```

```
    if (!Libros.size()) return MasEjemplares;
    sort(Libros.begin(),Libros.end(),[](Libro &aux1, Libro& aux2){
      return aux1.getCantidadEjemplares() > aux2.getCantidadEjemplares();
    });
    int mayor = Libros[0].getCantidadEjemplares();
    copy_if(Libros.begin(),Libros.end(),back_inserter(MasEjemplares),[mayor](Libro& aux1){
      return aux1.getCantidadEjemplares() == mayor;
    });
    return MasEjemplares;
  }
  vector<Libro> LibrosConVariasEditoriales(){
    vector<Libro>VariasEditoriales;
    copy_if(Libros.begin(),Libros.end(), back_inserter(VariasEditoriales),[](Libro aux1){
      return aux1.TieneVariasEditoriales();
    });
    return VariasEditoriales;
  }
  string EditorialConMasEjemplares(){
    map<string,int>Editoriales;
    for (auto& libro : Libros){
      vector<ejemplar>ejemplares = libro.getEjemplares();
      for_each(ejemplares.begin(),ejemplares.end(),[&Editoriales](ejemplar& aux1){
        Editoriales[aux1.getEditorial()]++;
      });
    }
    vector<pair<string,int>>aux(Editoriales.begin(), Editoriales.end());
// es lo mismo que
vector<pair<string, int>> aux(Editoriales.size()); copy(Editoriales.begin(), Editoriales.end(), aux.begin()); //
    sort(aux.begin(),aux.end(),[](pair<string,int>aux1, pair<string,int>aux2){
      return aux1.second > aux2.second;
    });
    return aux[0].first; }};
```

Restaurante - software que controle los costos de sus platos

Un restaurante nos contrata para hacer un software que controle los costos de sus platos. Los platos están compuestos por ingredientes los cuales pueden ser platos o ingredientes básicos. Por ejemplo los panqueques con dulce de leche, está compuesto por panqueques (que es un plato) y dulce de leche que es un ingrediente básico.

El software debe mantener el costo de cada uno de los platos, y el costo está dado por la suma de los costos de los ingredientes (básicos o platos).

(25) Realice el diseño de la solución e implemente las clases. Conteste: Se puede utilizar polimorfismo, cual es la ventaja o desventaja?

(25) Diseñe una estructura de archivos y realice los métodos necesarios para guardar los platos con sus ingredientes.

(20) Dado un vector (de stl) con los platos realice las siguientes metodos, en la clase que crea conveniente:
- Ordenar los platos por menor costo.
- Guardar el top 5 de platos más baratos en un archivo de texto. Con el siguiente formato: nombre del plato : costo
- Guardar en un archivo de texto los platos que no contengan harina.

```
#include <string>
#include <cstring>
using namespace std;

struct ingre{
    int cod;
    char nombre_i[200];
    float precio_i;
};

class ingredientes
{
private:
    string nombre_i;
    float costo_i;
    int cod;
public:
    ingredientes();
    ingredientes(string,float);
    float getcostoi();
    string getnombre();
    ingre getingre();
};
//—----------//
ingredientes::ingredientes() {}

ingredientes::ingredientes(string n , float
c)
{this->costo_i=c;
 this->nombre_i=n;}

float ingredientes::getcostoi()
{return costo_i;}

string ingredientes::getnombre()
{return nombre_i;}

ingre ingredientes::getingre()
{ingre aux;
strcpy(aux.nombre_i,nombre_i.c_str());
    aux.precio_i = costo_i;
    aux.cod = cod;
    return aux;
}
```

```
#include "ingredientes.h"
#include <string>
#include <vector>
#include <cstring>
using namespace std;

struct plat{
    int cod;
    char nombre_p[200];
    float precio_p;
};

class Plato
{
private:
   string nombre_p;
   float costo_p;
   int cod;
   vector<ingredientes> ing;
public:
   Plato();
   Plato(string,float,
vector<ingredientes>);
   plat getDatos();
   string getnombre();
   float getcostop();
   float getcostoi();
   float costo_total();
   string getning();
   vector<ingredientes> geting();
};
//—----------//
Plato::Plato() {}

Plato::Plato(string n, float c,
vector<ingredientes> i)
{
    this->nombre_p=n;
    this->costo_p=c;
    this->ing.resize(i.size());

copy(i.begin(),i.end(),this->ing.begin());
}

plat Plato::getDatos()
{plat aux;

strcpy(aux.nombre_p,nombre_p.c_str());
    aux.precio_p=costo_p;
    aux.cod=cod;
    return aux;
}

string Plato::getnombre()
{return nombre_p;}

float Plato::getcostop()
```

```
#include <vector>
#include "plato.h"
#include <algorithm>
#include <fstream>
using namespace std;

class Restaurante
{
private:
   vector<Plato>platos;
public:
   Restaurante();
   void addplato(Plato& p);
   void guardarPlatoeIng(const char*
direccionp, const char* direccioni);
   void ordenar();
   void top5(const char *direcciont);
   void platosintac(const char
*direccionsintac);
};
//—----------//
Restaurante::Restaurante() {}

void Restaurante::addplato(Plato &p)
{
   platos.push_back(p);
}

void
Restaurante::guardarPlatoeIng(const
char *direccionp, const char *direccioni)
{
   ofstream archivop(direccionp,
ios::binary);
   ofstream archivoi(direccioni,
ios::binary);
   if (archivop.fail() || archivoi.fail())
return;
   for (auto & p : platos){

archivop.write(reinterpret_cast<char*>(
&p.getDatos()),sizeof(plat));
      auto ings = p.geting();
      for (ingredientes auxi : ings){

archivoi.write(reinterpret_cast<char*>(
&auxi.getingre()),sizeof(ingre));
      }
   }
   archivop.close();
   archivoi.close();
}

void Restaurante::ordenar()
{
   sort( platos.begin(),platos.end(),[]
   (Plato& a,Plato& b) {return
```

	{return costo_p;}
	
	float Plato::getcostoi()
	{
	float aux=0;
	for(int i=0;i<this->ing.size();i++){
	aux=aux+ing[i].getcostoi();
	}
	return aux;
	}
	
	float Plato::costo_total()
	{
	return this->costo_p+getcostoi();
	}
	
	string Plato::getning()
	{
	for(int i=0;i<ing.size();i++){
	return ing[i].getnombre();
	}
	}
	
	vector<ingredientes> Plato::geting()
	{return ing;}
		a.getcostop() < b.getcostop();}
		);
		}
		
		void Restaurante::top5(const char *direcciont)
		{
		ordenar();
		ofstream architop5(direcciont);
		if(architop5.fail()){return;}
		for(int i=0;i<5;i++){
		architop5<<"nombre plato: "<<platos[i].getnombre()<<" precio $";
		
		architop5<<platos[i].costo_total()<<endl;
		}
		architop5.close();
		}
		
		void Restaurante::platosintac(const char *direccionsintac)
		{
		ofstream celiaco(direccionsintac);
		vector<ingredientes> aux;
		
		copy(platos.begin(),platos.end(),aux.begin());
		
		for(int i=0;i<platos.size();i++){
		if(platos[i].getning()!="harina"){
		
		celiaco<<platos[i].getnombre()<<endl;
		} }
		celiaco.close();}

Empresa de dictado de cursos online

Una Empresa de dictado de cursos online lo contrata para realizar un software que mantenga sus cursos. Existen 3 tipos de cursos :

- Simples: Tienen un código (alfanumérico) , una cantidad de horas asignadas, un nombre, tema relacionado.
- Simple con correlatividades: Igual que el anterior pero es necesario cursar cursos anteriores. Por ejemplo para cursar cálculo 2 es necesario tener aprobado calculo 1. Las horas asignadas del curso son las horas cargadas para este curso más todas las horas de los cursos correlativos.
- Compuesto: Tienen un código (alfanumérico) , un nombre, tema relacionado y un conjunto de cursos simples que lo conforman. Por ejemplo el curso programación web, está compuesto por el curso de javascript, css, html y php. El cálculo de las horas asignadas son la suma de las horas asignadas de cada curso simple que lo conforma.

1. Realice un diseño del software y una función que permita cargar los cursos y calcule las horas asignadas de los mismos.
2. Realice una función que permita guardar los cursos en archivos. Diseñe el o los archivos para guardar la información de forma óptima.
3. Realice procedimiento con funciones STL que permita saber:

    1. Los 5 cursos que tienen menos horas asignadas.
    2. El Tema que está en la mayor cantidad de cursos.

```
#include <cstring>
#include <string>

using namespace std;
class Curso
{
public:
   string codigo;
   int hs;
   string nombre;
   string tema;
   Curso();
   Curso(string codigo, int hs, string nombre, string tema);
   virtual int calcualar_horas()=0;
   virtual string get_codigo();
   virtual int get_hs();
   virtual string get_nombre();
   virtual string get_tema();
};
//—-------------------------//
#include "curso.h"

Curso::Curso(){}

Curso::Curso(string codigo, int hs, string nombre, string tema) {}

string Curso::get_codigo(){ return codigo;}

int Curso::get_hs(){ return hs;}

string Curso::get_nombre(){ return nombre;}

string Curso::get_tema(){ return tema;}
```

```
#include "curso.h"
using namespace std;

class Simple:public Curso
{
private:
  // bool aprobado;
public:
   Simple();
    int calcualar_horas();
   void add_curso(string codigo, int
hs, string nombre, string tema);
};
//—---------//
#include "simple.h"

Simple::Simple() {

}

 int Simple::calcualar_horas()
{
   return this->hs;
```

```
#include "curso.h"
#include "simple.h"
#include <vector>

using namespace std;

class Compuesto:public Curso
{
private:
vector<Simple>Cursos_del_Curso;
public:
   Compuesto();
   void add_curso(string codigo, string
nombre, string
tema,vector<Simple>Cursos);
   void agregar_materias(Simple
&CursoC);
int calcualar_horas();
vector<Simple>get_cursoContenidos();
};
//—---------------------//
#include "compuesto.h"
```

```
#include "simple.h"
#include "curso.h"
#include <vector>

class SimpleCorrelativo:public Curso
{
private:
   vector<Simple>CorrelCurso;
public:
    SimpleCorrelativo();
    void add_curso(string codigo, int hs,
string nombre, string tema);
    void agregar_Correlativa(Simple
&CursoC);
    int calcualar_horas();
vector<Simple>get_cursoCorrelativos();};
//—-------------//
#include "simplecorrelavo.h"

SimpleCorrelativo::SimpleCorrelativo() {}

int SimpleCorrelativo::calcualar_horas()
{
```

```
}

void Simple::add_curso(string codigo, int hs, string nombre, string tema)
{
   this->codigo=codigo;
   this->hs=hs;
   this->nombre=nombre;
   this->tema=tema;
}
```

```
Compuesto::Compuesto() {}

void Compuesto::add_curso(string codigo, string nombre, string tema, vector<Simple>Cursos)
{
   this->codigo=codigo;
   this->nombre=nombre;
   this->tema=tema;
}

void Compuesto::agregar_materias(Simple &CursoC)
{
Cursos_del_Curso.push_back(CursoC);
}

int Compuesto::calcualar_horas()
{
   int aux_hs=0;
   for(int i=0;i<this->Cursos_del_Curso.size();i++)

aux_hs+=Cursos_del_Curso[i].calcualar_horas();
   return aux_hs;
}
vector<Simple> Compuesto::get_cursoContenidos()
{
   return Cursos_del_Curso;
}
```

```
   int aux_hs=0;
   for(int i=0;i<this->CorrelCurso.size();i++)

aux_hs+=CorrelCurso[i].calcualar_horas();

   return aux_hs+this->hs;
}

vector<Simple> SimpleCorrelativo::get_cursoCorrelativos()
{
   return CorrelCurso;
}

void SimpleCorrelativo::add_curso(string codigo, int hs, string nombre, string tema)
{
   this->codigo=codigo;
   this->hs=hs;
   this->nombre=nombre;
   this->tema=tema;
}
void SimpleCorrelativo::agregar_Correlativa(Simple &CursoC)
{
   CorrelCurso.push_back(CursoC);}
```

```
#include "compuesto.h"
#include "simple.h"
#include "simplecorrelavo.h"
#include <vector>
#include <algorithm>
#include <fstream>
class gestora
{
private:
   vector<Curso*>Cursos;
   vector<Simple>CursosSimples;
   vector<SimpleCorrelativo>CursoConCorrelativas;
   vector<Compuesto>CursoCompuesto;
public:
   gestora();
   void addcurso(Curso *&C);
   void guardarCursos(const char*direccion);
   void Top5HS(const char *direcciont);
   void ordenar();
   void temaMasFrecuente(vector<Curso*>& cursos);
   void guardarSimple(const char*direccion);
   void guardarCorrelativo(const char*direccion);
   void guardarCompuesto(const char*direccion);
};
```

```
void gestora::Top5HS(const char *direcciont)
   { ordenar();
      ofstream architop5(direcciont);
      if(architop5.fail()){return;}
      for(int i=0;i<5;i++){
         architop5<<"nombre : " <<Cursos[i]->get_nombre();
      }
      architop5.close();
   }
   void gestora::ordenar(){
      sort(Cursos.begin(),Cursos.end(),[]
           (Curso*& a,Curso*& b) {return a->get_hs() < b->get_hs();}
               );}

void temaMasFrecuente(vector<Curso*>& cursos) {
   std::unordered_map<std::string, int> temaContador;

   for (const auto& curso : cursos) {
      temaContador[curso->tema]++;
   }

   std::string temaMasFrecuente;
   int maxContador = 0;
```

```
//—------------------------//
#include "gestora.h"

gestora::gestora() {}

void gestora::addcurso(Curso *&C){ Cursos.push_back(C);}

void gestora::guardarCursos(const char *direccion)
{
   ofstream archivo1(direccion);
   if(archivo1.fail()){
      return;}
   ofstream archivo2(direccion);
   if(archivo2.fail()){
      return;}
   ofstream archivo3(direccion);
   if(archivo3.fail()){
      return;}
   for (const auto& curso : Cursos) {
  archivo1 << curso->codigo << "\n" << curso->nombre <<
"\n" << curso->tema << "\n";
      // Usar dynamic_cast para diferenciar los tipos
      if (Simple* simple = dynamic_cast<Simple*>(curso)) {
         // Guardar horas para curso simple
         archivo1 << "Tipo: Simple\n";
         archivo1 << simple->calcualar_horas() << "\n";
      }
      else if (SimpleCorrelativo *correlativo =
dynamic_cast<SimpleCorrelativo*>(curso)) {
         // Guardar horas para curso con correlatividades
         archivo1 << "Tipo: Con Correlatividades\n";
         archivo1 << correlativo->calcualar_horas() << "\n";
            vector<Simple>Aux
=correlativo->get_cursoCorrelativos();
            for(int i=0;i<Aux.size();i++){
               archivo2<<Aux[i].get_codigo();
               archivo2<<Aux[i].get_hs();
               archivo2<<Aux[i].get_tema();
               archivo2<<Aux[i].get_nombre();
            }archivo2<<endl;
      }else if (Compuesto *compuesto =
dynamic_cast<Compuesto*>(curso)) {
         // Guardar horas para curso compuesto
         archivo1 << "Tipo: Compuesto\n";
         archivo1 << compuesto->calcualar_horas() << "\n";
  vector<Simple>Aux =compuesto->get_cursoContenidos();
        for(int i=0;i<Aux.size();i++){
            archivo3<<Aux[i].get_codigo();
            archivo3<<Aux[i].get_hs();
            archivo3<<Aux[i].get_tema();
            archivo3<<Aux[i].get_nombre();
      }archivo3<<endl;}
   archivo1.close();
   archivo2.close();
   archivo3.close(); }}
```

```
for (const auto& par : temaContador) {
   if (par.second > maxContador) {
      maxContador = par.second;
      temaMasFrecuente = par.first;
   } }}
```

Generar tweets para promocionar empresa

Una empresa nos contrata para generar tweets para promocionar su empresa en twitter. El sistema debe tomar un archivo de texto que tiene un tweet por linea y generar un archivo binario con los tweets formateados y además el procedimiento debe imprimir algunas estadísticas.
La empresa tienen un archivo de texto que contiene un conjunto de líneas donde cada línea será un tweet. El archivo de salida debe ser binario y tener el siguiente formato:

tweet
- número : entero
- tweet : char[140]
- tag : char[20]

El archivo de texto tiene líneas que pueden tener más de 140 caracteres, en este caso se debe cortar la línea. El tag es la palabra con más ocurrencias en la línea que forma el tweet.

1. (10) Diseñe la solución (realice un diagrama de clases)

2. (30) Implemente la solución.

3. (20) Al concluir, el procedimiento debe imprimir:
   a. La palabra que más se repite en todo el archivo de texto.
   b. Los tweets repetidos.
   c. El tweets con menos cantidad de caracteres.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <unordered_map>
#include <algorithm>
#include <cstring>

class Tweet {
public:
    int numero;
    char contenido[140];
    char tag[20];
    // Constructor
    Tweet(int n, string& c) {
    numero = n;
   strncpy(contenido, c.c_str(), sizeof(contenido) - 1);
  //Asegurarse de que esté terminado en nulo
 contenido[sizeof(contenido) - 1] = '\0';
        calcularTag();
    }
    // Método para calcular el tag (medio al pepe)
    void calcularTag() {
        unordered_map<string, int> conteo;
        string palabra;
        istringstream iss(contenido); //lee con espacios pa
limpiar emojis
        while (iss >> palabra) {
```

```
class TweetManager {
private:
    vector<Tweet> tweets;
public:
    // Cargar tweets desde un archivo de texto
    void cargarTweets(const string& filename) {
        ifstream infile(filename);
        string linea;
        int numero = 0;
        while (getline(infile, linea)) {
            if (linea.size() > 140) {
                linea = linea.substr(0, 140); // Cortar a 140
caracteres
            }
            tweets.emplace_back(numero++, linea);}}
    // Guardar tweets en formato binario
    void guardarTweetsBinario(const string& filename) {
        ofstream outfile(filename, ios::binary);
        for (const auto& tweet : tweets) {
outfile.write(reinterpret_cast<const char*>(&tweet.numero),
sizeof(tweet.numero));
outfile.write(tweet.contenido, sizeof(tweet.contenido));
    outfile.write(tweet.tag, sizeof(tweet.tag));
        }}
    // Imprimir estadísticas
    void imprimirEstadisticas() {
        unordered_map<string, int> conteoPalabras;
        unordered_map<string, int> conteoTweets;
        string tweetMenor;
```

```
        // Limpiar palabra de caracteres no alfanuméricos
        palabra.erase(remove_if(palabra.begin(),
palabra.end(), [](char c) {
            return !isalnum(c);
        }), palabra.end());

        if (!palabra.empty()) {
            conteo[palabra]++;}
    }
    // Encontrar la palabra más repetida
auto max_it = max_element(conteo.begin(), conteo.end(),
[](const auto& a, const auto& b) {
            return a.second < b.second; });
    if (max_it != conteo.end()) {
  strncpy(tag, max_it->first.c_str(), sizeof(tag) - 1);
        tag[sizeof(tag) - 1] = '\0';
// Asegurarse de que esté terminado en nulo
    } else {
        tag[0] = '\0'; // Sin tag
    }}};
```

```
        int minLongitud = INT_MAX;
        for (const auto& tweet : tweets) {
 // Contar la palabra más repetida en todos los tweets
            istringstream iss(tweet.contenido);
            string palabra;
            while (iss >> palabra) {
          conteoPalabras[palabra]++;
            }

   // Contar tweets repetidos
      conteoTweets[tweet.contenido]++;

     // Encontrar el tweet con menor longitud
            if (strlen(tweet.contenido) < minLongitud) {
                minLongitud = strlen(tweet.contenido);
                tweetMenor = tweet.contenido;
            }}
        // a. Palabra que más se repite
  auto max_it = max_element(conteoPalabras.begin(),
  conteoPalabras.end(),
  [](const auto& a, const auto& b) {
                return a.second < b.second;
            });
   cout << "Palabra más repetida: " << max_it->first << " ("
  << max_it->second << " veces)\n";

        // b. Tweets repetidos
  cout << "Tweets repetidos:\n";
        for (const auto& [tweet, conteo] : conteoTweets) {
            if (conteo > 1) {
     cout << tweet << " (" << conteo << " veces)\n";
            }}
        // c. Tweet con menos caracteres
cout << "Tweet con menos caracteres: " << tweetMenor <<
"\n";
    }};
```

Una empresa lo contrata para realizar un sistema de monitorización de sitios webs, para invertir en publicidad, la empresa tiene la información de las visitas a sitios en un archivo binario con la siguiente estructura :

url : char[20]
tiempo de permanencia : int (en segundos)
origen : char[1] // B buscador, D directo, O otro

El sistema lleva un puntaje de las visitas a los sitios web según su origen:

- Buscador: 10 * tiempo de permanencia en segundos
- Directo: 15 * tiempo de permanencia en segundos
- Otro: 1 * tiempo de permanencia en segundos

Tenga en cuenta que un sitio web puede tener muchas visitas.

1. (22) Realice el diseño y programe las clases que permitan saber el puntaje de cada sitio web.

2. (20) Escriba un archivo de texto que permita saber el puntaje de cada sitio web con el siguiente formato: "url ……. puntaje".

3. (22) Realice funciones usando stl :

- Que obtenga los 5 sitios webs con mayor puntaje
- Que obtenga los sitios web que se accedieron de forma directa.

```
#include <fstream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <map>

using namespace std;

struct Datos{
   char url[20];
   int permanencia;
   char origen;
};

class Visita{
protected:
   int permanencia; char origen;
public:
   Visita(Datos data):
permanencia(data.permanencia),
origen(data.origen) {}
   virtual int getPuntuacion() = 0;
   char getOrigen() {return origen;}
};

class Buscador: public Visita{
public:
   Buscador(Datos data):  Visita(data) {}
   int getPuntuacion() { return 10 *
this->permanencia; }
```

```
int Web::getPuntaje(){
   auto puntaje = reduce(visitas.begin(),
visitas.end(), 0, [](int sumatoria, Visita*
&v)
   { return sumatoria +
v->getPuntuacion();});
   return puntaje;
}

bool Web::lookIfAccessed(char tipo){
   bool accesed;
   accesed = any_of(visitas.begin(),
visitas.end(), [tipo](Visita* v){
      return v->getOrigen() == tipo;
   });
   return accesed;
}

// Gestor:
// Nota: Lo ideal seria guardar las webs
en un mapa<String,Web>, queda mas
comodo asi.
class GestorWeb{
public:
   GestorWeb(const char *
URLarchivo);
   Web getWeb(string url);
   // Consigna 2:
   void escribirPuntajes(const char*
URLarchivo);
```

```
Web GestorWeb::getWeb(string url){
   // Si no encuentra la web, se inventa
una
   for (auto &web: webs)
      if (web.getURL() == url)
         return web;
   return Web(url);
}

// Consigna 2
void GestorWeb::escribirPuntajes(const
char* URLar){
   ofstream archivo(URLar);
   if (archivo.fail())
      return;

   for (auto &web: webs){
      archivo << "url: " <<
web.getURL() << " puntaje: " <<
web.getPuntaje() << endl;
   }
   archivo.close();
}

// Consigna 3:
vector<Web> GestorWeb::getTop5(){
   sort(webs.begin(), webs.end(), [](Web
&a, Web &b){ return a.getPuntaje() >
b.getPuntaje(); });
   vector<Web> top5;
```

```
};

class Directo: public Visita{
public:
    Directo(Datos data):  Visita(data) {}
    int getPuntuacion() { return 15 *
this->permanencia; }
};

class Otro: public Visita{
public:
    Otro(Datos data):  Visita(data) {}
    int getPuntuacion() { return 1 *
this->permanencia; }
};

class Web {
public:
    Web(string URL): url(URL) {}
    void addVisita(Visita* v) {
visitas.push_back(v); }
    string getURL() { return url; }
    vector<Visita*> getVisitas() { return
visitas; }
    // Para consigna 2:
    int getPuntaje();
    // Para consigna 3:
    bool lookIfAccessed(char Tipo);
private:
    string url;
    vector<Visita*> visitas;
};
```

```
    // Consigna 3:
    vector<Web> getTop5();
    vector<Web>
getSitiosWebIndexadosDirecto(); // esta
podria retornar vector<string> nomas
private:
    vector<Web> webs;
};

GestorWeb::GestorWeb(const char*
URLar){
    ifstream arch(URLar, ios::binary);
    Datos temp;
    map<string,vector<Datos>>
mapalectura;
    while (arch.read((char*) &temp,
sizeof(Datos))){
        string url = temp.url;
        mapalectura[url].push_back(temp);
    }
    arch.close();
    for (auto &aux: mapalectura){
        Web pagina(aux.first);
        for (auto &visita: aux.second){
            switch (visita.origen){
                case 'B': pagina.addVisita(new
Buscador(visita));
                break;
                case 'D': pagina.addVisita(new
Directo(visita));
                break;
                case 'O': pagina.addVisita(new
Otro(visita));
                break;
            }
        }
        this->webs.push_back(pagina);
    }
}
```

```
    copy(webs.begin(),
next(webs.begin(), 5),
back_inserter(top5));
    return top5;
}

vector<Web>
GestorWeb::getSitiosWebIndexadosDire
cto(){
    vector<Web> indexados;
    copy_if(webs.begin(), webs.end(),
back_inserter(indexados),
    [](Web &w){ return
w.lookIfAccessed('D'); });
    return indexados;
}
```

Una empresa lo contrata para realizar un sistema que mantenga el valor de sus bienes y permita calcular la amortización de los mismos. Dicha empresa cuenta con un archivo binario donde se encuentran todos los bienes y su valor.

codigo : int
tipo : char //T terreno, C común, M mueble
valor : double

Los bienes pueden ser:
T: Terrenos, los cuales no amortizan, es decir el proceso de amortización no debería restar valor al bien.
C: Comunes, los cuales pierden un 5% su valor por mes
M: Muebles, los cuales pierden un 2% si valen menos de 10000 pesos y un 2.2%, si valen más.

Una vez al mes se corre un proceso de amortización que actualiza el valor de los bienes.

1. (22) Realice el diseño y programe las clases que permitan calcular el proceso de amortización.

2. (20) Escriba un archivo de texto que permita saber el valor de los bienes luego de corrido el proceso de amortización con este formato : "codigo ……. valor".

3. (22) Realice funciones usando stl :
   - Que obtenga el o los bienes con mayor valor
   - Que obtenga el valor total de todos los bienes discriminados por tipo.

4. (20) Realice una clase template que represente una lista circular **sin utilizar STL** que permita :

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <fstream>

using namespace std;

// Primer consigna:
struct DatosBienes{
    int codigo;
    char tipo;
    double valor;
};

// Superclase abstracta
class Bienes {
public:
    Bienes(int cod, char tip, double val): codigo(cod), tipo(tip), valor(val) {}
    Bienes(DatosBienes datos): codigo(datos.codigo), tipo(datos.tipo), valor(datos.valor) {}
    int getCodigo() {return codigo;}
```

```
void Comunes::amortizar() {
    this->valor *= 0.95;    // Le restamos el 5%
}

void Muebles::amortizar(){
    if (this->valor < 10000)
        this->valor *= 0.98; // Le restamos el 2%
    else this->valor *= 0.978; // Le restamos el 2.2%
}

class GestorBienes{
public:
    GestorBienes(const char* RutaArchivo);
    Bienes* getBien(int pos);
    void amortizarBienes();
    void crearArchivo(const char* RutaArchivo);
    vector<Bienes*> getBienesMayorValor();
        map<char, double> getTotalPorTipo();
private:
```

```
// Consigna 4:
template <class T>
class ListaCircular{
private:
struct Nodo{
    T dato;
    Nodo * link;
    Nodo * back;
};
Nodo * first;
Nodo * last;
T getRecursive (Nodo * nodo, int &Contador);

public:
    ListaCircular(): first(nullptr), last(nullptr) {}
    void insert(T dato);
    T get(int pos);
};

template <class T>
void ListaCircular<T>::insert(T dato){
    if (first == nullptr){
```

```
    char getTipo() {return tipo;}
    double getValor() {return valor;}
    virtual void amortizar() = 0;

protected:
    int codigo;
    char tipo;
    double valor;
};

// Clases hijas
// Lo ideal seria usar el constructor con el struct, para escribir menos.
// Osea que es al pedo el constructor con los datos por separado.
class Terrenos: public Bienes{
public:
    // Inicializamos el constructor de la clase padre con el :, de esta forma
    // no hay que programar el constructor de cada clase hija.
    Terrenos(int Codigo, char Tipo, double Valor): Bienes(Codigo, Tipo, Valor) {}
    Terrenos(DatosBienes datos): Bienes(datos) {}
    // Terrenos no amortiza asi que no hacemos nada.
    void amortizar() {};
};

class Comunes: public Bienes{
public:
    Comunes(int Codigo, char Tipo, double Valor): Bienes(Codigo, Tipo, Valor) {}
    Comunes(DatosBienes datos): Bienes(datos) {}
    void amortizar();
};

class Muebles: public Bienes{
public:
    Muebles(int Codigo, char Tipo, double Valor): Bienes(Codigo, Tipo, Valor) {}
    Muebles(DatosBienes datos): Bienes(datos) {}
    void amortizar();
};
```

```
    vector<Bienes*> bienes;
};
GestorBienes::GestorBienes(const char* Ruta){
    ifstream archivo (Ruta, ios::binary | ios::ate); // EFICIENTE.
    if (archivo.fail())  // Obligatorio
        return;
    long tamanio = archivo.tellg(); // E
  tamanio /= sizeof(DatosBienes); //E
    archivo.seekg(0, ios::beg); // E
    this->bienes.reserve(tamanio); // E
    DatosBienes aux;
    while( archivo.read((char*) &aux, sizeof(DatosBienes)) ){
            switch (aux.tipo){
          case 'T': bienes.push_back(new Terrenos(aux));
          break;
          case 'C': bienes.push_back(new Comunes(aux));
          break;
          case 'M': bienes.push_back(new Muebles(aux));
          break;
      }}}
void GestorBienes::amortizarBienes(){
    for (auto &bien: bienes)
        bien->amortizar();}
vector <Bienes*> GestorBienes::getBienesMayorValor(){
    sort(bienes.begin(), bienes.end(), [](Bienes* a, Bienes* b) {
        return (a->getValor()) > (b->getValor());
    });
    vector<Bienes*> bienesMayorValor;
    double mayorValor = bienes[0]->getValor();
copy_if(bienes.begin(), bienes.end(), back_inserter(bienesMayorValor), [mayorValor](Bienes* bien){
return bien->getValor() == mayorValor;
    });
    return bienesMayorValor;}
map<char, double> GestorBienes::getTotalPorTipo(){
    map<char, double> totalBienes;
    totalBienes['T'] = 0.0;
    totalBienes['C'] = 0.0;
    totalBienes['M'] = 0.0;
    for (auto &bien: bienes)
        totalBienes[bien->getTipo()] += bien->getValor();
    return totalBienes;}
```

```
        first = last = new Nodo{dato, first, first};
        return;
    }

    last->link = new Nodo{dato, first, last};
    last = last->link;
    first->back = last;
}
template <class T>
T ListaCircular<T>::get(int pos){
    // Aqui podemos usar la solucion recursiva: getRecursive (first, pos);
    // que busca el nodo de forma recursiva, pero tambien les voy a
    // mostrar la solucion iterativa.

    int restar; bool Derecha = true; Nodo * aux = first;
    if (pos < 0) {
        restar = 1;
        Derecha = false;
    } else restar = -1;

    while (pos) {
        if (Derecha)
            aux = aux->link;
        else aux = aux->back;
        pos += restar;
    }

    // Si pos = 0, no va a entrar al while, por lo tanto devuelve el dato del primer nodo.
    return aux->dato;
}

template <class T>
T ListaCircular<T>::getRecursive (Nodo * nodo, int &contador){
    if (!contador) // Es equivalente a contador == 0, porque C++ toma el 0 como false.
        return nodo->dato;

    if (contador < 0)
        return  getRecursive(nodo->back, ++contador);
    else return getRecursive(nodo->link, --contador);
}
```

La empresa Telecom lo contrata dado que desea brindar un servicio de notificaciones para empresas. El sistema deberá poder enviar email y mensajes de texto. Con las siguientes características:

- El email tiene una dirección de email a quien va dirigido y un texto.
- El mensaje de texto un nro telefónico y un texto de no más de 150 caracteres

Los email a enviar están en un archivo binario con el siguiente formato:

int nro
char[100] email
char[500] texto

Los mensajes de texto están en un archivo binario con el siguiente formato:

int nro
int nro_telefono
char[150] texto

El sistema debe crear un archivo de texto que es utilizado para enviar las notificaciones, el cual contiene una línea por notificación. En el caso de email se debe escribir la letra "E" seguida del email y el texto. Y en el caso del mensaje de texto la letra "T", seguida del número de teléfono y el texto. (para realizar esto sobreescriba el operador <<)

El archivo de texto debe ser escrito de forma ordenada por número (campo nro)
Además es necesario saber los email o teléfonos repetidos.

```
#include "notif.h"
#include "telecom.h"
#include <string>

using namespace std;
class email:public notif
{
private:
    string direc;
public:
    email();
    email(emails);
    string getdirec();
    void setdirec(string direc);
};
//———————//

email::email() {}

email::email(emails e){
    this->direc=e.email;
    this->nro=e.nro;
    this->texto=e.texto;
}
string email::getdirec()
{
    return direc;
}

void email::setdirec(string direc)
{
    this->direc=direc;}
```

```
#include "notif.h"
#include "telecom.h"
class sms:public notif
{
private:
    int nro_tel;
public:
    sms();
    sms(mensajes);
    int getnrotel();
};
//———————//
sms::sms() {}

sms::sms(mensajes m)
{
    this->nro=m.nro;
    this->nro_tel=m.nro_telefono;
    this->texto=m.texto;
}
int sms::getnrotel()
{
    return nro_tel;
}
```

```
#include <string>
using namespace std;
class notif
{
public:
    notif();
    int nro;
    string texto;
    virtual int getnro();
    virtual string gettexto();//agregar=0
};
//———————//
notif::notif() {}

int notif::getnro()
{
    return nro;
}

string notif::gettexto()
{
    return texto;
}
```

```
#include "notif.h"
#include <vector>
struct emails{
    int nro;
    char email[100];
    char texto[500];};
struct mensajes{
    int nro;
    int nro_telefono;
    char texto[150];};
class telecom
{
private:
    vector<notif*>N;
public:
    telecom();
    void cargaremail(const char *Ruta);
    void cargarsms(const char *Ruta);
    void creararchi(const char*Ruta);
    void ordenarpornro();
    void repetidos();
};
//—---//
#include "telecom.h"
#include "email.h"
#include "sms.h"
#include <algorithm>
#include <fstream>
#include <map>
telecom::telecom() {}
void telecom::cargaremail(const char* Ruta)
{
    ifstream archivo (Ruta, ios::binary | ios::ate); //
EFICIENTE.
    if (archivo.fail())  // Obligatorio
        return;
    long tamanio = archivo.tellg(); // E
    tamanio /= sizeof(emails); //E
    archivo.seekg(0, ios::beg); // E
    this->N.reserve(tamanio); // E
    emails aux;
    while( archivo.read((char*) &aux, sizeof(emails)) ){
        N.push_back(new email(aux));
    }}
void telecom::cargarsms(const char *Ruta)
{
    ifstream archivo (Ruta, ios::binary | ios::ate); //
EFICIENTE.
    if (archivo.fail())  return;
    long tamanio = archivo.tellg(); // E
    tamanio /= sizeof(mensajes); //E
    archivo.seekg(0, ios::beg); // E
    this->N.reserve(tamanio); // E
    mensajes aux;
    while( archivo.read((char*) &aux, sizeof(mensajes)) ){
        N.push_back(new sms(aux));}}
```

```
void telecom::creararchi(const char *Ruta)
{
    ofstream archi(Ruta);
    if(archi.fail())return;
    ordenarpornro();
    for (const auto& em : N) {
    if (email* e = dynamic_cast<email*>(em)) {
        // Guardar horas para curso simple
        archi << "E";
        archi << e->getdirec()<<" "<<e->gettexto() <<
"\n";
    }
    else if (sms *m = dynamic_cast<sms*>(em)) {
        // Guardar horas para curso con correlatividades
        archi<< "T";
        archi<< m->getnrotel()<<" "<<m->gettexto() <<
"\n";
        }archi<<endl;

        archi.close();
}
}
void telecom::ordenarpornro()
{
    sort(N.begin(),N.end(),[](notif*& a,notif*& b){return
a->getnro()<b->getnro();});
}

void telecom::repetidos()
{
map<string,int>mail;
map<int,int>tel;
for_each(N.begin(),N.end(),[&mail](email& aux1){
mail[aux1.getdirec()]++;});
vector<pair<string, int>> aux(mail.size());
copy(mail.begin(), mail.end(), aux.begin()); //
sort(aux.begin(),aux.end(),[](pair<string,int>aux1,
pair<string,int>aux2){
    return aux1.second > aux2.second;
});
for_each(N.begin(),N.end(),[&tel](sms& saux1){
    tel[saux1.getnrotel()]++;});
vector<pair<int, int>> saux(tel.size()); copy(tel.begin(),
tel.end(), saux.begin()); //
sort(saux.begin(),saux.end(),[](pair<int,int>saux1,
pair<int,int>saux2){
    return saux1.second > saux2.second;
});
 }
```