Carrera: Técnico Universitario en Programación
Materia: Programación II
Tema: Datos y consignas para resolver el parcial 1

Parcial 1 - Programación II

Integrantes de la cátedra

Kloster, Daniel - Profesor

Wenner Maximiliano - Ayudante

Contenido

El parcial consta de 4 preguntas. La primera se debe resolver escribiendo el código en un archivo de CodeBlock en C++, de nombre apellido_nombre.cpp, que luego debe subirse en el link **Entrega parcial 1 02/10/2023,** y las otras seleccionando las opciones que se considere del cuestionario.

La primera pregunta requiere del desarrollo de funciones, y puede necesitar del armado de clases. Debe desarrollarse incluyendo el archivo **parcial1l.h** que se puede bajar desde el link **Entrega parcial 1 02/10/2023.** Este archivo no debe ni copiarse en el cpp que se desarrolle, ni modificarse.

Las preguntas para cada alumno se obtienen ingresando en el cuestionario **Cuestionario parcial 1 lunes 02/10/2023** del aula virtual.

La pregunta 1 tiene un puntaje de 4 puntos; las restantes 2 puntos cada una.

Las opciones que propone el punto 1 se deben seleccionar de acuerdo a lo que cada alumno considere sobre el trabajo que realizó. No tiene efecto en la nota final, ya que cada programa se analiza de manera individual.

El tiempo efectivo de trabajo, una vez aclaradas las dudas sobre el funcionamiento, es de **2 horas**. **Pasado ese tiempo no puede entregarse el examen, por lo que se considerará como ausente.**

La entrega del cuestionario del parcial debe hacerse presionando el botón correspondiente del cuestionario al finalizar la resolución.

Se recuerda que el programa del punto 1 **debe compilar para ser corregido**. **Las respuestas deben adecuarse de manera estricta a lo que el enunciado de cada pregunta pide**.

NO SE CORREGIRAN EXAMENES QUE SE INICIEN MAS ALLA DE LAS 18:35:

Enunciado para preguntas 1 y 2

La Facultad está organizando torneos de deportes entre los integrantes de los distintos claustros. Para gestionar las actividades tiene los siguientes archivos con el formato que se muestra a continuación:

jugadores.dat
DNI
Nombre
Apellido
Email
Teléfono
Claustro (1: docente; 2 alumno; 3 no docente; 4 graduado)
Id de deporte (número entero)
Número de equipo (número entero)
Fecha de inscripción
Matrícula ($)
Estado

deportes.dat
Id de deporte
Nombre
Categoría de deporte
Año de origen
Estado

equipos.dat
Número de equipo: el número del equipo es único y no se repite
Nombre
Nivel (1: inicial; 2: intermedio; 3: avanzado)
Estado

En el caso que fuera necesario el desarrollo de clases nuevas, agregarlas en el cpp donde se desarrollan los puntos. La/s clase/s nuevas deben contener de manera completa todos los métodos que sean necesarios para resolver los problemas que las preguntas plantean.

a) **Generar un archivo con los equipos que tengan al menos 10 jugadores inscriptos. Cada registro debe tener el ID de equipo, el nombre del equipo, y la categoría del deporte al que pertenece el equipo.**
b) **Hacer un vector dinámico para listar el archivo del punto anterior.**
c) **Sobrecargar el operador == para comparar un objeto Jugador con un objeto Deporte. Debe devolver verdadero cuando coinciden los Id de deporte**

a) **Hacer un archivo con los deportes que tengan jugadores de todos los claustros. Los registros del archivo nuevo deben tener el mismo formato que el archivo de deportes.**
b) **Crear un vector dinámico con los equipos de nivel inicial. Listar el vector.**
c) **Sobrecargar el operador == para la clase Jugador, que reciba un valor entero y lo compare con el Id de claustro**

a) **Generar un archivo con los jugadores que hayan pagado más de $10000 de matrícula y que se hayan inscripto este año. Cada registro del archivo debe tener el siguiente formato: DNI, nombre, apellido, claustro y nombre del deporte.**
b) **Crear un vector con los jugadores pertenecientes a un equipo cuyo número de equipo se ingresa por teclado. Listar el vector**
c) **Agregar una sobrecarga para el operador > de la clase Jugador de manera tal que sea verdadero cuando el año de inscripción sea mayor a un valor de año que se recibe cmo parámetro.**

Archivo de cabecera

```
#ifndef PARCIAL1L_H_INCLUDED

#define PARCIAL1L_H_INCLUDED



void cargarCadena(char *pal, int tam){

    int i;

    fflush(stdin);

    for(i=0; i<tam; i++)

    {

        pal[i]=cin.get();

        if(pal[i]=='\n') break;

    }

    pal[i]='\0';

    fflush(stdin);

}
```

```
class Fecha{
private:
    int dia,mes, anio;
public:
    void Cargar(){
        cin>>dia;
        cin>>mes;
        cin>>anio;
    }
    void Mostrar(){
        cout<<dia<<"/";
        cout<<mes<<"/";
        cout<<anio<<endl;
    }
    int getDia(){return dia;}
    int getMes(){return mes;}
    int getAnio(){return anio;}

    void setDia(int d){dia=d;}
    void setMes(int m){mes=m;}
    void setAnio(int a){anio=a;}

};

class Jugador{
private:
    int DNI, claustro, idDeporte, idEquipo;
    char nombre[25], apellido[30];
```

```
    char email[30];
    int telefono;
    Fecha inscripcion;
    float matricula;
    bool estado;
public:
   int getDNI(){return DNI;}
   int getClaustro(){return claustro;}
   int getIDdeporte(){return idDeporte;}
   int getIDequipo(){return idEquipo;}
   const char *getNombre(){return nombre;}
   const char *getApellido(){return apellido;}
   bool getEstado(){return estado;}
   Fecha getFechaInscirpcion(){return inscripcion;}

   void setEstado(bool e){estado=e;}

   void Mostrar(){

   }
} ;

class Deporte{
private:
    int idDeporte, anioOrigen, idCategoria;
    char nombre[30];
    bool estado;
public:
```

```
   int getIDdeporte(){return idDeporte;}
   int getIDcategoria(){return idCategoria;}
   const char *getNombre(){return nombre;}
   bool getEstado(){return estado;}

   void setEstado(bool e){estado=e;}
   void Mostrar(){

   }

};

class Equipo{
private:
    int IdEquipo, nivel;
    char nombre[30];
    bool estado;
public:
    int getIDequipo(){return IdEquipo;}
    int getNivel(){return nivel;}
    const char *getNombre(){return nombre;}
    bool getEstado(){return estado;}

    void setEstado(bool e){estado=e;}
    void Mostrar(){

   }
};
```

```
class ArchivoJugadores{
private:
    char nombre[30];
public:
    ArchivoJugadores(const char *n){
        strcpy(nombre, n);
    }
    Jugador leerRegistro(int pos){
        Jugador reg;
        reg.setEstado(false);
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return reg;
        fseek(p, sizeof reg*pos,0);
        fread(&reg, sizeof reg,1, p);
        fclose(p);
        return reg;
    }
    int contarRegistros(){
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return -1;
        fseek(p, 0,2);
        int tam=ftell(p);
        fclose(p);
        return tam/sizeof(Jugador);
    }
```

```
    bool grabarRegistro(Jugador reg){
        FILE *p;
        p=fopen(nombre, "ab");
        if(p==NULL) return false;
        int escribio=fwrite(&reg, sizeof reg,1, p);
        fclose(p);
        return escribio;
    }
};



class ArchivoDeportes{
private:
    char nombre[30];
public:
    ArchivoDeportes(const char *n){
        strcpy(nombre, n);
    }
    Deporte leerRegistro(int pos){
        Deporte reg;
        reg.setEstado(false);
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return reg;
        fseek(p, sizeof reg*pos,0);
        fread(&reg, sizeof reg,1, p);
        fclose(p);
        return reg;
```

```
    }
    int contarRegistros(){
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return -1;
        fseek(p, 0,2);
        int tam=ftell(p);
        fclose(p);
        return tam/sizeof(Deporte);
    }
    bool grabarRegistro(Deporte reg){
        FILE *p;
        p=fopen(nombre, "ab");
        if(p==NULL) return false;
        int escribio=fwrite(&reg, sizeof reg,1, p);
        fclose(p);
        return escribio;
    }
};

class ArchivoEquipos{
private:
    char nombre[30];
public:
    ArchivoEquipos(const char *n){
        strcpy(nombre, n);
    }
    Equipo leerRegistro(int pos){
```

```
        Equipo reg;
        reg.setEstado(false);
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return reg;
        fseek(p, sizeof reg*pos,0);
        fread(&reg, sizeof reg,1, p);
        fclose(p);
        return reg;
    }
    int contarRegistros(){
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return -1;
        fseek(p, 0,2);
        int tam=ftell(p);
        fclose(p);
        return tam/sizeof(Equipo);
    }
    bool grabarRegistro(Equipo reg){
        FILE *p;
        p=fopen(nombre, "ab");
        if(p==NULL) return false;
        int escribio=fwrite(&reg, sizeof reg,1, p);
        fclose(p);
        return escribio;
    }
};
```

```
#endif // PARCIAL1L_H_INCLUDED

///


# include<iostream>

# include<cstdlib>

# include<cstring>


using namespace std;

///se incluye el punto h necesario

# include "parcial1l.h"


/// a1. Generar un archivo con los equipos que tengan al menos 10 jugadores inscriptos.

///Cada registro debe tener el ID de equipo, el nombre y la categoría.


class Equipos10{

private:

    int IDequipo, IDcategoria;

    char nombre[30];

    bool estado;

public:

    void setEquipo(int e){IDequipo=e;}

    void setNombre(const char *nom){strcpy(nombre, nom);}

    void setCategoria(int cat){IDcategoria=cat;}

    void setEstado(bool e){estado=e;}

    void Mostrar(){

        cout<<"EQUIPO "<<IDequipo<<endl;

        cout<<"NOMBRE "<<nombre<<endl;
```

```
        cout<<"CATEGORIA "<<IDcategoria<<endl;
    }

};

class ArchivoEquipos10{
private:
    char nombre[30];
public:
    ArchivoEquipos10(const char *n){
        strcpy(nombre, n);
    }
    Equipos10 leerRegistro(int pos){
        Equipos10 reg;
        reg.setEstado(false);
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return reg;
        fseek(p, sizeof reg*pos,0);
        fread(&reg, sizeof reg,1, p);
        fclose(p);
        return reg;
    }
    int contarRegistros(){
        FILE *p;
        p=fopen(nombre, "rb");
        if(p==NULL) return -1;
        fseek(p, 0,2);
```

```
        int tam=ftell(p);
        fclose(p);
        return tam/sizeof(Equipos10);
    }
    bool grabarRegistro(Equipos10 reg){
        FILE *p;
        p=fopen(nombre, "ab");
        if(p==NULL) return false;
        int escribio=fwrite(&reg, sizeof reg,1, p);
        fclose(p);
        return escribio;
    }
};

int contarJugadores(int equipo){
    ArchivoJugadores archiJ("jugadores.dat");
    Jugador reg;
    int cantReg, cantJug=0;
    cantReg=archiJ.contarRegistros();
    for(int i=0;i<cantReg;i++){
        reg=archiJ.leerRegistro(i);
        if(reg.getIDequipo()==equipo) cantJug++;
    }
    return cantJug;
}

int buscarDeporte(int equipo){
    ArchivoJugadores archiJ("jugadores.dat");
```

```
    Jugador reg;
    int cantReg;
    cantReg=archiJ.contarRegistros();
    for(int i=0;i<cantReg;i++){
        reg=archiJ.leerRegistro(i);
        if(reg.getIDequipo()==equipo) return reg.getIDdeporte();
    }
    return -1;
}

int buscarCategoria(int deporte){
    ArchivoDeportes archiD("deportes.dat");
    Deporte reg;
    int cantReg;
    cantReg=archiD.contarRegistros();
    for(int i=0;i<cantReg;i++){
        reg=archiD.leerRegistro(i);
        if(reg.getIDdeporte()==deporte) return reg.getIDcategoria();
    }
    return -1;
}

void puntoA1(){
    Equipos10 aux;
    ArchivoEquipos10 archiE10("equipos10.dat");
    Equipo reg;
    ArchivoEquipos archiE("equipos.dat");
    int cantReg, cantJug, categoria, deporte;
```

```
    cantReg=archiE.contarRegistros();
    for(int i=0;i<cantReg;i++){
        reg=archiE.leerRegistro(i);
        cantJug=contarJugadores(reg.getIDequipo());
        if(cantJug>=10){
                deporte=buscarDeporte(reg.getIDequipo());
                categoria=buscarCategoria(deporte);
                aux.setEquipo(reg.getIDequipo());
                aux.setNombre(reg.getNombre());
                aux.setCategoria(categoria);
                archiE10.grabarRegistro(aux);
        }
    }

}

void puntoB1(){
    Equipos10 *v;
    ArchivoEquipos10 archiE10("equipos10.dat");
    int cantReg;
    cantReg=archiE10.contarRegistros();
    if(cantReg==0){
        cout<<"NO HAY REGISTROS EN EL ARCHIVO"<<endl;
        return;
    }
    v=new Equipos10[cantReg];
    if(v==NULL) return;
    for(int i=0;i<cantReg;i++){
```

```
            v[i]=archiE10.leerRegistro(i);
    }
    for(int i=0;i<cantReg;i++){
        v[i].Mostrar();
    }
    delete v;
}


int main(){
    puntoA1();
    puntoB1();
       cout<<endl;
       system("pause");
       return 0;
}




///punto c1
///Sobrecargar el operador == para comparar un objeto Jugador con un objeto Deporte. Debe devolver verdadero cuando coinciden los Id de deporte
///en la clase Jugador iría el código


bool operator==(const Deporte &obj){
    if(idDeporte==obj.getIDdeporte()) return true;
    return false;
}
```