```
/*
 * Proyecto: Examen02_Pregunta02_2022-2
 * Archivo:  Direccion.h
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 24 de noviembre de 2022, 05:22 PM
 */

#ifndef DIRECCION_H
#define DIRECCION_H

struct Direccion{
    char codigo[7];
    char direccion[60];
    char distrito[60];
    double latidud;
    double longidud;
};

#endif /* DIRECCION_H */

/*
 * Proyecto: Examen02_Pregunta02_2022-2
 * Archivo:  ProductoPedido.h
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 24 de noviembre de 2022, 05:30 PM
 */

#ifndef PRODUCTOPEDIDO_H
#define PRODUCTOPEDIDO_H

struct ProductoPedido{
    char nombre[80];
    int cantidad;
    double precio;
};

#endif /* PRODUCTOPEDIDO_H */

/*
 * Proyecto: Examen02_Pregunta02_2022-2
 * Archivo:  Pedido.h
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 24 de noviembre de 2022, 05:23 PM
 */

#ifndef PEDIDO_H
#define PEDIDO_H
#include "Direccion.h"
#include "ProductoPedido.h"

struct Pedido{
    int codigo;
    struct ProductoPedido productos[10];
    int cantProd;
    int hora;
    char repartidorRappi[60];
    char codigoUs[7];
    struct Direccion direccionRappi;
    double distancia;
    double montoTotal;
};

#endif /* PEDIDO_H */

/*
 * Proyecto: Examen02_Pregunta02_2022-2
```

```
 * Archivo:  FuncionesAuxiliares.h
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 24 de noviembre de 2022, 08:24 PM
 */

#ifndef FUNCIONESAUXILIARES_H
#define FUNCIONESAUXILIARES_H

void crearArchivoDirecciones(const char *, const char *);
void pruebaArchivoDirecciones(const char *);
void crearArchivoPedidos(const char *, const char *);
void crearArchivoBinario(const char *);
void completarCamposPedidos(const char *,const char *,const char *);
void pruebaArchivoPedidos(const char *);
int buscarPedido(int ,fstream &);
void datosDelArchivo(int &,int &,fstream &);
void agregaProductos(struct ProductoPedido *,int &, double &, ifstream &);
void completarRappi(const char *,const char *);
void completarDirecciones(const char *,const char *);
void actualizaDirecciones(struct Direccion, const char*);
void imprimeLinea(ofstream &,char ,int);
void ordenarArchivoPedidos(const char *);
void cambiarPedido(int,int , struct Pedido , struct Pedido ,fstream );
void reporteFinal(const char *,const char *);

#endif /* FUNCIONESAUXILIARES_H */

#include <iostream>
#include <iomanip>
using namespace std;
#include "FuncionesAuxiliares.h"

int main(int argc, char** argv) {
    crearArchivoDirecciones("direcciones.csv", "direcciones.bin");
    pruebaArchivoDirecciones("direcciones.bin");

    crearArchivoPedidos("Pedidos.csv", "Pedidos.bin");
    completarCamposPedidos("Pedidos.bin","direcciones.bin","rappi.csv");
    cout<<"Pedidos.bin original"<<endl;
    pruebaArchivoPedidos("Pedidos.bin");
    ordenarArchivoPedidos("Pedidos.bin");
    cout<<"Pedidos.bin ordenado"<<endl;
    pruebaArchivoPedidos("Pedidos.bin");
    reporteFinal("Pedidos.bin","Reporte_tp_rappi.txt");
    return 0;
}

/*
 * Proyecto: Examen02_Pregunta02_2022-2
 * Archivo:  FuncionesAuxiliares.cpp
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 24 de noviembre de 2022, 08:25 PM
 */

#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
#include <cstring>
#include <cmath>
#include "FuncionesAuxiliares.h"
#include "Direccion.h"
#include "Pedido.h"
#define NO_ENCONTRADO -1
#define MAX_CAR_LIN 75

void crearArchivoDirecciones(const char *nomArchCsv, const char *nomArchBin){
```

```
139       ifstream archCsv(nomArchCsv, ios::in);
140       if(not archCsv.is_open()){
141           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchCsv<<endl;
142           exit(1);
143       }
144       ofstream archBin(nomArchBin, ios::out|ios::binary);
145       if(not archBin.is_open()){
146           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
147           exit(1);
148       }
149       struct Direccion direccion;
150       char cod[7], dir[50],dist[50],c;
151       double lat, lon;
152       while(true){
153           archCsv.getline(cod,7,',');
154           if(archCsv.eof())break;
155           archCsv.getline(dir,50,',');
156           archCsv.getline(dist,50,',');
157           archCsv>>lat>>c>>lon>>ws;
158
159           strcpy(direccion.codigo,cod);
160           strcpy(direccion.direccion,dir);
161           strcpy(direccion.distrito,dist);
162           direccion.latidud = lat;
163           direccion.longidud = lon;
164           archBin.write(reinterpret_cast<const char*>(&direccion),
165                         sizeof(struct Direccion));
166       }
167   }
168
169   void crearArchivoPedidos(const char *nomArchCsv, const char *nomArchBin){
170       ifstream archCsv(nomArchCsv, ios::in);
171       if(not archCsv.is_open()){
172           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchCsv<<endl;
173           exit(1);
174       }
175
176       crearArchivoBinario(nomArchBin);
177
178       fstream archBin(nomArchBin, ios::in|ios::out|ios::binary);
179       if(not archBin.is_open()){
180           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
181           exit(1);
182       }
183       int codPed,cant,posPed;
184
185       struct Pedido pedido;
186       while(true){
187           archCsv>>codPed;
188           if(archCsv.eof())break;
189           archCsv.get();
190           posPed = buscarPedido(codPed,archBin);
191           if(posPed != NO_ENCONTRADO){
192               archBin.seekg(posPed*sizeof(struct Pedido),ios::beg);
193               archBin.read(reinterpret_cast<char*>(&pedido),
194                            sizeof(struct Pedido));
195               agregaProductos(pedido.productos,pedido.cantProd,pedido.montoTotal,
196                               archCsv);
197               archBin.seekg(posPed*sizeof(struct Pedido),ios::beg);
198           }
199           else {
200               pedido.codigo = codPed;
201               pedido.cantProd = 0;
202               pedido.montoTotal = 0.0;
203               agregaProductos(pedido.productos,pedido.cantProd,pedido.montoTotal,
204                               archCsv);
205               archBin.seekg(0,ios::end);
206           }
207           archBin.write(reinterpret_cast<const char*>(&pedido),
```

```
                        sizeof(struct Pedido));
        archBin.flush();
    }
}

void crearArchivoBinario(const char *nomArchBin){
    ofstream archBin(nomArchBin, ios::out|ios::binary);
    if(not archBin.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
        exit(1);
    }
}

int buscarPedido(int codPed,fstream &archBin){
    int tamReg, numReg;
    struct Pedido pedido;

    datosDelArchivo(tamReg,numReg, archBin);
    for(int i=0; i<numReg; i++){
        archBin.read(reinterpret_cast<char*>(&pedido),sizeof(struct Pedido));
        if(codPed == pedido.codigo) return i;
    }
    return NO_ENCONTRADO;
}

void datosDelArchivo(int &tamReg,int &numReg,fstream &archPed){
    int tamArch;
    tamReg = sizeof(struct Pedido);
    archPed.seekg(0, ios::end);
    tamArch = archPed.tellg();
    numReg = tamArch/tamReg;
    archPed.seekg(0, ios::beg);
}

void agregaProductos(struct ProductoPedido *producto,int &cantProd,
                     double &montoTotal,ifstream &arch){
    char nomb[80],c;
    int cant;
    double pu;
    while(true){
        arch.getline(nomb,80,',');
        arch>>cant>>c>>pu;
        strcpy(producto[cantProd].nombre,nomb);
        producto[cantProd].cantidad = cant;
        producto[cantProd].precio = pu;
        montoTotal += cant*pu;
        cantProd++;
        if(arch.get() == '\n')break;
    }
}

void completarCamposPedidos(const char *nomArchPed,const char *nomArchDir,
                            const char *nomArchRappi){
    completarRappi(nomArchPed,nomArchRappi);
    completarDirecciones(nomArchPed,nomArchDir);
}

void completarRappi(const char *nomArchPed,const char *nomArchRappi){
    fstream archPed(nomArchPed, ios::in|ios::out|ios::binary);
    if(not archPed.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchPed<<endl;
        exit(1);
    }

    ifstream archRappi(nomArchRappi, ios::in);
    if(not archRappi.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchRappi<<endl;
        exit(1);
    }
```

```
277       int hh, mm, codPed, posPed;
278       char nombRappi[60], codUsuario[7], c;
279       struct Pedido pedido;
280
281       while(true){
282           archRappi>>hh;
283           if (archRappi.eof())break;
284           archRappi>>c>>mm>>c;
285           archRappi.getline(nombRappi,60,',');
286           archRappi.getline(codUsuario,7,',');
287           archRappi>>codPed;
288           posPed = buscarPedido(codPed,archPed);
289           if(posPed != NO_ENCONTRADO){
290               archPed.seekg(posPed*sizeof(struct Pedido),ios::beg);
291               archPed.read(reinterpret_cast<char*>(&pedido),
292                            sizeof(struct Pedido));
293               pedido.hora = hh;
294               strcpy(pedido.repartidorRappi,nombRappi);
295               strcpy(pedido.codigoUs,codUsuario);
296               archPed.seekg(posPed*sizeof(struct Pedido),ios::beg);
297               archPed.write(reinterpret_cast<const char*>(&pedido),
298                            sizeof(struct Pedido));
299               archPed.flush();
300           }
301       }
302   }
303
304   void completarDirecciones(const char *nomArchPed,const char *nomArchDir){
305       ifstream archDir(nomArchDir, ios::in|ios::binary);
306       if(not archDir.is_open()){
307           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchDir<<endl;
308           exit(1);
309       }
310
311       struct Direccion direccion;
312       while(true){
313           archDir.read(reinterpret_cast<char*>(&direccion),
314                        sizeof(struct Direccion));
315           if(archDir.eof())break;
316           actualizaDirecciones(direccion,nomArchPed);
317       }
318   }
319
320   void actualizaDirecciones(struct Direccion direccion, const char*nomArchPed){
321       fstream archPed(nomArchPed, ios::in|ios::out|ios::binary);
322       if(not archPed.is_open()){
323           cout<<"ERROR: no se pudo abrir el archivo "<<nomArchPed<<endl;
324           exit(1);
325       }
326
327       struct Pedido pedido;
328       int tamReg, numReg;
329
330       datosDelArchivo(tamReg,numReg, archPed);
331       for(int i=0; i<numReg; i++){
332           archPed.seekg(i*tamReg, ios::beg);
333           archPed.read(reinterpret_cast<char*>(&pedido),sizeof(struct Pedido));
334           if(archPed.eof())break;
335           if(strcmp(pedido.codigoUs, direccion.codigo)==0){
336               pedido.direccionRappi = direccion;
337               pedido.distancia = sqrt(pow(fabs(-12.0766-direccion.latidud),2)+
338                                       pow(fabs(-77.0833-direccion.longidud),2));
339               archPed.seekg(i*tamReg, ios::beg);
340               archPed.write(reinterpret_cast<const char*>(&pedido),
341                            sizeof(struct Pedido));
342               archPed.flush();
343           }
344       }
345
```

```
}
void pruebaArchivoDirecciones(const char *nomArchBin){
    ifstream archBin(nomArchBin, ios::in|ios::binary);
    if(not archBin.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
        exit(1);
    }
    struct Direccion dir;
    cout.precision(4);
    cout<<fixed;
    while(true){
        archBin.read(reinterpret_cast<char*>(&dir),sizeof(struct Direccion));
        if(archBin.eof())break;
        cout<<left<<setw(10)<<dir.codigo<<setw(40)<<dir.direccion
            <<setw(20)<<dir.distrito<<right<<setw(10)<<dir.latidud
            <<setw(10)<<dir.longidud<<endl;
    }
}

void pruebaArchivoPedidos(const char *nomArchBin){
    ifstream archBin(nomArchBin, ios::in|ios::binary);
    if(not archBin.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
        exit(1);
    }
    struct Pedido pedido;
    int cp;
    cout.precision(2);
    cout<<fixed;
    while(true){
        archBin.read(reinterpret_cast<char*>(&pedido),sizeof(struct Pedido));
        if(archBin.eof())break;
        cout<<right<<setw(5)<<pedido.codigo<<setw(4)<<pedido.cantProd
            <<setw(4)<<pedido.hora<<"  "
            <<left<<setw(30)<<pedido.repartidorRappi
            <<setw(10)<<pedido.codigoUs<<endl;
        cout<<"  "<<left<<setw(30)<<pedido.direccionRappi.direccion
            <<setw(30)<<pedido.direccionRappi.distrito
            <<right<<setw(10)<<pedido.distancia
            <<setw(10)<<pedido.montoTotal<<endl;

        for(int i=0; i<pedido.cantProd;i++){
            cout<<right<<setw(10)<<i+1<<") "
                <<left<<setw(40)<<pedido.productos[i].nombre
                <<right<<setw(4)<<pedido.productos[i].cantidad
                <<setw(10)<<pedido.productos[i].precio<<endl;
        }
    }
}


void ordenarArchivoPedidos(const char *nomArchBin){
    fstream archPed(nomArchBin, ios::in|ios::out|ios::binary);
    if(not archPed.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchBin<<endl;
        exit(1);
    }
    struct Pedido pedidoI,pedidoJ;
    int tamReg,numReg;
    datosDelArchivo(tamReg,numReg, archPed);

    for (int i=0;i<numReg-1;i++)
        for (int j=i+1;j<numReg;j++){
            archPed.seekg(i*sizeof(struct Pedido),ios::beg);
            archPed.read(reinterpret_cast<char*>(&pedidoI),sizeof(struct Pedido));
            archPed.seekg(j*sizeof(struct Pedido),ios::beg);
            archPed.read(reinterpret_cast<char*>(&pedidoJ),sizeof(struct Pedido));
            if ((pedidoI.hora>pedidoJ.hora) or
                    (pedidoI.hora==pedidoJ.hora and
```

```
                    pedidoI.distancia<pedidoJ.distancia)){
                archPed.seekg(i*sizeof(struct Pedido),ios::beg);
                archPed.write(reinterpret_cast<const char*>(&pedidoJ),sizeof(struct
                Pedido));
                archPed.seekg(j*sizeof(struct Pedido),ios::beg);
                archPed.write(reinterpret_cast<const char*>(&pedidoI),sizeof(struct
                Pedido));
            }
        }
}

void reporteFinal(const char *nomArchPed, const char *nomArchRep){
    ofstream archRep(nomArchRep, ios::out);
    if(not archRep.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchRep<<endl;
        exit(1);
    }

    ifstream archPed(nomArchPed, ios::in|ios::binary);
    if(not archPed.is_open()){
        cout<<"ERROR: no se pudo abrir el archivo "<<nomArchPed<<endl;
        exit(1);
    }
    struct Pedido pedido;
    int cp;
    archRep.precision(2);
    archRep<<fixed;
    archRep<<"tp_rappi"<<endl;
    archRep<<"REPORTE DE PEDIDOS"<<endl;
    imprimeLinea(archRep,'=',MAX_CAR_LIN);
    archRep<<left<<setw(7)<<"PEDIDO"<<setw(20)<<"Nombre Rappi"<<setw(7)<<"HORA"
           <<setw(25)<<"DISTANCIA EUCLIDIANA"
           <<setw(15)<<"PRECIO TOTAL"<<endl;
    imprimeLinea(archRep,'-',MAX_CAR_LIN);
    while(true){
        archPed.read(reinterpret_cast<char*>(&pedido),sizeof(struct Pedido));
        if(archPed.eof())break;
        archRep<<right<<setw(5)<<pedido.codigo<<"
        "<<left<<setw(20)<<pedido.repartidorRappi
           <<right<<setw(4)<<pedido.hora<<setw(16)<<pedido.distancia
           <<setw(20)<<pedido.montoTotal<<endl;
    }
}


void imprimeLinea(ofstream &archRep,char car,int n){
    for(int i=0; i<n; i++) archRep.put(car);
    archRep<<endl;
}
```