1. Código

src/fatFood.cpp

```cpp
#include <vector>
#include <iostream>
#include <utility>
#include "interfaz.h"
#include "local.h"

using namespace std;

int main(){
    Local l(
        vector<pair<Bebida,Cantidad>> {{PestiCola,5}},
        vector<pair<Hamburguesa,Cantidad>> {{McGyver,5}},
        vector<Empleado> {"Hello"}
    );

    //l.guardar(cout);
    MenuPrincipal();
    return 0;
}
```

src/tipos.h

```cpp
#ifndef TIPOS_H
#define TIPOS_H

#include <vector>
#include <stdlib.h>

#include <string>
#include <iostream>

using namespace std;
typedef string Empleado;
typedef int Energia;
typedef int Cantidad;

static const char* BEBIDA_STR[] =
    {"PestiCola","FalsaNaranja","SeVeNada", "AguaConGags","AguaSinGags"};
static const char* HAMBURGUESA_STR[] =
    {"McGyver","CukiQueFresco","McPato","BigMacabra"};

enum Bebida{PestiCola, FalsaNaranja, SeVeNada, AguaConGags, AguaSinGags};
enum Hamburguesa{McGyver, CukiQueFresco, McPato, BigMacabra};

#define MaxH 3
#define MaxB 4

#endif /*TIPOS_H*/
```

src/combo.h

```cpp
#ifndef COMBO_H_INCLUDED
#define COMBO_H_INCLUDED

#include <vector>
#include "tipos.h"
#include "auxiliar.h"

class Combo {
```

```
9
10      public:
11
12          Combo();
13          Combo(const Bebida b, const Hamburguesa h, const Energia d);
14
15          Bebida      bebidaC() const;
16          Hamburguesa sandwichC() const;
17          Energia     dificultadC() const;
18
19          void mostrar(std::ostream& os) const;
20          void guardar(std::ostream& os) const;
21          void cargar (std::istream& is);
22
23          bool operator==(const Combo& otroCombo) const;
24
25      private:
26
27          Bebida      _bebida;
28          Hamburguesa _sandwich;
29          Energia     _dificultad;
30
31          enum {ENCABEZADO_ARCHIVO = 'C'};
32
33  };
34
35  // Definirlo usando mostrar, para poder usar << con este tipo.
36  std::ostream & operator<<(std::ostream & os,const Combo & c);
37  std::ostream & operator<<(std::ostream & os,const Hamburguesa & c);
38  std::ostream & operator<<(std::ostream & os,const Bebida & c);
39
40  std::istream & operator>>(std::istream & is, Combo & c);
41  std::istream & operator>>(std::istream & is, Hamburguesa & c);
42  std::istream & operator>>(std::istream & is, Bebida & c);
43
44  #endif // COMBO_H_INCLUDED
```

src/combo.cpp

```
1   #include "combo.h"
2   #include <string.h>
3   #include <stdexcept>
4
5   Combo::Combo(){
6   }
7
8   Combo::Combo(const Bebida b, const Hamburguesa h, const Energia d ){
9       _bebida = b;
10      _sandwich = h;
11      _dificultad = d;
12  }
13
14  Bebida      Combo::bebidaC() const{
15      return _bebida;
16  }
17
18  Hamburguesa Combo::sandwichC() const{
19      return _sandwich;
20  }
21
22  Energia     Combo::dificultadC() const{
23      return _dificultad;
24  }
```

```
25
26 void Combo::mostrar(std::ostream& os) const{
27      os<<"Bebida: "<<bebidaC()<<endl;
28      os<<"Sandwich: "<<sandwichC()<<endl;
29      os<<"Dificultad: "<<dificultadC();
30 }
31
32 void Combo::guardar(std::ostream& os) const{
33     os << '{'<<" "
34        << (char)ENCABEZADO_ARCHIVO<<" "
35        << _bebida<<" "
36        << _sandwich<<" "
37        << _dificultad<<" "
38        << '}';
39 }
40
41 void Combo::cargar (std::istream& is){
42     char head;
43
44     is >> head; //Inicio {
45
46     is >> head;
47     if (head != ENCABEZADO_ARCHIVO){
48         throw std::invalid_argument("Encabezado invÃ¡lido");
49     }
50     is >> _bebida
51        >> _sandwich
52        >> _dificultad
53        >> head; //Fin }
54 }
55
56 bool Combo::operator==(const Combo& otroCombo) const{
57     return _sandwich == otroCombo.sandwichC() &&
58            _bebida == otroCombo.bebidaC() &&
59            _dificultad == otroCombo.dificultadC();
60 }
61
62 std::ostream & operator<<(std::ostream & os,const Combo & c){
63     c.mostrar(os);
64     return os;
65 }
66
67 std::ostream & operator<<(std::ostream & os,const Hamburguesa & h){
68     os << HAMBURGUESA_STR[h];
69     return os;
70 }
71
72 std::ostream & operator<<(std::ostream & os,const Bebida & b){
73     os << BEBIDA_STR[b];
74     return os;
75 }
76
77 std::istream & operator>>(std::istream & is, Combo & c){
78     c.cargar(is);
79     return is;
80 }
81
82 std::istream & operator>>(std::istream & is, Hamburguesa & c){
83     string s;
84     is >> s;
85     for(int i=0; i<=MaxH; i++){
86         if(HAMBURGUESA_STR[i] == s){
87             c = (Hamburguesa)i;
```

```
            break;
        }
    }
    return is;
}

std::istream & operator>>(std::istream & is, Bebida & b){
    string s;
    is >> s;
    for(int i=0; i<=MaxB; i++){
        if(BEBIDA_STR[i] == s){
            b = (Bebida)i;
            break;
        }
    }
    return is;
}
```

src/pedido.h

```
#ifndef PEDIDO_H_INCLUDED
#define PEDIDO_H_INCLUDED

#include <vector>
#include <stdexcept>
#include "tipos.h"
#include "combo.h"
#include "auxiliar.h"

class Pedido {

    public:

        Pedido();
        Pedido(const int n, const Empleado e, const vector<Combo> cs);

        int             numeroP() const;
        Empleado        atendioP() const;
        vector<Combo>    combosP() const;

                Energia dificultadP() const;

        void  agregarComboP(const Combo c);
        void  anularComboP(int i);
        void  cambiarBebidaComboP(const Bebida b, int i);
        void  elMezcladitoP();

        void mostrar(std::ostream& os) const;
        void guardar(std::ostream& os) const;
        void cargar (std::istream& is);

        bool operator==(const Pedido& otroPedido) const;

    private:

        vector<Combo> _combos;
        Empleado     _atendio;
        int          _numero;

        enum {ENCABEZADO_ARCHIVO = 'P'};

};

```

```
// Auxiliares
int countBebidasP(const Pedido&, Bebida);
int countSandwichesP(const Pedido&, Hamburguesa);

// Definirlo usando mostrar, para poder usar << con este tipo.
std::ostream & operator<<(std::ostream & os,const Pedido & p);
std::istream & operator>>(std::istream & is, Pedido & p);

#endif // PEDIDO_H_INCLUDED
```

src/pedido.cpp

```
#include "pedido.h"
#include <algorithm>
vector<Combo> combosDelPedidoSinRep (const Pedido *p);
vector<Combo> combosRepetidos (const Pedido *p);
vector<Bebida> bebidasDelPedidoSinRep (const Pedido *p);
vector<Hamburguesa> sandwichesDelPedidoSinRep (const Pedido *p);
bool estaC(const Combo c, const vector<Combo> cs);
void cambiarBebida (Combo &c, const vector<Bebida> bs);
void cambiarSandwich (Combo &c, const vector<Hamburguesa> hs);

Pedido::Pedido(){
}

Pedido::Pedido(const int nro, const Empleado e, const vector<Combo> combos){
    _numero = nro;
    _atendio = e;
    _combos = combos;
}

int Pedido::numeroP() const{
    return _numero;
}

Empleado Pedido::atendioP() const{
    return _atendio;
}

vector<Combo> Pedido::combosP() const{
    return _combos;
}

Energia Pedido::dificultadP() const{
    int n=combosP().size();
    int sum=0;
    int i=0;
    while(i<n){
        sum=sum+combosP()[i].dificultadC();
        i++;
    }
    return sum;
}

void  Pedido::agregarComboP(const Combo c){
    _combos.push_back(c);
}

void  Pedido::anularComboP(int i){
    _combos.erase(_combos.begin()+i);
}

void  Pedido::cambiarBebidaComboP(const Bebida b, int i){
```

```
52      Combo old = _combos[i];
53      _combos[i] = Combo(b,old.sandwichC(),old.dificultadC());
54  }
55
56  void  Pedido::elMezcladitoP(){
57      int t = combosP().size();
58      vector<Combo> res = combosDelPedidoSinRep (this);
59      for(auto &i : combosRepetidos(this)){
60          int j=0;
61          int n = bebidasDelPedidoSinRep(this).size();
62          int m = sandwichesDelPedidoSinRep(this).size();
63          while (j<n){
64              int k=0;
65              Hamburguesa s=i.sandwichC();
66              while (k<m){
67                  cambiarSandwich(i,sandwichesDelPedidoSinRep(this));
68                      if (estaC(i,res)){
69                          k++;
70                      }
71                      else{
72                          res.push_back(i);
73                          k=m;
74                      }
75                  }
76
77              if (i.sandwichC()==s){
78                  cambiarBebida(i,bebidasDelPedidoSinRep(this));
79                  if (estaC(i,res)){
80                      j++;
81                  }
82                  else{
83                      res.push_back(i);
84                      j=n;
85                  }
86              }
87              else{
88                  j=n;
89              }
90          }
91      }
92      int a;
93      for(a=0;a<t;a++){
94      _combos[a]=res[a];
95      }
96  }
97
98  void Pedido::mostrar(std::ostream& os) const{
99      os<<"Numero de pedido: "<<numeroP()<<endl;
100      os<<"Empleado que atendio el pedido: "<<atendioP()<<endl;
101      os<<"Los combos que integran el pedido son: "<<endl;
102      for (int i=0;i<combosP().size();i++){
103      os<<"Sandwich: "<<combosP()[i].sandwichC()<<" Bebida:
         "<<combosP()[i].bebidaC()<<" Dificultad: "<<combosP()[i].dificultadC()<<endl;
104      }
105  }
106
107  void Pedido::guardar(std::ostream& os) const{
108      os << '{' << " "
109         << (char)ENCABEZADO_ARCHIVO << " "
110         << _numero << " "
111         << _atendio << " [ ";
112      for (auto &c : _combos){
113          c.guardar(os);
```

```
114         os << " ";
115     }
116     os << "] }";
117 }
118 
119 void Pedido::cargar (std::istream& is){
120     char aux;
121     is.ignore(0xff,'{'); // "  {"
122     trimIS(is);
123 
124     if(is.get() != ENCABEZADO_ARCHIVO){
125         throw std::invalid_argument("Encabezado invÃ¡lido");
126     }
127     is >> _numero
128        >> _atendio
129        >> _combos;
130     is.ignore(0xff,'}'); // "  }"
131 }
132 
133 bool Pedido::operator==(const Pedido& otroPedido) const{
134     return atendioP() == otroPedido.atendioP() &&
135            numeroP() == otroPedido.numeroP() &&
136            combosP() == otroPedido.combosP();
137 }
138 
139 // Auxiliares
140 int countBebidasP(const Pedido& p, Bebida b){
141     int count = 0;
142     for(auto &c : p.combosP()){
143         if(c.bebidaC() == b)
144             count++;
145     }
146     return count;
147 }
148 
149 int countSandwichesP(const Pedido& p, Hamburguesa s){
150     int count = 0;
151     for(auto &c : p.combosP()){
152         if(c.sandwichC() == s)
153             count++;
154     }
155     return count;
156 }
157 
158 bool estaC(const Combo c, const vector<Combo> cs){
159 int i = 0;
160 int n = cs.size();
161 while (i < n && (cs[i].bebidaC() != c.bebidaC() || cs[i].sandwichC() !=
    c.sandwichC())){
162 i = i + 1;
163 }
164 return i < n;
165 }
166 
167 bool estaB(const Bebida b, const vector<Bebida> bs){
168 int i = 0;
169 int n = bs.size();
170 while (i < n && bs[i] != b){
171 i = i + 1;
172 }
173 return i < n;
174 }
175 
```

```
bool estaS(const Hamburguesa h, const vector<Hamburguesa> hs){
int i = 0;
int n = hs.size();
while (i < n && hs[i] != h){
i = i + 1;
}
return i < n;
}

vector<Bebida> bebidasDelPedido (const Pedido *p){
    vector<Bebida> res;
    res.reserve(p->combosP().size());
    for (auto &i : p->combosP()){
        res.push_back(i.bebidaC());
        }
    return res;
}

vector<Bebida> bebidasDelPedidoSinRep (const Pedido *p){
    vector<Bebida> res;
    res.reserve(bebidasDelPedido(p).size());
    for (auto &i : bebidasDelPedido(p)){
        if (!estaB(i,res)){
        res.push_back(i);
        }
    }
    return res;
}

vector<Hamburguesa> sandwichesDelPedido (const Pedido *p){
    vector<Hamburguesa> res;
    res.reserve(p->combosP().size());
    for (auto &i : p->combosP()){
        res.push_back(i.sandwichC());
        }
    return res;
}

vector<Hamburguesa> sandwichesDelPedidoSinRep (const Pedido *p){
    vector<Hamburguesa> res;
    res.reserve(sandwichesDelPedido(p).size());
    for (auto &i : sandwichesDelPedido(p)){
        if (!estaS(i,res)){
        res.push_back(i);
        }
    }
    return res;
}

vector<Combo> combosDelPedidoSinRep (const Pedido *p){
    vector<Combo> res;
    res.reserve(p->combosP().size());
    for (auto &i : p->combosP()){
        if (!estaC(i,res)){
        res.push_back(i);
        }
    }
    return res;
}

vector<Combo> combosRepetidos (const Pedido *p){
    vector<Combo> res;
    vector<Combo> aux = p->combosP();
```

```
239     int n = p->combosP().size();
240     int i =0;
241     while (i<n){
242         aux.erase(aux.begin()+i);
243         if (estaC(p->combosP()[i],aux)){
244         res.push_back(p->combosP()[i]);
245         i++;
246         }
247         else{
248             i++;
249         }
250     }
251     return res;
252 }
253 
254 void cambiarBebida (Combo &c, const vector<Bebida> bs){
255     int n = bs.size();
256     int i = 0;
257     while (i<n)
258         if (c.bebidaC()==bs[i]){
259             c= Combo(bs[(i+1) % n],c.sandwichC(), c.dificultadC());
260             i=n;
261         }
262         else{
263             i++;
264         }
265 }
266 
267 void cambiarSandwich (Combo &c, const vector<Hamburguesa> hs){
268     int n = hs.size();
269     int i = 0;
270     while (i<n)
271         if (c.sandwichC()==hs[i]){
272             c= Combo(c.bebidaC(),hs[(i+1) % n], c.dificultadC());
273             i=n;
274         }
275         else{
276             i++;
277         }
278 }
279 
280 std::ostream & operator<<(std::ostream & os,const Pedido& p){
281     p.mostrar(os);
282     return os;
283 }
284 
285 std::istream & operator>>(std::istream & is, Pedido & p){
286     p.cargar(is);
287     return is;
288 }
```

src/local.h

```
1  #ifndef LOCAL_H_INCLUDED
2  #define LOCAL_H_INCLUDED
3  
4  #include <vector>
5  #include "pedido.h"
6  #include "auxiliar.h"
7  #include "combo.h"
8  #include "pedido.h"
9  
10 class Local{
```

```
    public:

        Local();
        Local(const vector< pair <Bebida,Cantidad> > bs, const vector< pair
           <Hamburguesa,Cantidad> > hs, const vector<Empleado> es);

        Cantidad            stockBebidasL(const Bebida b) const;
        Cantidad            stockSandwichesL(const Hamburguesa h) const;
        vector<Bebida>       bebidasDelLocalL() const;
        vector<Hamburguesa>  sandwichesDelLocalL() const;
        vector<Empleado>     empleadosL() const;
        vector<Empleado>     desempleadosL() const;
        Energia             energiaEmpleadoL(const Empleado e) const;
        vector<Pedido>       ventasL() const;
        bool                unaVentaCadaUnoL() const;
        void                venderL(const Pedido p);
        vector<Empleado>     candidatosAEmpleadosDelMesL() const;
        void                sancionL(const Empleado e, const Energia n);
        Empleado            elVagonetaL() const;
        void                anularPedidoL(int n);
        void                agregarComboAlPedidoL(const Combo c, int n);


        void mostrar(std::ostream& os) const;
        void guardar(std::ostream& os) const;
        void cargar (std::istream& is);

    private:
        vector< pair <Hamburguesa,Cantidad> > _sandwiches;
        vector< pair <Bebida,Cantidad> >      _bebidas;
        vector< pair <Empleado, Energia> >    _empleados;
        vector< Pedido >                      _ventas;



        enum {ENCABEZADO_ARCHIVO = 'L'};

};

std::ostream & operator<<(std::ostream & os,const Local & c);
std::istream & operator>>(std::istream & is, Local & l);

#endif // LOCAL_H_INCLUDED
```

src/local.cpp

```
#include "local.h"
#include <string.h>
#include <algorithm>

vector<Pedido> pedidosDelEmpleado();
int maxDescansoEmplado(Empleado);
vector< pair<Empleado, int> > empleadoYdescanso(const Local *l);
vector< pair <Empleado, Energia> > empleadosYenegiaL(const Local *l);
vector<Empleado> empleadosConMasVentas(const Local *l);
int maxCantPedidos(const Local *l);
int maxCantCombos(const Local *l);
vector<Combo> combosDelEmpleado(const Local *l,Empleado e);

Local::Local() {
}

Local::Local(const vector< pair<Bebida,Cantidad> > bs,
```

```
18              const vector< pair<Hamburguesa,Cantidad> > hs,
19              const vector<Empleado> es) {
20      _bebidas = bs;
21      _sandwiches = hs;
22
23      _empleados.reserve(es.size());
24      for(auto &i : es){
25          _empleados.push_back(pair<Empleado,Energia>(i,100));
26      }
27  }
28
29  Cantidad Local::stockBebidasL(const Bebida b) const{
30      int n = _bebidas.size();
31      int i = 0;
32      int stock;
33      while (i<n){
34          if (_bebidas[i].first != b){
35              i++;
36          }
37          else{
38              stock= _bebidas[i].second;
39              i=n;
40          }
41      }
42      return stock;
43  }
44
45  Cantidad Local::stockSandwichesL(const Hamburguesa h) const{
46      int n = _sandwiches.size();
47      int i = 0;
48      int stock;
49      while (i<n){
50          if (_sandwiches[i].first != h){
51              i++;
52          }
53          else{
54              stock= _sandwiches[i].second;
55              i=n;
56          }
57      }
58      return stock;
59  }
60
61  vector<Bebida>       Local::bebidasDelLocalL() const{
62      vector<Bebida> res;
63      res.reserve(_bebidas.size());
64      for (auto &i : _bebidas){
65          res.push_back(i.first);
66      }
67      return res;
68  }
69
70  vector<Hamburguesa>  Local::sandwichesDelLocalL() const{
71      vector<Hamburguesa> v;
72      v.reserve(_sandwiches.size());
73      for(auto &i : _sandwiches){
74          v.push_back(i.first);
75      }
76      return v;
77  }
78
79  vector<Empleado>     Local::empleadosL() const{
80      vector<Empleado> res;
```

```
    res.reserve(_empleados.size());
    for (auto &i : _empleados){
        if(i.second>=0){
        res.push_back(i.first);
        }
    }
    return res;
}


vector<Empleado>     Local::desempleadosL() const{
     vector<Empleado> res;
    res.reserve(_empleados.size());
    for (auto &i : _empleados){
        if(i.second<0){
        res.push_back(i.first);
        }
    }
    return res;
}

Energia  Local::energiaEmpleadoL(const Empleado e) const{
    for (auto &i : _empleados)
        if (i.first == e)
            return i.second;
}

vector<Pedido>       Local::ventasL() const{
    return _ventas;
}

vector<Empleado> Local::candidatosAEmpleadosDelMesL() const{
    vector<Empleado> res;
    vector<Empleado> emp = empleadosConMasVentas(this);
    int n = emp.size();
    int i = 0;
    int maxCombos = maxCantCombos(this);
    while (i<n){
        if (combosDelEmpleado(this,emp[i]).size() == maxCombos){
        res.push_back(emp[i]);
        i++;
        }
        else{
        i++;
        }
    }
    return res;
}

void Local::venderL(const Pedido p){
    //duda: si no se cumple los requiere debo decir que no se cumplieron o
       simplemente se supone que se cumplen?
   bool exito=true;
   for (int i=0; i<_empleados.size();i++)
   {
        if (_empleados[i].first==p.atendioP())
        {
            _empleados[i].second= _empleados[i].second-p.dificultadP();
            if (_empleados[i].second<0)
            {
                exito=false;
            }
        }
```

```
143     }
144     if (exito)
145     {
146 
147      for (int i=0; i<_bebidas.size();i++)
148      {
149          for (int j=0; j<p.combosP().size();j++)
150          {
151              if (_bebidas[i].first==p.combosP()[j].bebidaC())
152              {
153                  _bebidas[i].second--;
154              }
155          }
156      }
157     for (int i=0; i<_sandwiches.size();i++)
158     {
159          for (int j=0; j<p.combosP().size();j++)
160          {
161              if (_sandwiches[i].first==p.combosP()[j].sandwichC())
162              {
163                  _sandwiches[i].second--;
164              }
165          }
166     }
167     _ventas.push_back(p);
168     }
169 }
170 
171 void Local::sancionL(const Empleado e, const Energia n){
172     for (auto &i : _empleados)
173         if (i.first == e)
174             i.second -= n;
175 }
176 
177 void Local::anularPedidoL(int n){
178     auto esNumeroN = [n](Pedido p){return p.numeroP() == n;};
179     auto itPedido = find_if(_ventas.begin(),_ventas.end(),esNumeroN);
180 
181     // Modificar energia y stock
182     for(auto &empleado : _empleados){
183         if(empleado.first == itPedido->atendioP())
184             empleado.second += itPedido->dificultadP();
185     }
186     for(auto &bebida : _bebidas){
187         bebida.second += countBebidasP(*itPedido,bebida.first);
188     }
189     for(auto &sandwich : _sandwiches){
190         sandwich.second += countSandwichesP(*itPedido,sandwich.first);
191     }
192 
193     // Cambiar los numeros de pedido
194     for(Pedido &pedido : _ventas){
195         if(pedido.numeroP() > n)
196             pedido = Pedido(pedido.numeroP()-1,
197                             pedido.atendioP(),
198                             pedido.combosP());
199     }
200 
201     _ventas.erase(itPedido);
202 }
203 
204 void Local::agregarComboAlPedidoL(const Combo c, int n){
205     Pedido p = _ventas[0];
```

```
206     int m = _ventas.size();
207     int i=0;
208     int j=0;
209     while (i<m){
210         if (_ventas[i].numeroP()==n){
211             p=_ventas[i];
212             j=i;
213             i=m;
214         }
215         else{
216             i++;
217         }
218     }
219     int b;
220     int s;
221     int e;
222     for(i=0;i<_bebidas.size();i++){
223         if (_bebidas[i].first == c.bebidaC()){
224             b=i;
225         }
226     }
227     for(i=0;i<_sandwiches.size();i++){
228         if (_sandwiches[i].first == c.sandwichC()){
229             s=i;
230         }
231     }
232     for(i=0;i<_empleados.size();i++){
233         if (_empleados[i].first == p.atendioP()){
234             e=i;
235         }
236     }
237     _bebidas[b].second--;
238     _sandwiches[s].second--;
239     _empleados[e].second-=c.dificultadC();
240     vector<Combo> cs = p.combosP();
241     cs.push_back(c);
242     p= Pedido(p.numeroP(), p.atendioP(), cs);
243     _ventas[j]=p;
244 }
245
246 bool Local::unaVentaCadaUnoL() const{
247     vector<Pedido> v=_ventas;
248     sort(v. begin(), v. end(), [] (Pedido p1, Pedido p2) {return p1.numeroP() <
        p2.numeroP();});
249     int i=0;
250     while(i<v.size()) {
251         bool pertenece=false;
252         for (int j=0;j<_empleados.size();j++) {
253             if ((v[i].atendioP()==_empleados[j].first)&& (_empleados[j].second>=0)) {
254                     pertenece=true;
255             }
256         }
257         if (pertenece==false) {
258             v.erase(v.begin()+i);
259             i--;
260         }
261         i++;
262     }
263     if (v.size()<=1)
264     {
265         return true;
266     }
267     if (v.size()>1)
```

```
268     {
269         bool estado=true;
270         for (int i=0;i<v.size()-1;i++)
271         {
272             for (int j=i+1;j<v.size();j++)
273             {
274                 if (v[i].atendioP()==v[j].atendioP())
275                 {
276                     if ((j+1<v.size())&&(!(v[i+1].atendioP()==v[j+1].atendioP())))
277                     {
278                         estado=false;
279                     }
280                 }
281             }
282         }
283          return estado;
284     }
285 
286 }
287 
288 Empleado Local::elVagonetaL() const{
289     vector< pair<Empleado,int> > empYdes = empleadoYdescanso(this);
290     int i = 1, n = empYdes.size();
291     int vago = 0;
292     while (i < n){
293         if (empYdes[i].second > empYdes[vago].second)
294             vago = i;
295         i++;
296     }
297     return empYdes[vago].first;
298 }
299 
300 void Local::guardar(std::ostream& os) const{
301     os << "{ " << (char)ENCABEZADO_ARCHIVO << " "
302        << _bebidas << " "
303        << _sandwiches << " "
304        << _empleados << " [ ";
305     for (auto &e : _ventas){
306         e.guardar(os);
307         os << " ";
308     }
309     os << "] }";
310 }
311 
312 
313 
314 void Local::mostrar(std::ostream& os) const{
315     os<<"Bebidas del local y su stock: "<<_bebidas<<endl;
316     os<<"Sandwiches del local y su stock: "<<_sandwiches<<endl;
317     os<<"Empleados del local y su energÃa: "<<empleadosYenegiaL(this)<<endl;
318     os<<"Desempleados del local: "<<desempleadosL()<<endl;
319     os<<"Ventas de local:" << endl;
320 
321     for(auto p : _ventas){
322         os << endl << p << endl;
323     }
324 }
325 
326 void Local::cargar (std::istream& is){
327     char head;
328     is >> head; //Inicio {
329 
330     is >> head;
```

```
331     if (head != ENCABEZADO_ARCHIVO){
332         throw std::invalid_argument("Encabezado invÃ¡lido");
333     }
334     is >> _bebidas
335        >> _sandwiches
336        >> _empleados
337        >> _ventas;
338 
339     is >> head; //Fin }
340 }
341 
342 std::ostream & operator<<(std::ostream & os,const Local & l){
343     l.mostrar(os);
344     return os;
345 }
346 
347 std::istream & operator>>(std::istream & is, Local & l){
348     l.cargar(is);
349     return is;
350 }
351 
352 
353 //Shhhh aca no pasa nada
354 vector<Pedido> pedidosDelEmpleado(const Local *l,Empleado e){
355     vector<Pedido> res;
356     for (auto &i : l->ventasL())
357         if (i.atendioP()==e)
358             res.push_back(i);
359     return res;
360 }
361 
362 int maxDescansoEmpleado(const Local *l,Empleado e){
363     vector<Pedido> pedidos = pedidosDelEmpleado(l,e);
364     if (pedidos.size()==0)
365         return l->ventasL().size();
366     int maxDescanso = pedidos[0].numeroP() - l->ventasL()[0].numeroP();
367     for (int i = 1; i < pedidos.size(); i++)
368         maxDescanso = max(pedidos[i].numeroP() - pedidos[i-1].numeroP(),maxDescanso);
369     maxDescanso = max(pedidos.back().numeroP() -
        l->ventasL().back().numeroP(),maxDescanso);
370     return maxDescanso;
371 }
372 
373 vector< pair<Empleado, int> > empleadoYdescanso(const Local *l)
374 {
375     vector< pair<Empleado,int> > res;
376     vector<Empleado> emp = l->empleadosL();
377     res.reserve(emp.size());
378     for(auto &i : emp)
379         res.push_back(pair<Empleado,int>(i,maxDescansoEmpleado(l,i)));
380     return res;
381 }
382 
383 vector< pair <Empleado, Energia> >  empleadosYenegiaL(const Local *l){
384     vector< pair <Empleado, Energia> >  res;
385     vector<Empleado> emp = l->empleadosL();
386     res.reserve(emp.size());
387     for (auto &i : emp)
388         res.push_back(pair<Empleado,Energia>(i,l->energiaEmpleadoL(i)));
389     return res;
390 }
391 vector<Empleado> empleadosConMasVentas(const Local *l){
392     vector<Empleado> res;
```

```
393     vector<Empleado> emp = l->empleadosL();
394     int n = emp.size();
395     int i = 0;
396     while (i<n){
397         if (pedidosDelEmpleado(l,emp[i]).size() == maxCantPedidos(l)){
398         res.push_back(emp[i]);
399         i++;
400         }
401         else{
402         i++;
403         }
404     }
405     return res;
406 }
407
408 int maxCantPedidos(const Local *l){
409     int res;
410     vector<Empleado> emp = l->empleadosL();
411     int n = emp.size();
412     int i=1;
413     Empleado e = emp[0];
414     while (i<n){
415         if (pedidosDelEmpleado(l,emp[i]).size() >= pedidosDelEmpleado(l,e).size()){
416             e=emp[i];
417             i++;
418         }
419         else{
420             i++;
421         }
422     }
423     res = pedidosDelEmpleado(l,e).size();
424     return res;
425 }
426
427 int maxCantCombos(const Local *l){
428     int res;
429     vector<Empleado> emp = empleadosConMasVentas(l);
430     int n = emp.size();
431     int i=1;
432     Empleado e = emp[0];
433     while (i<n){
434         if (combosDelEmpleado(l,emp[i]).size() >= combosDelEmpleado(l,e).size()){
435             e=emp[i];
436             i++;
437         }
438         else{
439             i++;
440         }
441     }
442     res = combosDelEmpleado(l,e).size();
443     return res;
444 }
445
446 vector<Combo> combosDelEmpleado(const Local *l,Empleado e){
447     vector<Combo> res = vector<Combo>(1);
448     for (auto &i : l->ventasL())
449         if (i.atendioP()==e)
450             for (auto &j : i.combosP())
451                 res.push_back(j);
452     return res;
453 }
```

src/auxiliar.h

```cpp
#ifndef AUX_H_INCLUDED
#define AUX_H_INCLUDED

#include <string>
#include <vector>
#include <istream>
#include "tipos.h"

// Stream functions

void trimIS(std::istream & is);

// Vector and pair I/O

template<typename T, typename A>
std::ostream & operator<<(std::ostream & os,const vector<T,A> & v){
    os << "[ ";
    for(auto &i : v){
        os << i << " ";
    }
    os << "]";
    return os;
}

template<typename T, typename Q>
std::ostream & operator<<(std::ostream & os,const pair<T,Q> & p){
    os << "( " << p.first << " " << p.second << " )";
    return os;
}

template<typename T>
std::istream & operator>>(std::istream & is, vector<T> & v){
    T t;

    is.ignore(0xff,'['); // "  {"
    trimIS(is);
    while(is.peek() != ']'){
        is >> t;
        v.push_back(t);
        trimIS(is);
    }
    is.ignore(0xff,']'); // "  {"
    return is;
}

template<typename T, typename Q>
std::istream & operator>>(std::istream & is, pair<T,Q> & p){
    T t;
    Q q;

    is.ignore(0xff,'('); // "  {"
    is >> t >> q;
    is.ignore(0xff,')'); // "  {"

    p.first = t;
    p.second = q;
    return is;
}

#endif // AUX_H_INCLUDED
```

src/auxiliar.cpp

```cpp
#include "auxiliar.h"

void trimIS(std::istream & is){
    while(is.peek() == ' ')
        is.ignore();
}
```

2. Demostraciones

2.1. Demostración del problema 'elVagonetaL'

$P_c : vago == 0 \land n == |empYdes| \land i == 1 \land empYdes == [(e, descansoMasLargo(this, e)) \mid e \leftarrow empleados(this))]$
$Q_c : (\forall j \leftarrow [0..|empYdes|))\ snd(empYdes_j) \leq snd(empYdes_{vago}) \land n == |empYdes|$
$I : 0 \leq i \leq n \land n == |empYdes| \land (\forall j \leftarrow [0..i))\ snd(empYdes_j) \leq snd(empYdes_{vago})$
$B : i < n$
$cota : n - 1$
$fv : i$

$P_c \land B \longrightarrow I$:
Por P_c sabemos que $i == 1$ y por B sabemos que $i < n$, lo que implica $0 \leq i \leq n$.
También, dado que $i == 1$, lo que nos queda que

$$(\forall j \leftarrow [0..i))\ snd(empYdes_j) \leq snd(empYdes_{vago})$$

es lo mismo que

$$(\forall j \leftarrow [0])\ snd(empYdes_j) \leq snd(empYdes_{vago})$$

que es lo mismo que preguntar

$$snd(empYdes_0) \leq snd(empYdes_{vago})$$

y como sabemos por P_c que $vago == 0$, nos queda

$$snd(empYdes_0) \leq snd(empYdes_0)$$

que es una tautología. Entonces puedo decir que $P_c \land B \Rightarrow I$.

$I \land \neg B \longrightarrow Q_c$:
Por $\neg B$ sabemos que $i \geq n$ y por I sabemos que $i \leq n$, entonces $i == n$.
También sabemos por I que $n == |empYdes|$, lo que nos queda que $i == |empYdes|$.
Por I sabemos que

$$(\forall j \leftarrow [0..i))\ snd(empYdes_j) \leq snd(empYdes_{vago})$$

y por lo dicho recién sobre i, podemos decir que

$$(\forall j \leftarrow [0..|empYdes|))\ snd(empYdes_j) \leq snd(empYdes_{vago})$$

y entonces nos queda que $I \land \neg B \longrightarrow Q_c$

$I \land cota < fv \longrightarrow \neg B$:
Esto es lo mismo que decir que si se cumple el invariante y $n - 1 < i$ entonces se niega la guarda.
Sabemos que $n - 1 < i$ y por el invariante $i \in [0..n]$, por lo tanto como i es entero, $i == n$.
Entonces, como $i == n$ y $n \not< n$, la guarda no se cumple.

2.2. Demostración del problema 'candidatosAEmpleadosDelMesL'

$P_c : emp == empleadoConMasVentas(this) \land n == |emp| \land i == 0 \land res == []$
$\quad \land maxCombos = |combosVendidosPorElEmpleado(this, empleadosConMasCombos(this, emp)_0)|$
$Q_c : i == n \land res == [e \mid e \leftarrow [0..|emp|), |combosVendidosPorElEmpleado(this, e)| == maxCombos]$
$I : 0 \leq i \leq n \land res == [e \mid e \leftarrow [0..i), |combosVendidosPorElEmpleado(this, e)| == maxCombos]$
$B : i < n$
$cota : n - 1$
$fv : i$

$P_c \land B \longrightarrow I$:
Por P_c sabemos que $i == 0$ por lo tanto vale que $0 \leq i \leq n$ pues $0 \leq n$ por ser n la longitud de una lista.
También, dado que $i == 0$, $[0..i)$ es lo mismo que $[]$, entonces

$$[\, e \mid e \leftarrow [0..i), |combosVendidosPorElEmpleado(this, e)| == maxCombos\,] == []$$

que es igual a res por P_c. Luego, puedo decir que $P_c \land B \Rightarrow I$ $\therefore$.

$I \land \neg B \longrightarrow Q_c$:

Por $\neg B$ sabemos que $i \geq n$ y por I sabemos que $i \leq n$, entonces $i == n$.

De I sabemos que

$$res == [e \mid e \leftarrow [0..i), |combosVendidosPorElEmpleado(this, e)| == maxCombos]$$

y por lo dicho recién sobre i, como $n == |emp|$ podemos decir que

$$res == [e \mid e \leftarrow [0..|emp|), |combosVendidosPorElEmpleado(this, e)| == maxCombos]$$

y entonces nos queda que $I \wedge \neg B \longrightarrow Q_c$

$I \wedge cota < fv \longrightarrow \neg B$:

Por I sabemos que $i \in [0..n]$ y si $cota < fv$, esto quiere decir que $n - 1 < i$, que es lo mismo que $n \leq i$ pues n e i son numeros naturales.

Entonces si $i \leq n$ y $n \leq i$, resulta $i == n$ con lo cual negamos la B pues este decía que $i < n$.

2.3. Demostración del problema 'agregarComboAlPedidoL'

$P_c : i == 0 \wedge j == 0 \wedge p == ventas(this)_0 \wedge m == |ventas(this)| \wedge n == pre(n) \wedge c == pre(c)$
$Q_c : i == m \wedge p == ventas(this)_j \wedge numero(ventas(this))_j == n$
$I : 0 \leq i \leq m \wedge ((p == ventas(this)_j \wedge numero(ventas(this))_j == n \wedge (\exists v \leftarrow ventas(this))numero(v) == n)$
$\vee (\forall k \leftarrow [0..i))numero(ventas(this)_k) \neq n)$
$B : i < m$
$cota : m - 1$
$fv : i$

$P_c \wedge B \longrightarrow I$:

Por P_c sabemos que $i == 0$ y por B sabemos que $i < m$, lo que implica $0 \leq i \leq m$.

Ahora, supongamos que con $i == 0$ se cumple que $numero(ventas(this)_i) == n$, ($p == ventas(this)_j$ es verdad por P_c que nos dice que $j == 0$ y que $p == ventas(this)_0$. También es verdad que $numero(ventas(this))_j == n$ porque habíamos supuesto que $numero(ventas(this)_i) == n$ con $i == 0$ y sabemos por P_c que $j == 0$, y $(\exists v \leftarrow ventas(this))numero(v) == n$ es trivial porque en la anterior afirmación aseguramos que existía con 0; lo que nos basta para asegurar la veracidad del OR.

Ahora supongamos que no se cumple con $i == 0$, entonces es necesario que $(\forall k \leftarrow [0..i))numero(ventas(this)_k) \neq n)$ sea verdad; como $i == 0$ por P_c, es lo mismo que preguntar si $numero(ventas(this)_0) \neq n$, que es verdad por que lo habíamos supuesto, lo que nos permite asegurar la veracidad del OR.

Entonces, como el OR se cumple en ambos casos y $0 \leq i \leq m$, podemos decir que $P_c \wedge B \longrightarrow I$.

$I \wedge \neg B \longrightarrow Q_c$:

Por $\neg B$ sabemos que $i \geq m$ y por I sabemos que $i \leq m$, entonces $i == m$.

Considerando lo anterior nos queda que $(\forall k \leftarrow [0..i))numero(ventas(this)_k) \neq n)$ es lo mismo que negar que existe alguno en toda la secuencia que cumpla $numero(ventas(this)_alguno) == n$, pero eso es imposible dado que es uno de los requiere del problema. Entonces no queda otra que $((p == ventas(this)_j \wedge numero(ventas(this))_j == n \wedge (\exists v \leftarrow ventas(this))numero(v) == n)$ sea verdad, en particular, nos importa que $((p == ventas(this)_j \wedge numero(ventas(this))_j == n$ sea verdad.

Entonces nos queda que $I \wedge \neg B \longrightarrow Q_c$.

$I \wedge cota < fv \longrightarrow \neg B$:

Por I sabemos que $i \in [0..m]$ y si $cota < fv$, esto quiere decir que $m - 1 < i$, que es lo mismo que $m \leq i$ pues m e i son numeros naturales.

Entonces si $i \leq m$ y $m \leq i$, resulta $i == m$ con lo cual negamos la B pues este decía que $i < m$.