

1. Código

src/fatFood.cpp

```
1 #include <vector>
2 #include <iostream>
3 #include <utility>
4 #include "interfaz.h"
5 #include "local.h"
6
7 using namespace std;
8
9 int main(){
10     Local l(
11         vector<pair<Bebida,Cantidad>> {{PestiCola,5}},
12         vector<pair<Hamburguesa,Cantidad>> {{McGyver,5}},
13         vector<Empleado> {"Hello"}
14     );
15
16     //l.guardar(cout);
17     MenuPrincipal();
18     return 0;
19 }
```

src/tipos.h

```
1 #ifndef TIPOS_H
2 #define TIPOS_H
3
4 #include <vector>
5 #include <stdlib.h>
6
7 #include <string>
8 #include <iostream>
9
10 using namespace std;
11 typedef string Empleado;
12 typedef int Energia;
13 typedef int Cantidad;
14
15 static const char* BEBIDA_STR[] =
16     {"PestiCola","FalsaNaranja","SeVeNada", "AguaConGags","AguaSinGags"};
17 static const char* HAMBURGUESA_STR[] =
18     {"McGyver","CukiQueFresco","McPato","BigMacabra"};
19
20 enum Bebida{PestiCola, FalsaNaranja, SeVeNada, AguaConGags, AguaSinGags};
21 enum Hamburguesa{McGyver, CukiQueFresco, McPato, BigMacabra};
22
23 #define MaxH 3
24 #define MaxB 4
25
26 #endif /*TIPOS_H*/
```

src/combo.h

```
1 #ifndef COMBO_H_INCLUDED
2 #define COMBO_H_INCLUDED
3
4 #include <vector>
5 #include "tipos.h"
6 #include "auxiliar.h"
7
8 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 inválido");
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;

```

```

88         break;
89     }
90 }
91 return is;
92 }
93
94 std::istream & operator>>(std::istream & is, Bebida & b){
95     string s;
96     is >> s;
97     for(int i=0; i<=MaxB; i++){
98         if(BEBIDA_STR[i] == s){
99             b = (Bebida)i;
100             break;
101         }
102     }
103     return is;
104 }

```

src/pedido.h

```

1  #ifndef PEDIDO_H_INCLUDED
2  #define PEDIDO_H_INCLUDED
3
4  #include <vector>
5  #include <stdexcept>
6  #include "tipos.h"
7  #include "combo.h"
8  #include "auxiliar.h"
9
10 class Pedido {
11
12     public:
13
14         Pedido();
15         Pedido(const int n, const Empleado e, const vector<Combo> cs);
16
17         int          numeroP() const;
18         Empleado     atendioP() const;
19         vector<Combo> combosP() const;
20
21         Energia dificultadP() const;
22
23         void agregarComboP(const Combo c);
24         void anularComboP(int i);
25         void cambiarBebidaComboP(const Bebida b, int i);
26         void elMezcladitoP();
27
28         void mostrar(std::ostream& os) const;
29         void guardar(std::ostream& os) const;
30         void cargar (std::istream& is);
31
32         bool operator==(const Pedido& otroPedido) const;
33
34     private:
35
36         vector<Combo> _combos;
37         Empleado      _atendio;
38         int           _numero;
39
40         enum {ENCABEZADO_ARCHIVO = 'P'};
41
42 };
43

```

```

44 // Auxiliares
45 int countBebidasP(const Pedido&, Bebida);
46 int countSandwichesP(const Pedido&, Hamburguesa);
47
48 // Definirlo usando mostrar, para poder usar << con este tipo.
49 std::ostream & operator<<(std::ostream & os, const Pedido & p);
50 std::istream & operator>>(std::istream & is, Pedido & p);
51
52 #endif // PEDIDO_H_INCLUDED

```

src/pedido.cpp

```

1 #include "pedido.h"
2 #include <algorithm>
3 vector<Combo> combosDelPedidoSinRep (const Pedido *p);
4 vector<Combo> combosRepetidos (const Pedido *p);
5 vector<Bebida> bebidasDelPedidoSinRep (const Pedido *p);
6 vector<Hamburguesa> sandwichesDelPedidoSinRep (const Pedido *p);
7 bool estaC(const Combo c, const vector<Combo> cs);
8 void cambiarBebida (Combo &c, const vector<Bebida> bs);
9 void cambiarSandwich (Combo &c, const vector<Hamburguesa> hs);
10
11 Pedido::Pedido(){
12 }
13
14 Pedido::Pedido(const int nro, const Empleado e, const vector<Combo> combos){
15     _numero = nro;
16     _atendio = e;
17     _combos = combos;
18 }
19
20 int Pedido::numeroP() const{
21     return _numero;
22 }
23
24 Empleado Pedido::atendioP() const{
25     return _atendio;
26 }
27
28 vector<Combo> Pedido::combosP() const{
29     return _combos;
30 }
31
32 Energia Pedido::dificultadP() const{
33     int n=combosP().size();
34     int sum=0;
35     int i=0;
36     while(i<n){
37         sum=sum+combosP()[i].dificultadC();
38         i++;
39     }
40     return sum;
41 }
42
43 void Pedido::agregarComboP(const Combo c){
44     _combos.push_back(c);
45 }
46
47 void Pedido::anularComboP(int i){
48     _combos.erase(_combos.begin()+i);
49 }
50
51 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:
104             "<<combosP()[i].bebidaC()<<" Dificultad: "<<combosP()[i].dificultadC()<<endl;
105     }
106 }
107
108 void Pedido::guardar(std::ostream& os) const{
109     os << '{' << " "
110         << (char)ENCABEZADO_ARCHIVO << " "
111         << _numero << " "
112         << _atendio << " [ ";
113     for (auto &c : _combos){
114         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 inválido");
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

```

```

176 bool estaS(const Hamburguesa h, const vector<Hamburguesa> hs){
177     int i = 0;
178     int n = hs.size();
179     while (i < n && hs[i] != h){
180         i = i + 1;
181     }
182     return i < n;
183 }
184
185 vector<Bebida> bebidasDelPedido (const Pedido *p){
186     vector<Bebida> res;
187     res.reserve(p->combosP().size());
188     for (auto &i : p->combosP()){
189         res.push_back(i.bebidaC());
190     }
191     return res;
192 }
193
194 vector<Bebida> bebidasDelPedidoSinRep (const Pedido *p){
195     vector<Bebida> res;
196     res.reserve(bebidasDelPedido(p).size());
197     for (auto &i : bebidasDelPedido(p)){
198         if (!estaB(i,res)){
199             res.push_back(i);
200         }
201     }
202     return res;
203 }
204
205 vector<Hamburguesa> sandwichesDelPedido (const Pedido *p){
206     vector<Hamburguesa> res;
207     res.reserve(p->combosP().size());
208     for (auto &i : p->combosP()){
209         res.push_back(i.sandwichC());
210     }
211     return res;
212 }
213
214 vector<Hamburguesa> sandwichesDelPedidoSinRep (const Pedido *p){
215     vector<Hamburguesa> res;
216     res.reserve(sandwichesDelPedido(p).size());
217     for (auto &i : sandwichesDelPedido(p)){
218         if (!estaS(i,res)){
219             res.push_back(i);
220         }
221     }
222     return res;
223 }
224
225 vector<Combo> combosDelPedidoSinRep (const Pedido *p){
226     vector<Combo> res;
227     res.reserve(p->combosP().size());
228     for (auto &i : p->combosP()){
229         if (!estaC(i,res)){
230             res.push_back(i);
231         }
232     }
233     return res;
234 }
235
236 vector<Combo> combosRepetidos (const Pedido *p){
237     vector<Combo> res;
238     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{

```

```

11     public:
12
13         Local();
14         Local(const vector< pair <Bebida,Cantidad> > bs, const vector< pair
15             <Hamburguesa,Cantidad> > hs, const vector<Empleado> es);
16
17         Cantidad          stockBebidasL(const Bebida b) const;
18         Cantidad          stockSandwichesL(const Hamburguesa h) const;
19         vector<Bebida>    bebidasDelLocalL() const;
20         vector<Hamburguesa> sandwichesDelLocalL() const;
21         vector<Empleado>  empleadosL() const;
22         vector<Empleado>  desempleadosL() const;
23         Energia           energiaEmpleadoL(const Empleado e) const;
24         vector<Pedido>    ventasL() const;
25         bool              unaVentaCadaUnoL() const;
26         void              venderL(const Pedido p);
27         vector<Empleado>  candidatosAEmpleadosDelMesL() const;
28         void              sancionL(const Empleado e, const Energia n);
29         Empleado          elVagonetaL() const;
30         void              anularPedidoL(int n);
31         void              agregarComboAlPedidoL(const Combo c, int n);
32
33         void mostrar(std::ostream& os) const;
34         void guardar(std::ostream& os) const;
35         void cargar (std::istream& is);
36
37     private:
38         vector< pair <Hamburguesa,Cantidad> > _sandwiches;
39         vector< pair <Bebida,Cantidad> >      _bebidas;
40         vector< pair <Empleado, Energia> >     _empleados;
41         vector< Pedido >                      _ventas;
42
43
44
45         enum {ENCABEZADO_ARCHIVO = 'L'};
46
47 };
48
49 std::ostream & operator<<(std::ostream & os,const Local & c);
50 std::istream & operator>>(std::istream & is, Local & l);
51
52 #endif // LOCAL_H_INCLUDED

```

src/local.cpp

```

1  #include "local.h"
2  #include <string.h>
3  #include <algorithm>
4
5  vector<Pedido> pedidosDelEmpleado();
6  int maxDescansoEmpleado(Empleado);
7  vector< pair<Empleado, int> > empleadoYdescanso(const Local *l);
8  vector< pair <Empleado, Energia> > empleadosYenergiaL(const Local *l);
9  vector<Empleado> empleadosConMasVentas(const Local *l);
10 int maxCantPedidos(const Local *l);
11 int maxCantCombos(const Local *l);
12 vector<Combo> combosDelEmpleado(const Local *l,Empleado e);
13
14 Local::Local() {
15 }
16
17 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;

```

```

81     res.reserve(_empleados.size());
82     for (auto &i : _empleados){
83         if(i.second>=0){
84             res.push_back(i.first);
85         }
86     }
87     return res;
88 }
89
90
91 vector<Empleado>      Local::desempleadosL() const{
92     vector<Empleado> res;
93     res.reserve(_empleados.size());
94     for (auto &i : _empleados){
95         if(i.second<0){
96             res.push_back(i.first);
97         }
98     }
99     return res;
100 }
101
102 Energia Local::energiaEmpleadoL(const Empleado e) const{
103     for (auto &i : _empleados)
104         if (i.first == e)
105             return i.second;
106 }
107
108 vector<Pedido>      Local::ventasL() const{
109     return _ventas;
110 }
111
112 vector<Empleado> Local::candidatosAEmpleadosDelMesL() const{
113     vector<Empleado> res;
114     vector<Empleado> emp = empleadosConMasVentas(this);
115     int n = emp.size();
116     int i = 0;
117     int maxCombos = maxCantCombos(this);
118     while (i<n){
119         if (combosDelEmpleado(this,emp[i]).size() == maxCombos){
120             res.push_back(emp[i]);
121             i++;
122         }
123         else{
124             i++;
125         }
126     }
127     return res;
128 }
129
130 void Local::venderL(const Pedido p){
131     //duda: si no se cumple los requiere debo decir que no se cumplieron o
132     //simplemente se supone que se cumplen?
133     bool exito=true;
134     for (int i=0; i<_empleados.size();i++)
135     {
136         if (_empleados[i].first==p.atendioP())
137         {
138             _empleados[i].second= _empleados[i].second-p.dificultadP();
139             if (_empleados[i].second<0)
140             {
141                 exito=false;
142             }
143         }
144     }

```

```

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() <
249         p2.numeroP();});
250     int i=0;
251     while(i<v.size()) {
252         bool pertenece=false;
253         for (int j=0;j<_empleados.size();j++) {
254             if ((v[i].atendioP()==_empleados[j].first)&& (_empleados[j].second>=0)) {
255                 pertenece=true;
256             }
257         }
258         if (pertenece==false) {
259             v.erase(v.begin()+i);
260             i--;
261         }
262         i++;
263     }
264     if (v.size()<=1)
265     {
266         return true;
267     }
268     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 Empleado Local::elVagonetaL() const{
288     vector< pair<Empleado,int> > empYdes = empleadoYdescanso(this);
289     int i = 1, n = empYdes.size();
290     int vago = 0;
291     while (i < n){
292         if (empYdes[i].second > empYdes[vago].second)
293             vago = i;
294         i++;
295     }
296     return empYdes[vago].first;
297 }
298
299 void Local::guardar(std::ostream& os) const{
300     os << "{ " << (char)ENCABEZADO_ARCHIVO << " "
301         << _bebidas << " "
302         << _sandwiches << " "
303         << _empleados << " [ ";
304     for (auto &e : _ventas){
305         e.guardar(os);
306         os << " ";
307     }
308     os << "] }";
309 }
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 energÍa: "<<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 inválido");
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() -
370         l->ventasL().back().numeroP(), maxDescanso);
371     return maxDescanso;
372 }
373
374 vector< pair<Empleado, int> > empleadoYdescanso(const Local *l)
375 {
376     vector< pair<Empleado, int> > res;
377     vector<Empleado> emp = l->empleadosL();
378     res.reserve(emp.size());
379     for(auto &i : emp)
380         res.push_back(pair<Empleado, int>(i, maxDescansoEmpleado(l, i)));
381     return res;
382 }
383
384 vector< pair <Empleado, Energia> > empleadosYenergiaL(const Local *l){
385     vector< pair <Empleado, Energia> > res;
386     vector<Empleado> emp = l->empleadosL();
387     res.reserve(emp.size());
388     for (auto &i : emp)
389         res.push_back(pair<Empleado, Energia>(i, l->energiaEmpleadoL(i)));
390     return res;
391 }
392
393 vector<Empleado> empleadosConMasVentas(const Local *l){
394     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 }

```

```

1  #ifndef AUX_H_INCLUDED
2  #define AUX_H_INCLUDED
3
4  #include <string>
5  #include <vector>
6  #include <istream>
7  #include "tipos.h"
8
9  // Stream functions
10
11 void trimIS(std::istream & is);
12
13 // Vector and pair I/O
14
15 template<typename T, typename A>
16 std::ostream & operator<<(std::ostream & os, const vector<T,A> & v){
17     os << "[";
18     for(auto &i : v){
19         os << i << " ";
20     }
21     os << "]";
22     return os;
23 }
24
25 template<typename T, typename Q>
26 std::ostream & operator<<(std::ostream & os, const pair<T,Q> & p){
27     os << "(" << p.first << " " << p.second << ")";
28     return os;
29 }
30
31 template<typename T>
32 std::istream & operator>>(std::istream & is, vector<T> & v){
33     T t;
34
35     is.ignore(0xff, '['); // " {"
36     trimIS(is);
37     while(is.peek() != ' '){
38         is >> t;
39         v.push_back(t);
40         trimIS(is);
41     }
42     is.ignore(0xff, ']'); // " {"
43     return is;
44 }
45
46 template<typename T, typename Q>
47 std::istream & operator>>(std::istream & is, pair<T,Q> & p){
48     T t;
49     Q q;
50
51     is.ignore(0xff, '('); // " {"
52     is >> t >> q;
53     is.ignore(0xff, ')'); // " {"
54
55     p.first = t;
56     p.second = q;
57     return is;
58 }
59
60 #endif // AUX_H_INCLUDED

```

```
1  #include "auxiliar.h"
2
3  void trimIS(std::istream & is){
4      while(is.peek() == ' ')
5          is.ignore();
6  }
```

2. Demostraciones

2.1. Demostración del problema 'elVagonetaL'

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

$P_c \wedge B \longrightarrow I$:

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

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

es lo mismo que

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

que es lo mismo que preguntar

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

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

$$\text{snd}(empYdes_0) \leq \text{snd}(empYdes_0)$$

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

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

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

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

y por lo dicho recién sobre i , podemos decir que

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

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

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

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'

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

$P_c \wedge B \longrightarrow I$:

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 $P_c \wedge B \Rightarrow I$.

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

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'

$P_c : i == 0 \wedge j == 0 \wedge p == ventasL(this)_0 \wedge m == |ventasL(this)| \wedge n == pre(n) \wedge c == pre(c)$

$Q_c : i == m \wedge p == ventasL(this)_j \wedge numeroP(ventasL(this))_j == n$

$I : 0 \leq i \leq m \wedge ((p == ventasL(this)_j \wedge numeroP(ventasL(this))_j == n \wedge (\exists v \leftarrow ventas(this)) numeroP(v) == n) \vee (\forall k \leftarrow [0..i)) numeroP(ventasL(this)_k) \neq n)$

$B : i < m$

$cota : m - 1$

$fv : i$

$P_c \wedge B \longrightarrow I$:

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 $numeroP(ventasL(this)_i) == n$, $(p == ventasL(this)_j)$ es verdad por P_c que nos dice que $j == 0$ y que $p == ventasL(this)_0$. También es verdad que $numeroP(ventasL(this))_j == n$ porque habíamos supuesto que $numeroP(ventasL(this)_i) == n$ con $i == 0$ y sabemos por P_c que $j == 0$, y $(\exists v \leftarrow ventas(this)) numeroP(v) == n$ 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 $(\forall k \leftarrow [0..i)) numeroP(ventasL(this)_k) \neq n$ sea verdad; como $i == 0$ por P_c , es lo mismo que preguntar si $numeroP(ventasL(this)_0) \neq n$, 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 $(\forall k \leftarrow [0..i)) numeroP(ventasL(this)_k) \neq n$ es lo mismo que negar que existe alguno en toda la secuencia que cumpla $numeroP(ventasL(this)_alguno) == n$, pero eso es imposible dado que es uno de los requiere del problema. Entonces no queda otra que $((p == ventasL(this)_j \wedge numeroP(ventasL(this))_j == n \wedge (\exists v \leftarrow ventas(this)) numeroP(v) == n)$ sea verdad, en particular, nos importa que $((p == ventasL(this)_j \wedge numeroP(ventasL(this))_j == n$ 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 $i < m$.