LCOV - code coverage report

Current view:	top level - src - ejercicios.cpp (source / functions)		Hit	Total	Coverage
Test:	coverage.info	Lines:	346	347	99.7 %
Date:	2018-11-17 01:01:53	Functions:	52	52	100.0 %

```
   Line data    Source code

 1            : #include "ejercicios.h"
 2            :
 3            : // Auxiliares - Ord(item)
 4            :
 5            : // Individuos
 6            : int codusuI = int(ItemInd::CODUSU);
 7            : int ano4I = int(ItemInd::ANO4);
 8            : int trimestreI = int(ItemInd::TRIMESTRE);
 9            : int componente = int(ItemInd::COMPONENTE);
10            : int nivel_ed = int(ItemInd::NIVEL_ED);
11            : int estado = int(ItemInd::ESTADO);
12            : int cat_ocup = int(ItemInd::CAT_OCUP);
13            : int edad = int(ItemInd::EDAD);
14            : int genero = int(ItemInd::GENERO);
15            : int pp3e_tot = int(ItemInd::PP3E_TOT);
16            : int pp04_cod = int(ItemInd::PP04D_COD);
17            : int p21 = int(ItemInd::P21);
18            : // Hogares
19            : int itf = int(ItemHogar::ITF);
20            : int ii7 = int(ItemHogar::PROP);
21            : int codusuH = int(ItemHogar::CODUSU_HOG);
22            : int trimestreH = int(ItemHogar::TRIMESTRE_HOG);
23            : int ano4H = int(ItemHogar::ANO4_HOG);
24            :
25            : /* EJERCICIO 1 */
26            :
27            : /* Funciones auxiliares */
28         25 : bool esMatrizIndividuo(eph_i ti){
29            :     //(∀ i, j : Z)(0 ≤ i < |t| ∧ 0 ≤ j < |t|) −→ L |t[i]| = |t[j| ∧ |t[i]| = cantidadItemsIndividuo()
30         25 :     bool res = true;
31      25206 :     for (int i = 0; i < ti.size(); ++i) {
32      25181 :         if (ti[i].size() != 12){
33          8 :             res = false;
34            :         }
35            :     }
36         25 :     return res;
37            : }
38            :
39         17 : bool esMatrizHogar(eph_h th){
40            :     //(∀ i, j : Z)(0 ≤ i < |t| ∧ 0 ≤ j < |t|) −→ L |t[i]| = |t[j| ∧ |t[i]| = cantidadItemsHogar()
41         17 :     bool res = true;
42       8635 :     for (int i = 0; i < th.size(); ++i) {
43       8618 :         if (th[i].size() != 5){
44          1 :             res = false;
45            :         }
46            :     }
47         17 :     return res;
48            : }
49            :
50      25103 : bool individuoValido(individuo ind){
51      25103 :     vector<int> p = ind;
52      25103 :     bool res = true;
53            :     // enum class ItemInd {CODUSU,ANO4, COMPONENTE,NIVEL_ED,TRIMESTRE,CH4,CH6,ESTADO,CAT_OCUP,PP3E_TOT,PP04D_COD,P21};
54            :     // Se revisan predicados de especificacion, en caso de no cumplir uno, la variable res se modifica a false.
55            :     // Codosu > 0, Componente > 0, Ano > 0
56      25103 :     if (p[codusuI] <= 0 || p[ano4I] <= 0 || p[componente] <= 0){
57          1 :         res = false;
```

```
    :     }
    :     // Trimestre en rango
25103 :     if (1 > p[trimestreI] || p[trimestreI] > 4){
    1 :         res = false;
    :     }
    :     // Nivel_Ed en rango
25103 :     if (-1 > p[nivel_ed] || p[nivel_ed] > 1){
    1 :         res = false;
    :     }
    :     // Estado en rango
25103 :     if ((p[estado] < 0 && p[estado] != -1) || p[estado] > 1){
  999 :         res = false;
    :     }
    :     // catOcup EnRango
25103 :     if (-1 > p[cat_ocup] || p[cat_ocup] > 4){
    2 :         res = false;
    :     }
    :     // Edad en rango
25103 :     if (-1 > p[edad] || p[edad] > 110){
    1 :         res = false;
    :     }
    :     // PP3E_Tot ^ PP04D_COD>= -1
25103 :     if (p[pp3e_tot] < -1 || p[pp04_cod] < -1){
    1 :         res = false;
    :     }
    :     // P21 En rango
25103 :     if (-1 > p[p21]){
  217 :         res = false;
    :     }
    :     // ¬trabaja(p) ⟶ p[@P 21] <= 0
25103 :     if (p[estado] != 1 && p[p21] > 0){
    1 :         res = false;
    :     }
    :     // ¬trabaja(p) -> p[@P P 3E T ot] <= 0
25103 :     if (!(p[estado] == 1 || p[pp3e_tot] <= 0)){
    1 :          res = false;
    :      }
    :     // trabaja[p] -> p[@ch6] > 10
25103 :     if (p[estado] == 1 && p[edad] < 10) {
    1 :          res = false;
    :     }
    :
50206 :     return res;
    : }
    :
   12 : bool individuosValidos(eph_i ti){
   12 :     bool res = true;
25115 :     for (int i = 0; i < ti.size(); ++i) {
25103 :         if (!individuoValido(ti[i])) {
 1221 :             res = false;
    :         }
    :     }
   12 :     return res;
    : }
    :
 8610 : bool hogarValido(hogar hog){
 8610 :     vector<int> h = hog;
 8610 :     bool res = true;
    :     // enum class ItemHogar {CODUSU,ANO4,TRIMESTRE,ITF,II7};
    :     // p[@Codusu] > 0 ∧ p[@Ano4] > 0 ^ p[@Itf ] ≥ 0
 8610 :     if (h[codusuH] <= 0 || h[ano4H] <= 0 || h[itf] < 0) {
    2 :         res = false;
    :     }
    :     // ∧ trimestreEnRango(p[@trim])
 8610 :     if (1 > h[trimestreH] || h[trimestreH] > 4) {
    2 :         res = false;
    :     }
    :     // pII7EnRango(p[@P ii7])
```

```
126       8610 :     if ((h[ii7] < 1 && h[ii7] != -1) || h[ii7] > 3) {
127          2 :         res = false;
128            :     }
129      17220 :     return res;
130            : }
131            :
132         15 : bool hogaresValidos(eph_h th){
133         15 :     bool res = true;
134       8625 :     for (int h = 0; h < th.size(); ++h) {
135       8610 :         if (!hogarValido(th[h])){
136          2 :             res = false;
137            :         }
138            :     }
139         15 :     return res;
140            : }
141            :
142         16 : bool individuosDistintos(eph_i ti){
143            :     // (∀ p1, p2 : individuo) p1 ∈ ti∧p2 ∈ ti ∧ p1 6 = p2 −→ L p1[@Codusu] 6 = p2[@Codusu] ∨ p1[@Componente] 6 = p2[@Componente]
144         16 :     bool res = true;
145      25143 :     for (int i = 0; i < ti.size(); ++i) {
146  115321350 :         for (int j = i+1; j < ti.size() ; ++j) {
147  115296223 :             if (ti[i][codusuI] == ti[j][codusuI]) {
148      36655 :                 if (ti[i][componente] == ti[j][componente]) {
149          1 :                     res = false;
150            :                 }
151            :             }
152            :         }
153            :     }
154         16 :     return res;
155            : }
156            :
157         13 : bool hogaresUnicos (eph_h th){
158            :     // (∀ p1, p2 : hogar) p1 ∈ th ∧ p2 ∈ th ∧ p1 6 = p2 −→ L p1[@Codusu] 6 = p2[@Codusu]
159         13 :     bool res = true;
160       8615 :     for (int i = 0; i < th.size(); ++i) {
161   12594843 :         for (int j = i+1; j < th.size(); ++j) {
162   12586241 :             if (th[i][codusuH] == th[j][codusuH]){
163          1 :                 res = false;
164            :             }
165            :         }
166            :     }
167         13 :     return res;
168            : }
169            :
170          8 : bool mismoPeriodo (eph_i ti, eph_h th){
171          8 :     bool res = true;
172      23119 :     for (int i = 0; i < ti.size(); ++i) {
173   74476829 :         for (int h = 0; h < th.size(); ++h) {
174   74453718 :             if (ti[i][ano4I] != th[h][ano4H] || ti[i][trimestreI] != th[h][trimestreH]){
175          4 :                 res = false;
176            :             }
177            :         }
178            :     }
179          8 :     return res;
180            : }
181            :
182      23105 : bool individuoConHogar(individuo i, eph_h th){
183      23105 :     bool res = false;
184   74476799 :     for (int h = 0; h < th.size(); ++h) {
185   74453694 :         if (i[codusuI] == th[h][codusuH]){
186      23104 :             res = true;
187            :         }
188            :     }
189      23105 :     return res;
190            : }
191            :
192          7 : bool individuosConHogar(eph_i ti, eph_h th){
193          7 :     bool res = true;
```

```
194       23112 :     for (int i = 0; i < ti.size(); ++i) {
195       23105 :         if(!individuoConHogar(ti[i], th)){
196           1 :             res = false;
197             :         }
198             :     }
199           7 :     return res;
200             : }
201             :
202        7744 : bool hogarConIndividuo(hogar h, eph_i ti){
203        7744 :     bool res = false;
204    35452255 :     for (int i = 0; i < ti.size(); ++i) {
205    35452254 :         if (h[codusuH] == ti[i][codusuI]) {
206        7743 :             res = true;
207        7743 :             break;
208             :         }
209             :     }
210        7744 :     return res;
211             : }
212             :
213           6 : bool hogaresConIndividuos(eph_i ti, eph_h th){
214           6 :     bool res = true;
215        7750 :     for (int h = 0; h < th.size(); ++h) {
216        7744 :         if (!hogarConIndividuo(th[h], ti)) {
217           1 :             res = false;
218             :         }
219             :     }
220           6 :     return res;
221             : }
222             :
223        7739 : int cumIngresoIndividual(eph_i ti, int codusu){
224        7739 :     int res = 0;
225    74461385 :     for (int i = 0; i < ti.size(); ++i) {
226    74453646 :         if (ti[i][codusuI] == codusu){
227       23093 :             if(ti[i][p21] == -1){
228             :                 // Si no se responde en encuesta, no contar el "-1" como no responde, simplemente ignorarlo.
229        2658 :                 continue;
230             :             }
231       20435 :             res += ti[i][p21];
232             :         }
233             :     }
234        7739 :     return res;
235             : }
236             :
237           5 : bool ingresoHogarEnRango(eph_i ti, eph_h th){
238           5 :     bool res = true;
239        7744 :     for (int h = 0; h < th.size(); ++h) {
240        7739 :         if (th[h][itf] > cumIngresoIndividual(ti, th[h][codusuH])) {
241           1 :             res = false;
242             :         }
243             :     }
244           5 :     return res;
245             : }
246             :
247           8 : bool tablasCompatibles(eph_i ti, eph_h th){
248           8 :     return (mismoPeriodo(ti, th) && individuosConHogar(ti, th) && hogaresConIndividuos(ti, th) && ingresoHogarEnRango(ti, th));
249             : }
250             :
251             : // Resolucion Ejercicio
252          25 : bool esEncuestaValida ( eph_i ti , eph_h th ) {
253          25 :     bool res = esMatrizIndividuo(ti) && esMatrizHogar(th) && individuosDistintos(ti) && hogaresValidos(th) && hogaresUnicos(th) && individuosValidos(ti) && tablasCompatibles(ti,
254          25 :     return res;
255             : }
256             :
257             : /* EJERCICIO 2 */
258             :
259             : // Auxiliares
260          21 : bool esElActivoDeMayorIngreso(individuo p, eph_i ti){
261          21 :     bool res = true;
```

```
  262        330 :     for (int i = 0; i < ti.size(); ++i) {
  263        309 :         if (p[p21] < ti[i][p21]){
  264        100 :             res = false;
  265            :         }
  266            :     }
  267         21 :     return res;
  268            : }
  269            : // Se determina dentro de una tabla de invidiuos el individuo con mayor ingreso
  270          2 : individuo activoDeMayorIngreso(eph_i ti){
  271          2 :     individuo result;
  272         21 :     for (int i = 0; i < ti.size(); ++i) {
  273         21 :         if (esElActivoDeMayorIngreso(ti[i], ti)){
  274          2 :             result = ti[i];
  275          2 :             break;
  276            :         }
  277            :     }
  278          2 :     return result;
  279            : }
  280            :
  281            : // Resolucion Ejercicio
  282          2 : hogar mayorIngresoPorPersonaActiva ( eph_i ti , eph_h th ){
  283          2 :     hogar result;
  284          4 :     individuo mayorActivo = activoDeMayorIngreso(ti);
  285            :     // for (int h : th.size()) ranged-based loop
  286          7 :     for (int h = 0; h < th.size(); ++h) {
  287          7 :         if (mayorActivo[codusuI] == th[h][codusuH]){
  288          2 :             result = th[h];
  289          2 :             break;
  290            :         }
  291            :     }
  292          4 :     return result;
  293            : }
  294            :
  295            : /* EJERCICIO 3 */
  296            :
  297            : // Auxiliares
  298        105 : bool esProp(hogar h){
  299        105 :     return h[ii7] == 1;
  300            : }
  301            :
  302        105 : int compHogar(hogar h, eph_i ti){
  303        105 :     int res = 0;
  304      12340 :     for (int i = 0; i < ti.size(); ++i) {
  305      12235 :         if(ti[i][codusuI] == h[codusuH]){
  306        251 :             res += 1;
  307            :         }
  308            :     }
  309        105 :     return res;
  310            : }
  311            :
  312            : // Resolucion de ejercicio
  313          3 : float porcHogaresNoPropMiemMay3 (eph_i ti, eph_h th ){
  314          3 :     float res = 0.0;
  315          3 :     int hogaresNoPropMiemMay3 = 0;
  316        108 :     for (int h = 0; h < th.size(); ++h) {
  317        105 :         if (!esProp(th[h]) && compHogar(th[h], ti) >= 3){
  318         39 :             hogaresNoPropMiemMay3 += 1;
  319            :         }
  320            :     }
  321          3 :     res = float(hogaresNoPropMiemMay3) / float(th.size()) * 100;
  322          3 :     return res;
  323            : }
  324            :
  325            : /* EJERCICIO 4 */
  326            :
  327            : // Auxiliares
  328        220 : bool hogarConUnSoloHabitante(hogar h, eph_h ti){
  329        220 :     int individEnHogar = 0;
```

```
24810 :     for (int j = 0; j < ti.size(); ++j) {
24590 :         if (ti[j][codusuH] == h[codusuH]){
  526 :             individEnHogar += 1;
      :         }
      :     }
  220 :     return individEnHogar == 1;
      : }
   70 : individuo individuoDelHogar(hogar h, eph_h ti){
   70 :     individuo result;
 3252 :     for (int i = 0; i < ti.size(); ++i) {
 3252 :         if (h[codusuH] == ti[i][codusuI]){
   70 :             result = ti[i];
   70 :             break;
      :         }
      :     }
   70 :     return result;
      : }
      :
   70 : bool esUniversitario(individuo i){
   70 :     return i[nivel_ed] == 1;
      : }
      :
   42 : bool esMenorDe45(individuo i){
   42 :     return i[edad] < 45;
      : }
      :
   16 : bool esTrabajadorActivo(individuo i){
   16 :     return i[estado] == 1;
      : }
      :
  220 : bool esPerfilBuscado(hogar h, eph_i ti){
  220 :     bool res = false;
      :     // Primero chequear que hay un solo habitante, y almacenarlo en una variable para analizar los predicados correspondientes
  220 :     if (hogarConUnSoloHabitante(h, ti)){
  140 :         individuo habitante = individuoDelHogar(h, ti);
   70 :         res = esUniversitario(habitante) && esMenorDe45(habitante) && esTrabajadorActivo(habitante);
      :     }
  220 :     return res;
      : }
      :
   16 : bool hogarSoloGeneroG(hogar h, eph_i ti, int g){
   16 :     bool res = true;
 1218 :     for (int i = 0; i < ti.size(); ++i) {
 1208 :         if (ti[i][codusuI] == h[codusuH]) {
   16 :             if (ti[i][genero] != g) {
    6 :                 res = false;
    6 :                 break;
      :             }
      :         }
      :     }
   16 :     return res;
      : }
      :
    4 : int cantidadHogaresXGenero(eph_i ti, eph_h th, int g){
    4 :     int result = 0;
  114 :     for (int h = 0; h < th.size(); ++h) {
  110 :         if (esPerfilBuscado(th[h], ti)){
    8 :             if (hogarSoloGeneroG(th[h], ti, g)){
    5 :                 result += 1;
      :             }
      :         }
      :     }
    4 :     return result;
      : }
      :
    4 : int cantidadHogaresXNoGenero(eph_i ti, eph_h th, int g){
    4 :     int result = 0;
  114 :     for (int h = 0; h < th.size(); ++h) {
```

```
398         110 :         if (esPerfilBuscado(th[h], ti)){
399           8 :             if (!hogarSoloGeneroG(th[h], ti, g)){
400           3 :                 result += 1;
401             :             }
402             :         }
403             :     }
404           4 :     return result;
405             : }
406             :
407             :
408             : // Resolucion de ejercicio
409           4 : bool generoGPosiblesClientes (eph_i ti, eph_h th, int g) {
410           4 :     return cantidadHogaresXGenero(ti, th, g) > cantidadHogaresXNoGenero(ti, th, g);
411             : }
412             :
413             : /* EJERCICIO 5 */
414             : // Algoritmo de ordenamiento utilizado: Selection Sort
415             :
416             : // Auxiliares
417             : // Swap para tabla individuos
418          36 : void swapTI(eph_i & ti, int a, int b){
419          72 :     individuo temp = ti[a];
420          36 :     ti[a] = ti[b];
421          36 :     ti[b] = temp;
422          36 : }
423             :
424             : // retorna el ITF del hogar, segun el individuo
425         250 : int findITF(individuo i, eph_h th){
426         250 :     int itf_i = 0;
427         554 :     for (int h = 0; h < th.size(); ++h) {
428         554 :         if (i[codusuI] == th[h][codusuH]){
429         250 :             itf_i = th[h][itf];
430         250 :             break;
431             :         }
432             :     }
433         250 :     return itf_i;
434             : }
435             :
436             : // Busqueda del hogar con mayorITF en una secuencia. Utilizada para ordenar con ordenarSegunITF
437          18 : int findMaxITFPosition (eph_i ti, eph_h th, int d, int h){
438          18 :     int max = d;
439         112 :     for (int i = d; i < h; ++i) {
440          94 :         if(findITF(ti[max], th) < findITF(ti[i], th)){
441          10 :             max = i;
442             :         }
443             :     }
444          18 :     return max;
445             : }
446             : // Ordenar descendentemente segun ITF (Selection Sort)
447           2 : void ordenarSegunITF (eph_i & ti, eph_h th){
448          20 :     for (int i = 0; i < ti.size(); ++i){
449          18 :         int maxITFPos = findMaxITFPosition(ti, th, i, ti.size());
450          18 :         swapTI(ti, i, maxITFPos);
451             :     }
452           2 : }
453             :
454             : //Ordenar ascendentemente segun CODUSU en caso de empate, PRE: Ya esta ordenada por ITF
455           2 : void desempateSegunCODUSU(eph_i & ti, eph_h th){
456          20 :     for (int i = 0; i < ti.size(); ++i) {
457             :         // Idea: Insertion Sort
458          34 :         for (int j = i; j > 0  && findITF(ti[j], th) == findITF(ti[j-1], th); --j) {
459          16 :             if(ti[j][codusuI] < ti[j-1][codusuI]){
460           2 :                 swapTI(ti, j, j-1);
461             :             }
462             :         }
463             :     }
464           2 : }
465             :
```

```
466           : // Busqueda del menor componente del hogar dentro de un determinado rango, suponiendo que ya esta ordenada por ITF (mismos codusu juntos)
467        16 : int findMinComp (eph_i ti, int d, int h){
468        16 :     int min = d;
469        44 :     for (int i = d; i < h; ++i) {
470        41 :         if(ti[min][codusuH] != ti[i][codusuH]){
471        13 :             break;
472           :         }
473        28 :         if(ti[min][componente] > ti[i][componente]){
474         3 :             min = i;
475           :         }
476           :     }
477        16 :     return min;
478           : }
479           :
480           : //Ordenar ascendentemente segun Componente cada hogar
481         2 : void ordenarSegunComponente(eph_i & ti){
482        18 :     for (int i = 0; i < ti.size()-1; ++i) {
483        16 :         int minCompPos = findMinComp(ti, i, ti.size());
484        16 :         swapTI(ti, i, minCompPos);
485           :     }
486         2 : }
487           :
488           : // Resolucion de ejercicio
489         2 : void ordenarPorITF(eph_i & ti, eph_h th){
490         2 :     ordenarSegunITF(ti, th);
491         2 :     desempateSegunCODUSU(ti, th);
492         2 :     ordenarSegunComponente(ti);
493         2 : }
494           :
495           : /* EJERCICIO 6 */
496           :
497           : // Auxiliares
498       110 : bool esMaxNivelEducativo(int ne, hogar h, eph_i ti){
499       110 :     int max_nivel_ed = 0;
500     12820 :     for (int i = 0; i < ti.size(); ++i) {
501     12710 :         if (h[codusuH] == ti[i][codusuH]){
502       274 :             if(ti[i][nivel_ed] > max_nivel_ed){
503        62 :                 max_nivel_ed = ti[i][nivel_ed];
504           :             }
505           :         }
506           :     }
507       110 :     return max_nivel_ed == ne;
508           : }
509           :
510         4 : int cantHogaresNivelEducativo(int ne, eph_h th, eph_i ti){
511         4 :     int result = 0;
512       114 :     for (int h = 0; h < th.size(); ++h) {
513       110 :         if (esMaxNivelEducativo(ne, th[h], ti)){
514        55 :             result += 1;
515           :         }
516           :     }
517         4 :     return result;
518           : }
519           :
520           : // Resolucion de ejercicio
521         2 : lista_nivel_ed nivelEducativoXHogar(eph_i ti, eph_h th){
522         2 :     lista_nivel_ed result;
523         6 :     for (int ne = 0; ne < 2; ++ne) {
524         4 :         nivel_ed_hogar chunk = make_pair(ne, 100 * float(cantHogaresNivelEducativo(ne, th, ti)) / th.size());
525         4 :         result.push_back(chunk);
526           :     }
527         2 :     return result;
528           : }
529           :
530           : /* EJERCICIO 7 */
531           :
532           : // Auxiliares
533         2 : bool sonDistintosTrimestres(eph_i ti1, eph_i ti2, eph_h th1, eph_h th2){
```

```
534          2 :     bool res = true;
535          2 :     if(ti1[0][trimestreI] == ti2[0][trimestreI]){
536          2 :         res = false;
537            :     }
538          2 :     if(th1[0][trimestreH] == th2[0][trimestreH]){
539          2 :         res = false;
540            :     }
541          2 :     return res;
542            : }
543            :
544          4 : eph_h hogaresOrdenadosPorITFyCODUSU(eph_h t){
545          4 :     eph_h th = t;
546            :     // Ordenar por ITF
547       1655 :     for (int h = 0; h < th.size(); ++h) {
548       1651 :         int max = h;
549     677511 :         for (int i = h; i < th.size(); ++i) {
550     675860 :             if(th[i][itf] > th[max][itf]){
551       4582 :                 max = i;
552            :             }
553            :         }
554       1651 :         swap(th[h], th[max]);
555            :     }
556            :     // Ordenar por CODUSU
557       1655 :     for (int k = 0; k < th.size(); ++k) {
558            :         // Idea: Insertion Sort
559     114765 :         for (int j = k; j > 0  && th[j][itf] == th[j-1][itf]; --j) {
560     113114 :             if(th[j][codusuI] < th[j-1][codusuI]){
561      50692 :                 swap(th[j], th[j-1]);
562            :             }
563            :         }
564            :     }
565          4 :     return th;
566            : }
567            :
568         48 : int limIzquierdoCuartil(eph_h th, int c){
569         48 :     return th.size() * (c-1) / 4;
570            : }
571            :
572       5001 : int limDerechoCuarti(eph_h th, int c){
573       5001 :     return th.size() * c / 4;
574            : }
575            :
576         16 : float promedIngHogarCuart(eph_h th, int c){
577         16 :     int ingresosTotales = 0;
578         16 :     int hogaresEnCuartil = 0;
579            :     // estaEnCuartil
580       1667 :     for (int i = limIzquierdoCuartil(th, c); i < limDerechoCuarti(th, c); ++i) {
581       1651 :         ingresosTotales += th[i][itf];
582       1651 :         hogaresEnCuartil += 1;
583            :     }
584         16 :     return float(ingresosTotales) / float(hogaresEnCuartil);
585            : }
586            :
587         16 : float promedCompHogarCuart(eph_i ti, eph_h th, int c){
588         16 :     int componentes = 0;
589         16 :     int hogaresEnCuartil = 0;
590            :     // estaEnCuartil
591       1667 :     for (int h = limIzquierdoCuartil(th, c); h < limDerechoCuarti(th, c); ++h) {
592    3224776 :         for (int i = 0; i < ti.size(); ++i) {
593    3223125 :             if (ti[i][codusuI] == th[h][codusuH]){
594       3940 :                 componentes += 1;
595            :             }
596            :         }
597       1651 :         hogaresEnCuartil += 1;
598            :     }
599         16 :     return float(componentes) / float(hogaresEnCuartil);
600            : }
601            :
```

```
602          8 : int promCasosCuartil(eph_h th1, eph_h th2, int c){
603          8 :     int hogaresEncuestados = 0;
604        846 :     for (int h = limIzquierdoCuartil(th1, c); h < limDerechoCuarti(th1,c); ++h) {
605        838 :         hogaresEncuestados += 1;
606            :     }
607        821 :     for (int k = limIzquierdoCuartil(th2, c); k < limDerechoCuarti(th2,c); ++k) {
608        813 :         hogaresEncuestados += 1;
609            :     }
610          8 :     return hogaresEncuestados / 2;
611            : }
612            :
613            : // Resolucion de ejercicio
614          2 : lista_ev_hogares crecimientoHogarenoVsIngresos(eph_i ti1 , eph_h th1 , eph_i ti2 ,eph_h th2 ){
615          2 :     lista_ev_hogares result;
616          2 :     if (!sonDistintosTrimestres(ti1, ti2, th1, th2)){
617            :         // tablas de hogares ordenadas
618          4 :         eph_h tordh1 = hogaresOrdenadosPorITFyCODUSU(th1);
619          4 :         eph_h tordh2 = hogaresOrdenadosPorITFyCODUSU(th2);
620            :
621         10 :         for (int c = 1; c < 5; ++c) {
622          8 :             evolucion_hogar cuartil;
623          8 :             float promedioIng = 0;
624          8 :             float promedioComp = 0;
625          8 :             float promedioCasos = promCasosCuartil(tordh1, tordh2, c);
626            :
627          8 :             float promedioIngH1 = promedIngHogarCuart(tordh1, c);
628          8 :             float promedioCompH1 = promedCompHogarCuart(ti1, tordh1, c);
629          8 :             float promedioIngH2 = promedIngHogarCuart(tordh2, c);
630          8 :             float promedioCompH2 = promedCompHogarCuart(ti2, tordh2, c);
631            :
632          8 :             if(promedioIngH1 > 0) {
633          7 :                 promedioIng = 100 * (promedioIngH2 - promedioIngH1) / promedioIngH1;
634            :             }
635            :
636          8 :             if(promedioCompH1 > 0){
637          8 :                 promedioComp = 100 * (promedioCompH2 - promedioCompH1) / promedioCompH1;
638            :             }
639            :
640          8 :             cuartil = make_tuple(c, promedioComp, promedioIng, promedioCasos);
641          8 :             result.push_back(cuartil);
642            :         }
643            :     }else{
644          0 :         cout << "Las encuestas no son validas pues coinciden en las fechas" << endl;
645            :     }
646          2 :     return result;
647            : }
648            :
```

Generated by: LCOV version 1.13