

```
1 #define function unsigned int
2 #define var unsigned int
3
4 /*****/
5     Funciones básicas.
6 /*****/
7 function cero(){
8     return 0;
9 }
10
11 function cero(var X1){
12     return 0;
13 }
14
15 function S(var X1){
16     X1++;
17     return X1;
18 }
19
20 function U1_1(var X1){
21     return X1;
22 }
23
24 function U2_1(var X1, var X2){
25     return X1;
26 }
27
28 function U2_2(var X1, var X2){
29     return X2;
30 }
31
32 function U3_1(var X1, var X2, var X3){
33     return X1;
34 }
35
36 function U3_2(var X1, var X2, var X3){
37     return X2;
38 }
```

```
39 function U3_3(var X1, var X2, var X3){
40     return X3;
41 }
42
43 function U4_1(var X1, var X2, var X3, var X4){
44     return X1;
45 }
46
47 function U4_2(var X1, var X2, var X3, var X4){
48     return X2;
49 }
50
51 function U4_3(var X1, var X2, var X3, var X4){
52     return X3;
53 }
54
55 function U4_4(var X1, var X2, var X3, var X4){
56     return X4;
57 }
58
59 function U5_1(var X1, var X2, var X3, var X4, var X5){
60     return X1;
61 }
62
63 function U5_2(var X1, var X2, var X3, var X4, var X5){
64     return X2;
65 }
66
67 function U5_3(var X1, var X2, var X3, var X4, var X5){
68     return X3;
69 }
70
71 function U5_4(var X1, var X2, var X3, var X4, var X5){
72     return X4;
73 }
74
75 function U5_5(var X1, var X2, var X3, var X4, var X5){
76     return X5;
77 }
```

```
78 /*****
79
80 /* Devuelve siempre 0. */
81 function cero(var X1, var X2){
82     var V1 = U2_1(X1, X2);
83     return cero(V1);
84 }
85
86 /* Devuelve siempre 1. */
87 function uno(var X1){
88     var V1 = cero(X1);
89     return S(V1);
90 }
91
92 /* Devuelve siempre 1. */
93 function uno(var X1, var X2, var X3){
94     var V1 = U3_2(X1, X2, X3);
95     return uno(V1);
96 }
97
98 /* Devuelve siempre 2. */
99 function dos(var X1){
100     var V1 = uno(X1);
101     return S(V1);
102 }
103
104 /* Devuelve siempre 2. */
105 function dos(var X1, var X2){
106     var V1 = U2_1(X1, X2);
107     return dos(V1);
108 }
109
110 /* Devuelve siempre 2. */
111 function dos(var X1, var X2, var X3){
112     var V1 = U3_2(X1, X2, X3);
113     return dos(V1);
114 }
```

```

115 /* Devuelve siempre 3. */
116 function tres(var X1, var X2){
117     var V1 = dos(X1, X2);
118     return S(V1);
119 }
120
121 /* Devuelve siempre 3. */
122 function tres(var X1, var X2, var X3){
123     var V1 = dos(X1, X2, X3);
124     return S(V1);
125 }
126
127 /* Devuelve siempre 4. */
128 function cuatro(var X1, var X2){
129     var V1 = tres(X1, X2);
130     return S(V1);
131 }
132
133 /* Devuelve siempre 4. */
134 function cuatro(var X1, var X2, var X3){
135     var V1 = tres(X1, X2, X3);
136     return S(V1);
137 }
138
139 /* ----- */
140
141 /* Función auxiliar para la función «suma». */
142 function suma_aux(var X1, var X2, var X3){
143     var V1 = U3_2(X1, X2, X3);
144     return S(V1);
145 }

```

```

146 /* Devuelve la suma de X1 y X2. */
147 function suma(var X1, var X2){
148     if(X1 == 0){
149         return U1_1(X2);
150     }
151     else{
152         X1--;
153         var rr = suma(X1, X2);
154         return suma_aux(X1, rr, X2);
155     }
156 }
157
158 /* ----- */
159
160 /* Auxiliar para «suma3». */
161 function suma_aux3(var X1, var X2, var X3){
162     var V1 = U3_2(X1, X2, X3);
163     var V2 = U3_3(X1, X2, X3);
164     return suma(V1, V2);
165 }
166
167 /* Devuelve la suma de X1, X2 y X3. */
168 function suma3(var X1, var X2, var X3){
169     var V1 = U3_1(X1, X2, X3);
170     var V2 = suma_aux3(X1, X2, X3);
171     return suma(V1, V2);
172 }
173
174 /* Auxiliar para «suma4». */
175 function suma_aux4(var X1, var X2, var X3, var X4){
176     var V1 = U4_2(X1, X2, X3, X4);
177     var V2 = U4_3(X1, X2, X3, X4);
178     var V3 = U4_4(X1, X2, X3, X4);
179     return suma3(V1, V2, V3);
180 }

```

```

181 /* Devuelve la suma de X1, X2, X3 y X4. */
182 function suma4(var X1, var X2, var X3, var X4){
183     var V1 = U4_1(X1, X2, X3, X4);
184     var V2 = suma_aux4(X1, X2, X3, X4);
185     return suma(V1, V2);
186 }
187
188 /* Auxiliar para «suma5». */
189 function suma_aux5(var X1, var X2, var X3, var X4, var X5){
190     var V1 = U5_2(X1, X2, X3, X4, X5);
191     var V2 = U5_3(X1, X2, X3, X4, X5);
192     var V3 = U5_4(X1, X2, X3, X4, X5);
193     var V4 = U5_5(X1, X2, X3, X4, X5);
194     return suma4(V1, V2, V3, V4);
195 }
196
197 /* Devuelve la suma de X1, X2, X3, X4 y X5. */
198 function suma5(var X1, var X2, var X3, var X4, var X5){
199     var V1 = U5_1(X1, X2, X3, X4, X5);
200     var V2 = suma_aux5(X1, X2, X3, X4, X5);
201     return suma(V1, V2);
202 }
203
204 /* ----- */
205
206 /* Función auxiliar para la función «resta».
207  * devuelve X1 - 1. */
208 function anterior(var X1){
209     if(X1 == 0){
210         return cero();
211     }
212     else{
213         X1--;
214         var rr = anterior(X1);
215         return U2_1(X1, rr);
216     }
217 }

```

```

218 /* Función auxiliar para la función «resta». */
219 function resta_aux2(var X1, var X2, var X3){
220     var V1 = U3_2(X1, X2, X3);
221     return anterior(V1);
222 }
223
224 /* Función auxiliar para la función «resta». */
225 function resta_aux1(var X1, var X2){
226     if(X1 == 0){
227         return U1_1(X2);
228     }
229     else{
230         X1--;
231         var rr = resta_aux1(X1, X2);
232         return resta_aux2(X1, rr, X2);
233     }
234 }
235
236 /* Devuelve X1 menos X2. */
237 function resta(var X1, var X2){
238     var V1 = U2_2(X1, X2);
239     var V2 = U2_1(X1, X2);
240     return resta_aux1(V1, V2);
241 }
242
243 /* ----- */
244
245 /* Función auxiliar para la función «mult». */
246 function mult_aux(var X1, var X2, var X3){
247     var V1 = U3_2(X1, X2, X3);
248     var V2 = U3_3(X1, X2, X3);
249     return suma(V1, V2);
250 }

```

```

251 /* Devuelve la multiplicación de X1 y X2. */
252 function mult(var X1, var X2){
253     if(X1 == 0){
254         return cero(X2);
255     }
256     else{
257         X1--;
258         var rr = mult(X1, X2);
259         return mult_aux(X1, rr, X2);
260     }
261 }
262
263 /* ----- */
264
265 /* Función auxiliar para la función «mult_veces». */
266 function mult_veces_aux(var X1, var X2, var X3, var X4){
267     var V1 = U4_2(X1, X2, X3, X4);
268     var V2 = U4_3(X1, X2, X3, X4);
269     return mult(V1, V2);
270 }
271
272 /* Devuelve X3 multiplicado X1 veces por X2;
273  * esto es,  $X3 * (X2 ^ X1)$ . */
274 function mult_veces(var X1, var X2, var X3){
275     if(X1 == 0){
276         return U2_2(X2, X3);
277     }
278     else{
279         X1--;
280         var rr = mult_veces(X1, X2, X3);
281         return mult_veces_aux(X1, rr, X2, X3);
282     }
283 }
284
285 /* ----- */
286

```



```

287 /* Devuelve 0 si X1 es mayor que 0 y 1 si es 0. */
288 function no_sg(var X1){
289     var V1 = uno(X1);
290     var V2 = U1_1(X1);
291     return resta(V1, V2);
292 }
293
294 /* Devuelve 0 si X1 es 0 y 1 si es mayor que 0. */
295 function sg(var X1){
296     var V1 = uno(X1);
297     var V2 = no_sg(X1);
298     return resta(V1, V2);
299 }
300
301 /* Devuelve 1 si X1 es menor o igual que X2;
302  * 0 en caso contrario. */
303 function menor_ig(var X1, var X2){
304     var V1 = resta(X1, X2);
305     return no_sg(V1);
306 }
307
308 /* Devuelve 1 si X1 es mayor o igual que X2;
309  * 0 en caso contrario. */
310 function mayor_ig(var X1, var X2){
311     var V1 = U2_2(X1, X2);
312     var V2 = U2_1(X1, X2);
313     return menor_ig(V1, V2);
314 }
315
316 /* Devuelve 1 si X1 es igual que X2; 0 en caso contrario. */
317 function igual(var X1, var X2){
318     var V1 = mayor_ig(X1, X2);
319     var V2 = menor_ig(X1, X2);
320     return mult(V1, V2);
321 }
322
323 /* ----- */
324

```

```

325 /* Declaración para funciones que usan recursión indirecta. */
326 function div_emp(var, var, var, var);
327
328 /* Función auxiliar para «div».
329  * Comprueba si dividendo es mayor que el divisor. */
330 function div_aux_mayor_ig(var X1, var X2, var X3){
331     var V1 = U3_2(X1, X2, X3);
332     var V2 = U3_3(X1, X2, X3);
333     return mayor_ig(V1, V2);
334 }
335
336 /* Función auxiliar para «div».
337  * Suma uno al resultado provisional. */
338 function div_aux_sig(var X1, var X2, var X3, var X4, var X5){
339     var V1 = U5_3(X1, X2, X3, X4, X5);
340     return S(V1);
341 }
342
343 /* Función auxiliar para «div».
344  * Resta divisor al dividendo. */
345 function div_aux_rt(var X1, var X2, var X3, var X4, var X5){
346     var V1 = U5_4(X1, X2, X3, X4, X5);
347     var V2 = U5_5(X1, X2, X3, X4, X5);
348     return resta(V1, V2);
349 }
350
351 /* Función auxiliar para «div».
352  * Suma uno al resultado, resta al
353  * dividendo el divisor y empieza el ciclo. */
354 function div_aux2(var X1, var X2, var X3, var X4, var X5){
355     var V1 = div_aux_sig(X1, X2, X3, X4, X5);
356     var V3 = div_aux_rt(X1, X2, X3, X4, X5);
357     var V2 = U5_5(X1, X2, X3, X4, X5);
358     return div_emp(V1, V2, V3);
359 }
360

```

```

361 /* Función auxiliar para «div».
362  * X1: ¿es dividendo mayor que divisor?
363  * X2: resultado provisional.
364  * X3: dividendo.
365  * X4: divisor. */
366 function div_aux1(var X1, var X2, var X3, var X4){
367     if(X1 == 0){
368         return U3_1(X2, X3, X4);
369     }
370     else{
371         X1--;
372         var rr = div_aux1(X1, X2, X3, X4);
373         return div_aux2(X1, rr, X2, X3, X4);
374     }
375 }
376
377 /* Función auxiliar para «div».
378  * X1: resultado provisional.
379  * X2: dividendo.
380  * X3: divisor. */
381 function div_emp(var X1, var X2, var X3){
382     var V1 = div_aux_mayor_ig(X1, X2, X3);
383     var V2 = U3_1(X1, X2, X3);
384     var V3 = U3_2(X1, X2, X3);
385     var V4 = U3_3(X1, X2, X3);
386     return div_aux1(V1, V2, V3, V4);
387 }
388
389 /* Devuelve n, donde  $n = X1/X2$ .
390  * X1: dividendo.
391  * X2: divisor. */
392 function div(var X1, var X2){
393     var V1 = cero(X1, X2);
394     var V2 = U2_1(X1, X2);
395     var V3 = U2_2(X1, X2);
396     return div_emp(V1, V2, V3);
397 }
398
399 /* ----- */

```

```

400 /* Función auxiliar para «resto».
401  * Devuelve multiplicación de división y divisor. */
402 function resto_aux2(var X1, var X2, var X3){
403     var V1 = U3_1(X1, X2, X3);
404     var V2 = U3_3(X1, X2, X3);
405     return mult(V1, V2);
406 }
407
408 /* Función auxiliar para «resto». Resta al
409  * dividiendo la multiplicación de división y divisor. */
410 function resto_aux1(var X1, var X2, var X3){
411     var V1 = U3_2(X1, X2, X3);
412     var V2 = resto_aux2(X1, X2, X3);
413     return resta(V1, V2);
414 }
415
416 /* Devuelve el resto de X1/X2, esto es, X1 mod X2.
417  * X1: dividendo.
418  * X2: divisor. */
419 function resto(var X1, var X2){
420     var V1 = div(X1, X2);
421     var V2 = U2_1(X1, X2);
422     var V3 = U2_2(X1, X2);
423     return resto_aux1(V1, V2, V3);
424 }
425
426 /* ----- */
427
428 /* Devuelve 1 si X1 es divisible entre X2;
429  * 0 en caso contrario. */
430 function es_div(var X1, var X2){
431     var V1 = resto(X1, X2);
432     return no_sg(V1);
433 }
434

```

```

435 /* Devuelve 1 si X1 no es divisible
436    * entre X2; 0 en caso contrario. */
437 function no_es_div(var X1, var X2){
438     var V1 = es_div(X1, X2);
439     return no_sg(V1);
440 }
441
442 /* ----- */
443
444 /* Declaración para funciones que usan recursión indirecta. */
445 function vdiv_emp(var, var, var);
446
447 /* Función auxiliar para «vdiv».
448    * Devuelve 1 si es divisible X2 entre X3; si no 0. */
449 function vdiv_aux_es_div(var X1, var X2, var X3){
450     var V1 = U3_2(X1, X2, X3);
451     var V1 = U3_3(X1, X2, X3);
452     return es_div(V1, V2);
453 }
454
455 /* Función auxiliar para «vdiv».
456    * Divide X4 entre X5. */
457 function vdiv_aux_div(var X1,var X2,var X3,var X4,var X5){
458     var V1 = U5_4(X1, X2, X3, X4, X5);
459     var V2 = U5_5(X1, X2, X3, X4, X5);
460     return div(V1, V2);
461 }
462
463 /* Función auxiliar para «vdiv».
464    * Suma 1 al resultado provisional y divide
465    * dividendo entre divisor. */
466 function vdiv_aux2(var X1, var X2, var X3, var X4,var X5){
467     var V1 = div_aux_sig(X1, X2, X3, X4, X5);
468     var V2 = vdiv_aux_div(X1, X2, X3, X4, X5);
469     var V3 = U5_5(X1, X2, X3, X4, X5);
470     return vdiv_emp(V1, V2, V2);
471 }
472

```

```

473 /* Función auxiliar para «vdiv». Devuelve el resultado
474  * si X3 no es divisible entre X4; si no, sigue el ciclo. */
475 function vdiv_aux1(var X1, var X2, var X3, var X4){
476     if(X1 == 0){
477         return U3_1(X2, X3, X4);
478     }
479     else{
480         X1--;
481         var rr = vdiv_aux1(X1, X2, X3, X4);
482         return vdiv_aux2(X1, rr, X2, X3, X4);
483     }
484 }
485
486 /* Función auxiliar para «vdiv». Comienza el ciclo. */
487 function vdiv_emp(var X1, var X2, var X3){
488     var V1 = vdiv_aux_es_div(X1, X2, X3);
489     var V2 = U3_1(X1, X2, X3);
490     var V3 = U3_2(X1, X2, X3);
491     var V4 = U3_3(X1, X2, X3);
492     return vdiv_aux1(V1, V2, V3, V4);
493 }
494
495 /* Devuelve las veces que es divisible X1 entre X2. */
496 function vdiv(var X1, var X2){
497     var V1 = cero(X1, X2);
498     var V2 = U2_1(X1, X2);
499     var V3 = U2_2(X1, X2);
500     return vdiv_emp(V1, V2, V3);
501 }
502
503 /* ----- */
504
505 /* Función auxiliar para «nd_hasta_aux2».
506  * Suma 2 para que no se compruebe si es divisible entre 0 o 1.*/
507 function nd_hasta_aux3(var X1, var X2, var X3){
508     var V1 = U3_1(X1, X2, X3);
509     var V2 = dos(X1, X2, X3);
510     return suma(V1, V2);
511 }

```

```

512 /* Función auxiliar para «nd_hasta_aux1». */
513 function nd_hasta_aux2(var X1, var X2, var X3){
514     var V1 = U3_3(X1, X2, X3);
515     var V2 = nd_hasta_aux3(X1, X2, X3);
516     return no_es_div(V1, V2);
517 }
518
519 /* Función auxiliar para «no_div_hasta». */
520 function nd_hasta_aux1(var X1, var X2, var X3){
521     var V2 = nd_hasta_aux2(X1, X2, X3);
522     var V1 = U3_2(X1, X2, X3);
523     return mult(V1, V2);
524 }
525
526 /* Función auxiliar para «es_primo».
527  * Devuelve 1 si no hay divisor de X2 entre 2 y X1 + 1. */
528 function no_div_hasta(var X1, var X2){
529     if(X1 == 0){
530         return uno(X2);
531     }
532     else{
533         X1--;
534         var rr = no_div_hasta(X1, X2);
535         return nd_hasta_aux1(X1, rr, X2);
536     }
537 }
538
539 /* Función auxiliar para «es_primo». */
540 function partir(var X1){
541     var V1 = anterior(X1);
542     var V2 = dos(X1);
543     return div(V1, V2);
544 }
545
546 /* Devuelve 1 si X1 es primo; 0 en caso contrario. */
547 function es_primo(var X1){
548     var V1 = partir(X1);
549     var V2 = U1_1(X1);
550     return no_div_hasta(V1, V2);
551 }

```

```

552 /* Devuelve 0 si X1 es primo; 1 en caso contrario. */
553 function no_es_primo(var X1){
554     var V1 = es_primo(X1);
555     return no_sg(V1);
556 }
557
558 /* ----- */
559
560 /* Declaración de función. */
561 function sig_primo_desde(var);
562
563 /* Función auxiliar para «sig_primo_aux1». */
564 function sig_primo_aux2(var X1, var X2, var X3){
565     var V1 = U3_2(X1, X2, X3);
566     return sig_primo_desde(V1);
567 }
568
569 /* Si X2 es primo lo devuelve; si no
570  * sigue buscando el menor primo mayor que X2. */
571 function sig_primo_aux1(var X1, var X2){
572     if(X1 == 0){
573         return U1_1(X2);
574     }
575     else{
576         X1--;
577         var rr = sig_primo_aux1(X1, X2);
578         return sig_primo_aux2(X1, rr, X2);
579     }
580 }
581
582 /* Devuelve el menor primo mayor o igual que X1. */
583 function sig_primo(var X1){
584     var V1 = no_es_primo(X1);
585     var V2 = U1_1(X1);
586     return sig_primo_aux1(V1, V2);
587 }
588

```



```

589 /* Devuelve el menor primo mayor que X1. */
590 function sig_primo_desde(var X1){
591     var V1 = S(X1);
592     return sig_primo(V1);
593 }
594
595 /* Devuelve el menor primo mayor que el primo X2. */
596 function inst_primo_aux(var X1, var X2){
597     var V1 = U2_2(X1, X2);
598     return sig_primo_desde(V1);
599 }
600
601 /* Devuelve el primo con el que se
602  * codifica la instrucción en la posición X1. */
603 function inst_primo(var X1){
604     if(X1 == 0){
605         return dos();
606     }
607     else{
608         X1--;
609         var rr = inst_primo(X1);
610         return inst_primo_aux(X1, rr);
611     }
612 }
613 /* ----- */
614 /* Devuelve el primo correspondiente a la posición X1.
615  * X1: posición. */
616 function primo_num_inst(var X1, var X2){
617     var V1 = U2_1(X1, X2);
618     return inst_primo(V1);
619 }
620 /* Devuelve la instrucción en la posición X1.
621  * X1: posición.
622  * X2: registro. */
623 function inst(var X1, var X2){
624     var V1 = U2_2(X1, X2);
625     var V2 = primo_num_inst(X1, X2);
626     return vdiv(V1, V2);
627 }

```

```

628 /* Devuelve 1 si X1 es instrucción; 0 en caso contrario.
629  * X1: instrucción. */
630 function hay_inst(var X1, var X2){
631     var V1 = U2_1(X1, X2);
632     return sg(V1);
633 }
634
635 /* Devuelve: 0 si no hay instrucción o ésta es '1';
636  * 1 si '0'.
637  * 2 si '='.
638  * 3 si '*'. */
639 function tipo_inst_aux(var X1, var X2){
640     var V1 = U2_1(X1, X2);
641     var V2 = cuatro(X1, X2);
642     return resto(V1, V2);
643 }
644
645 /* Devuelve, según la instrucción en X1:
646  * 0 si no hay instrucción.
647  * 1 si '1'.
648  * 2 si '0'.
649  * 3 si '='.
650  * 4 si '*'. */
651 function tipo_inst(var X1, var X2){
652     var V1 = tipo_inst_aux(X1, X2);
653     var V2 = hay_inst(X1, X2);
654     return suma(V1, V2);
655 }
656
657 /* Auxiliar para «pos_inst».
658  * suma 3 para que al dividir entre 4
659  * no dé nunca 0 y dé el resultado correcto. */
660 function pos_inst_aux(var X1, var X2){
661     var V1 = U2_1(X1, X2);
662     var V2 = tres(X1, X2);
663     return suma(V1, V2);
664 }
665

```

```

666 /* Devuelva la posición a la
667  * que se refiere la instrucción X1. */
668 function pos_inst(var X1, var X2){
669     var V1 = pos_inst_aux(X1, X2);
670     var V2 = cuatro(X1, X2);
671     return div(V1, V2);
672 }
673
674 /* Devuelva la posición a la que
675  * señala el puntero en el registro X1. */
676 function sacar_puntero(var X1){
677     var V1 = U1_1(X1);
678     var V2 = dos(X1);
679     return vdiv(V1, V2);
680 }
681
682 /* Devuelve el registro con
683  * el puntero avanzado una posición. */
684 function avanzar_inst(var X1, var X2){
685     var V1 = dos(X1, X2);
686     var V2 = U2_2(X1, X2);
687     return mult(V1, V2);
688 }
689
690 /* ----- */
691
692 /* Devuelve el registro X2 modificado con una marca más
693  * en la posición X1, o un símbolo «1» si estaba vacía,
694  * y el puntero señalando a la siguiente posición. */
695 function marcar(var X1, var X2){
696     var V1 = cuatro(X1, X2);
697     var V2 = primo_num_inst(X1, X2);
698
699     var V3 = avanzar_inst(X1, X2);
700     return mult_veces(V1, V2, V3);
701 }
702
703 /* ----- */

```

```

704 /* Auxiliar para «borrar_aux2». Resta para mantener
705  * el símbolo en la posición referida si éste no es '1'. */
706 function borrar_aux3(var X1, var X2){
707     var V1 = U2_1(X1, X2);
708     var V2 = tipo_inst_aux(X1, X2);
709     return resta(V1, V2);
710 }
711
712 /* Auxiliar para «borrar_aux».
713  * X1: instrucción en posición referida.
714  * X2: primo en posición referida. */
715 function borrar_aux2(var X1, var X2){
716     var V1 = borrar_aux3(X1, X2);
717     var V2 = U2_2(X1, X2);
718     var V2 = uno(X1, X2);
719     return mult_veces(V1, V2, V3);
720 }
721
722 /* Devuelve el número entre el que hay que dividir
723  * el registro X2 para borrar la posición X1.
724  * X1: posición referida.
725  * X2: registro. */
726 function borrar_aux1(var X1, var X2){
727     var V1 = inst(X1, X2);
728     var V2 = primo_num_inst(X1, X2);
729     return borrar_aux2(V1, V2);
730 }
731
732 /* Devuelve el registro con la posición X1 borrada.
733  * X1: posición referida.
734  * X2: registro. */
735 function borrar(var X1, var X2){
736     var V1 = avanzar_inst(X1, X2);
737     var V2 = borrar_aux1(X1, X2);
738     return div(V1, V2);
739 }
740
741 /* ----- */
742

```

```

743 /* Devuelve la posición del puntero en registro X2. */
744 function quitar_puntero_aux2(var X1, var X2){
745     var V1 = U2_2(X1, X2);
746     return sacar_puntero(V1);
747 }
748
749 /* Devuelve el número entre el que hay
750  * que dividir registro X2 para quitarle el puntero. */
751 function quitar_puntero_aux1(var X1, var X2){
752     var V1 = quitar_puntero_aux2(X1, X2);
753     var V2 = dos(X1, X2);
754     var V3 = uno(X1, X2);
755     return mult_veces(V1, V2, V3);
756 }
757
758 /* Devuelve el registro X2 sin puntero.*/
759 function quitar_puntero(var X1, var X2){
760     var V1 = quitar_puntero_aux1(X1, X2);
761     var V2 = U2_2(X1, X2);
762     return div(V1, V2);
763 }
764
765 /* Devuelve el registro X2 con el
766  * puntero señalando a la posición X1. */
767 function saltar(var X1, var X2){
768     var V1 = U2_1(X1, X2);
769     var V2 = dos(X1, X2);
770     var V3 = quitar_puntero(X1, X2);
771     return mult_veces(V1, V2, V3);
772 }
773
774 /* ----- */
775 /* Devuelve la instrucción
776  * en la primera posición de registro X2. */
777 function inst_uno(var X1, var X2){
778     var V1 = uno(X1, X2);
779     var V2 = U2_2(X1, X2);
780     return inst(V1, V2);
781 }

```

```

782
783 /* Devuelve la posición referida por la
784  * instrucción en la primera posición de registro X2. */
785 function v_pos_uno(var X1, var X2){
786     var V1 = inst_uno(X1, X2);
787     var V2 = U2_2(X1, X2);
788     return pos_inst(V1, V2);
789 }
790
791 /* Devuelve la posición referida por la
792  * instrucción X1 de registro X2. */
793 function v_pos(var X1, var X2){
794     var V1 = inst(X1, X2);
795     var V2 = U2_2(X1, X2);
796     return pos_inst(V1, V2);
797 }
798
799 /* Devuelve 1 si la posición X1 de registro X2
800  * es igual que la primera; 0 en caso contrario. */
801 function hay_salto(var X1, var X2){
802     var V1 = v_pos_uno(X1, X2);
803     var V2 = v_pos(X1, X2);
804     return es_igual(V1, V2);
805 }
806
807 /* Devuelve el registro X2 con el puntero
808  * señalando a la siguiente posición de la actual si
809  * la posición X1 es distinta de la primera posición;
810  * a la siguiente de la siguiente en caso contrario. */
811 function comparar(var X1, var X2){
812     var V1 = hay_salto(X1, X2);
813     var V2 = dos(X1, X2);
814
815     var V3 = avanzar_inst(X1, X2);
816     return mult_veces(V1, V2, V3);
817 }
818
819 /* ----- */

```

```

820 /* Devuelve el registro X3 después de ejecutar 'marcar'. */
821 function eval_marcar_aux(var X1, var X2, var X3){
822     var V1 = U3_2(X1, X2, X3);
823     var V2 = U3_3(X1, X2, X3);
824     return marcar(V1, V2);
825 }
826
827 /* Si el tipo de instrucción X1 es 'marcar',
828  * devuelve 1; 0 en caso contrario. */
829 function es_marcar(var X1, var X2, var X3){
830     var V1 = U3_1(X1, X2, X3);
831     var V2 = uno(X1, X2, X3);
832     return es_igual(V1, V2);
833 }
834
835 /* Si X1 es 1 devuelve el registro X3
836  * después de ejecutar la instrucción 'marcar'
837  * en la posición X2; devuelve 0 si X1 no es 1. */
838 function eval_marcar(var X1, var X2, var X3){
839     var V1 = es_marcar(X1, X2, X3);
840     var V2 = eval_marcar_aux(X1, X2, X3);
841     return mult(V1, V2);
842 }
843 /* ----- */
844
845 /* Devuelve el registro X3 después de ejecutar 'borrar'. */
846 function eval_borrar_aux(var X1, var X2, var X3){
847     var V1 = U3_2(X1, X2, X3);
848     var V2 = U3_3(X1, X2, X3);
849     return borrar(V1, V2);
850 }
851
852 /* Si el tipo de instrucción X1 es
853  * 'borrar', devuelve 1; 0 en caso contrario. */
854 function es_borrar(var X1, var X2, var X3){
855     var V1 = U3_1(X1, X2, X3);
856     var V2 = dos(X1, X2, X3);
857     return es_igual(V1, V2);
858 }

```

```

859
860 /* Si X1 es 2 devuelve el registro X3
861  * después de ejecutar la instrucción 'borrar'
862  * en la posición X2; devuelve 0 si X1 no es 2. */
863 function eval_borrar(var X1, var X2, var X3){
864     var V1 = es_borrar(X1, X2, X3);
865     var V2 = eval_borrar_aux(X1, X2, X3);
866     return mult(V1, V2);
867 }
868
869 /* ----- */
870
871 /* Devuelve el registro X3 después de ejecutar 'comparar'. */
872 function eval_comparar_aux(var X1, var X2, var X3){
873     var V1 = U3_2(X1, X2, X3);
874     var V2 = U3_3(X1, X2, X3);
875     return comparar(V1, V2);
876 }
877
878 /* Si el tipo de instrucción X1 es
879  * 'comparar', devuelve 1; 0 en caso contrario. */
880 function es_comparar(var X1, var X2, var X3){
881     var V1 = U3_1(X1, X2, X3);
882     var V2 = tres(X1, X2, X3);
883     return es_igual(V1, V2);
884 }
885
886 /* Si X1 es 3 devuelve el registro X3
887  * después de ejecutar la instrucción 'comparar'
888  * en la posición X2; devuelve 0 si X1 no es 3. */
889 function eval_comparar(var X1, var X2, var X3){
890     var V1 = es_comparar(X1, X2, X3);
891
892     var V2 = eval_comparar_aux(X1, X2, X3);
893     return mult(V1, V2);
894 }
895 /* ----- */
896

```



```

897 /* Devuelve el registro X3 después de ejecutar 'saltar'. */
898 function eval_saltar_aux(var X1, var X2, var X3){
899     var V1 = U3_2(X1, X2, X3);
900     var V2 = U3_3(X1, X2, X3);
901     return saltar(V1, V2);
902 }
903
904 /* Si el tipo de instrucción X1 es 'saltar', devuelve 1;
905  * 0 en caso contrario. */
906 function es_saltar(var X1, var X2, var X3){
907     var V1 = U3_1(X1, X2, X3);
908     var V2 = cuatro(X1, X2, X3);
909     return es_igual(V1, V2);
910 }
911 /* Si X1 es 4 devuelve el registro X3
912  * después de ejecutar la instrucción 'saltar'
913  * en la posición X2; devuelve 0 si X1 no es 4. */
914 function eval_saltar(var X1, var X2, var X3){
915     var V1 = es_saltar(X1, X2, X3);
916     var V2 = eval_saltar_aux(X1, X2, X3);
917     return mult(V1, V2);
918 }
919
920 /* ----- */
921
922 /* Función auxiliar para «eval_no_inst». */
923 function eval_no_inst_aux(var X1, var X2, var X3){
924     var V1 = U3_1(X1, X2, X3);
925     return no_sg(V1);
926 }
927
928 /* Devuelve el registro X3 sin modificar
929  * si el tipo de instrucción X1 es 0;
930  * devuelve 0 en caso contrario. */
931 function eval_no_inst(var X1, var X2, var X3){
932     var V1 = no_inst_aux(X1, X2, X3);
933     var V2 = U3_3(X1, X2, X3);
934     return mult(V1, V2);
935 }

```

```

936 /* ----- */
937
938 /* Devuelve 1 si hay instrucción en X1; 0 en caso contrario . */
939 function hay_inst(var X1, var X2, var X3){
940     var V1 = U3_1(X1, X2, X3);
941     return sg(V1);
942 }
943
944 /* Devuelve el registro después de ejecutar la intrucción
945  * que indica X1 o el registro sin tocar si X1 es 0. */
946 function eval_registro(var X1, var X2, var X3){
947     var V1 = eval_marcar(X1, X2, X3);
948     var V2 = eval_borrar(X1, X2, X3);
949     var V3 = eval_comparar(X1, X2, X3);
950     var V4 = eval_saltar(X1, X2, X3);
951     var V5 = eval_no_inst(X1, X2, X3);
952     return suma(V1, V2, V3, V4, V5);
953 }
954
955 /* Comprueba si hay instrucción, ejecuta
956  * en caso de haberla y vuelve a empezar el ciclo. */
957 function ejec3(var X1, var X2, var X3){
958     var V1 = hay_inst(X1, X2, X3);
959     var V2 = eval_registro(X1, X2, X3);
960     return evaluar(V1, V2);
961 }
962
963 /* Saca el tipo de instrucción, la
964  * posición referida y llama a «ejec3».
965  * X1: instrucción.
966  * X2: registro. */
967 function ejec2(var X1, var X2){
968     var V1 = tipo_inst(X1, X2);
969     var V2 = pos_inst(X1, X2);
970     var V3 = U2_2(X1, X2);
971     return ejec3(V1, V2, V3);
972 }
973

```

```

974 /* Saca la instrucción y llama a «ejec2» con ésta y el registro X2.
975  * X1: puntero.
976  * X2: registro. */
977 function ejec1(var X1, var X2){
978     var V1 = inst(X1, X2);
979     var V2 = U2_2(X1, X2);
980     return ejec2(V1, V2);
981 }
982
983 /* Toma el registro X1 y llama a «ejec1»
984  * con éste y el puntero contenido en él. */
985 function ejecutar_inst(var X1){
986     var V1 = sacar_puntero(X1);
987     var V2 = U1_1(X1);
988     return ejec1(V1, V2);
989 }
990
991 /* X3: registro. */
992 function ejecutar(var X1, var X2, var X3){
993     var V1 = U3_3(X1, X2, X3);
994     return ejecutar_inst(V1);
995 }
996
997 /*
998  * Si X1 es 0 devuelve el registro y acaba la computación;
999  * si es 1 ejecuta la siguiente instrucción, si la hubiera,
1000  * en registro X2; es decir, ejecuta el siguiente paso.
1001  */
1002 function evaluar(var X1, var X2){
1003     if(X1 == 0){
1004         return U1_1(X2);
1005     }
1006     else{
1007         X1--;
1008         var rr = evaluar(X1, X2);
1009         return ejecutar(X1, rr, X2);
1010     }
1011 }
1012

```

```

1013 /* Auxiliar para «computar». */
1014 function computar_aux(var X1, var X2){
1015     var V1 = uno(X1, X2);
1016     var V2 = mult(X1, X2);
1017     return evaluar(V1, V2);
1018 }
1019
1020 /*
1021  * Función recursiva primitiva que computa el mismo valor
1022  * que el programa C-- que recibe como argumento; en caso de termina
1023  * devuelve un registro con el puntero y programa después del cómputo
1024  */
1025 function computar(var X1){
1026     var V1 = dos(X1);
1027     var V2 = U1_1(X1);
1028     return computar_aux(V1, V2);
1029 }

```