

## **Memoria de la práctica de la asignatura de programación**

***Valentín Barros Puertas (insvbp00)***

## Índice:

|                                                                                           |    |
|-------------------------------------------------------------------------------------------|----|
| Notas sobre la aplicación y herramientas utilizadas para la elaboración de la misma ..... | 2  |
| Programa principal (pro.pas) .....                                                        | 3  |
| Menú principal (menu.pas) .....                                                           | 4  |
| Control de la terminal (display.pas) .....                                                | 9  |
| Entrada filtrada (filteredinput.pas) .....                                                | 15 |
| Tipos y base de datos (filesystem.pas) .....                                              | 29 |
| Ordenación (sort.pas) .....                                                               | 37 |
| Impresión de la lista de alumnos (printlist.pas) .....                                    | 44 |

## ***Notas sobre la aplicación.***

- La documentación relativa a la programación de la aplicación se encuentra en los mismos archivos fuente, los cuales se incluyen en este documento.
- La aplicación almacena la información relativa a las notas de Febrero y Junio en la misma sección de memoria, es decir, aunque la interfaz de la misma indique lo contrario —ofrece las opciones Febrero y Junio—, modificar la nota de Febrero modifica la de Junio, y viceversa. Esto se hace así debido a la interpretación de la frase “[...] las convocatorias posibles en el curso (**febrero/junio**, septiembre y diciembre) [...]” en las especificaciones.
- Al crear un nuevo alumno, todos los datos son opcionales a excepción del login, puesto que este es necesario para poder buscar un alumno en concreto en la base de datos. Los datos *en blanco* aparecerán en los listados como una zona sin texto en el caso de los datos textuales y con un cero en el caso de los numéricos.

## ***Herramientas utilizadas para la realización de la aplicación.***

- Geany [<http://www.geany.org/>] como editor de textos.
- GNU Pascal Compiler [<http://www.gnu-pascal.de/>] como compilador.
- GNU source-highlight [<http://www.gnu.org/software/src-highlite/>] y el procesador de textos de OpenOffice.org [<http://es.openoffice.org/>] para la realización de la memoria.
- LinuxMint [<http://www.linuxmint.com/>] y EeeBuntu [<http://www.eeebuntu.org/>] como sistemas operativos para los entornos de producción.

```

1: PROGRAM pro;
2:
3: {*
4:
5:     Programa creado para la práctica de la asignatura
6:     de programación del primer curso de Ing. Téc. en
7:     Inf. de Sistemas de la UDC.
8:
9:     Autor: Valentín Barros Puertas (insvbp00 en FIC).
10:    Versión: 1.0.1 [05/02/2009].
11:
12:    Pascal Estándar Extendido (ISO 10206).
13: *}
14:
15: IMPORT
16:
17:     filesystem;
18:     menu;
19:
20: VAR
21:
22:     labels:MenuLabels;
23:     status:integer value 255;
24:
25: BEGIN
26:
27:     labels['a'] := 'Nuevo alumno';
28:     labels['b'] := 'Modificar alumno';
29:     labels['c'] := 'Listar alumnos en orden alfabético';
30:     labels['d'] := 'Listar notas de una convocatoria (Orden alfabético)';
31:     labels['e'] := 'Listar notas de una convocatoria (Orden según las notas)';
32:     labels['f'] := 'Salir';
33:
34:     repeat
35:
36:         status := drawMenu(labels, status);
37:
38:     until status < 1;
39:
40: END.

```

```

1: MODULE menu INTERFACE;
2:
3: {*
4:     Este módulo proporciona la función para iniciar el programa, que a su vez
5:     llama a las funciones necesarias para cada opción de la aplicación.
6: *}
7:
8: export menu = (drawMenu, menuItems, menuLabels);
9:
10: CONST
11:
12:     {Indica la última opción disponible en el menú principal.}
13:     LAST_ITEM      = 'f';
14:
15: TYPE
16:
17:     {Tipos para el texto de las opciones del menú principal.}
18:     MenuItem = set of 'a'..LAST_ITEM;
19:     MenuLabels = array['a'..LAST_ITEM] of string(57);
20:
21: {*
22:     Dibuja el menú principal de la aplicación y llama a selectMenuOption.
23:     Devuelve un entero que es utilizado por el programa principal para
24:     determinar si debe repetir de nuevo drawMenu (> 0), en cuyo caso pasará
25:     dicho entero de nuevo a la función mediante m, el cual puede indicar a
26:     la misma que debe imprimir un mensaje informativo bajo el título de la
27:     aplicación. options son las opciones disponibles en el menú principal.
28: *}
29: function drawMenu(var options:MenuLabels; m:integer):integer;
30:
31: {*
32:     Pide un caracter mediante la entrada estándar, el cual se utiliza para
33:     determinar qué opción de las disponibles (options) desea seleccionar
34:     el usuario. Devuelve un entero el cual es utilizado por el programa para
35:     determinar si debe repetir drawMenu o permitir la finalización de la
36:     aplicación (realmente es la propia función drawMenu la que devuelve este
37:     mismo entero al programa principal).
38: *}
39: function selectMenuOption(var options:MenuLabels):integer;
40:
41: END.
42:
43: MODULE menu IMPLEMENTATION;
44:
45: IMPORT
46:
47:     standardoutput;
48:     display;
49:     filesystem;
50:     filteredinput;
51:     printlist;
52:     sort;

```

```

53:
54: function drawMenu;
55: var
56:
57:     i:char value 'a';
58:
59: begin
60:
61:     clearDisplay;
62:
63:     setTitle('Menú principal');
64:
65:     for i := 'a' to LAST_ITEM do
66:     begin
67:
68:         write(i, ' ', options[i]);
69:         downLines(1);
70:
71:     end;
72:
73:     downLines(1);
74:     moveMargin(-4);
75:     write('Introduzca la letra correspondiente a la opción del menú y pulse intro: ');
76:
77:     if m > 0 then
78:
79:         case m of
80:
81:             1: setError('Opción inválida. ');
82:             2: setError('Opción inválida. Comprueba el estado del bloqueo de mayúsculas en el teclado. ');
83:             3: setMssg('Los datos del nuevo alumno han sido guardados correctamente. ');
84:             4: setError('No ha sido posible guardar los datos del nuevo alumno en el fichero ./data');
85:             5: setMssg('Datos guardados correctamente. ');
86:             6: setMssg('No se ha modificado nada en la base de datos. ');
87:             7: setMssg('La base de datos no contiene alumnos. ');
88:
89:             otherwise;
90:
91:         end;
92:
93:     ;
94:
95:     drawMenu := selectMenuOption(options);
96:
97: end;
98:
99: function selectMenuOption;
100: var
101:
102:     items:MenuItems value ['a'..LAST_ITEM];
103:     dataDump:PtrToDump;
104:     option:char;
105:     optionValue:integer;
106:

```

```

107: begin
108:
109:     readChar(option);
110:
111:     if option in items then
112:         begin
113:
114:             case option of
115:
116:                 'a': selectMenuOption := newStudent;
117:                 'b': selectMenuOption := editStudent;
118:                 'c':
119:                     begin
120:
121:                         newDump(dataDump);
122:
123:                         if dataDump = NIL then selectMenuOption := 7
124:                         else
125:                             begin
126:
127:                                 bubbleSortAlphabetical(dataDump);
128:
129:                                 printOrdered(dataDump, -1);
130:
131:                                 disposeDump(dataDump);
132:
133:                                 selectMenuOption := 255;
134:
135:                             end;
136:
137:                         end;
138:                 'd':
139:                     begin
140:
141:                         clearDisplay;
142:                         setTitle('Elegir convocatoria');
143:
144:                         setMssg('Deja en blanco el campo y pulsa intro para volver al menú principal.');

```

```

166:
167:         otherwise optionValue := -1;
168:
169:     end;
170:
171:     if optionValue > -1 then
172:     begin
173:
174:         newDump(dataDump);
175:
176:         if dataDump = NIL then selectMenuOption := 7
177:         else
178:         begin
179:
180:             bubbleSortAlphabetical(dataDump);
181:
182:             printOrdered(dataDump, optionValue);
183:
184:             disposeDump(dataDump);
185:
186:             selectMenuOption := 255;
187:
188:         end;
189:
190:     end
191:     else selectMenuOption := 255;
192:
193: end;
194: 'e':
195: begin
196:
197:     clearDisplay;
198:     setTitle('Elegir convocatoria');
199:
200:     setMssg('Deja en blanco el campo y pulsa intro para volver al menú principal.');

```



```

225:         end;
226:
227:         if optionValue > -1 then
228:             begin
229:
230:                 newDump(dataDump);
231:
232:                 if dataDump = NIL then selectMenuOption := 7
233:                 else
234:                     begin
235:
236:                         bubbleSortByMarks(dataDump, optionValue);
237:
238:                         printOrdered(dataDump, optionValue);
239:
240:                         disposeDump(dataDump);
241:
242:                         selectMenuOption := 255;
243:
244:                     end;
245:
246:                 end
247:                 else selectMenuOption := 255;
248:
249:             end;
250:             'f': selectMenuOption := 0;
251:
252:         end;
253:
254:     end
255:     else if (ord(option) > 64) and (ord(option) < 91) then selectMenuOption := 2
256:     else selectMenuOption := 1;
257:     ;
258:
259: end;
260:
261: END.

```

```

1: MODULE display INTERFACE;
2:
3: {*
4:     Este módulo proporciona distintos procedimientos relacionados
5:     con el control de la terminal. Dichos procedimientos permiten
6:     borrar la pantalla, mover el cursor por la misma y escribir
7:     con diferentes formatos (colores y texto en negrita).
8:
9:     Este módulo ha sido probado con éxito en gnome-terminal (objetivo
10:    de la práctica), xterm y la terminal inicial de Linux (modo sin
11:    iniciar el servidor X, no se como denominarlo).
12:
13:    NOTA: Para crear este módulo empecé por buscar información
14:    acerca de imprimir la secuencia chr(27) + '[2J' para borrar la
15:    pantalla, facilitada por el profesor de la asignatura en clase.
16:    A partir de ahí llegué a métodos similares para borrar la pantalla
17:    en C/C++ y otros lenguajes. Posteriormente encontré un módulo
18:    de ejemplo que circula entre los alumnos de FIC y que
19:    incluyo como ./doc/ansi.pas (parece que está escrito en
20:    Free Pascal), el cual me dio las claves para seguir buscando,
21:    hasta que encontré en diversas páginas de Internet
22:    la especificación del estándar ANSI X3.64 (o ECMA-48).
23: *}
24:
25: export display = (    BOLD, clear, clearDisplay, clearDown, clearLine, downLines, goToXY,
26:                      memoLine, memoXY, moveHorizontally, moveMargin, returnToLine,
27:                      returnToXY, REVERSE, resetFormat, setError, setFormat, setMssg, setTitle);
28:
29: CONST
30:
31:     {Inicia las secuencias de escape ANSI.}
32:     ESC                = chr(27) + '[';
33:     {Códigos para el formato del texto.}
34:     BOLD                = 1;
35:     FOREGROUND_BLUE    = 34;
36:     FOREGROUND_RED     = 31;
37:     REVERSE             = 7;
38:     {Márgenes y decoración en el título.}
39:     MARGIN_LEFT        = 10;
40:     MARGIN_UP          = 4;
41:     TITLE_LEFT_DECORATION = '|';
42:     TITLE_RIGHT_DECORATION = '| |';
43:
44: {*
45:     Limpia la pantalla. Pretende ser un análogo
46:     al clear de Linux.
47: *}
48: procedure clear;
49:
50: {*
51:     "Inicia" la pantalla. Resetea los márgenes,
52:     limpia la pantalla y sitúa el cursor
53:     en el lugar inicial.
54: *}
55: procedure clearDisplay;
56:
57: {*
58:     Limpia la pantalla a partir de la línea donde
59:     está el cursor, hacia abajo.
60: *}

```

```

61: procedure clearDown;
62:
63: {*
64:     Borra el mensaje bajo el título de la aplicación.
65: *}
66: procedure clearMssg;
67:
68: {*
69:     Sitúa el cursor al inicio de la línea en la que se encuentre
70:     (respetando el margen) y la limpia.
71: *}
72: procedure clearLine;
73:
74: {*
75:     Mueve la "línea de escritura" lines líneas, hacia
76:     abajo si lines es positivo y hacia arriba si es negativo.
77:     Sitúa el cursor al comienzo de dicha línea (respetando el margen).
78: *}
79: procedure downLines(lines:integer);
80:
81: {*
82:     Mueve el cursor a la línea y, columna x.
83: *}
84: procedure goToXY(x, y:integer);
85:
86: {*
87:     Memoriza la "línea de escritura".
88: *}
89: procedure memoLine;
90:
91: {*
92:     Memoriza la posición del cursor.
93: *}
94: procedure memoXY;
95:
96: {*
97:     Mueve el cursor n columnas, hacia la
98:     derecha si n es positivo o hacia
99:     la izquierda si es negativo.
100: *}
101: procedure moveHorizontally(n:integer);
102:
103: {*
104:     Mueve el margen izquierdo de la aplicación n columnas,
105:     hacia la derecha si n es positivo o hacia la izquierda
106:     si es negativo, y sitúa el cursor al principio
107:     de la línea (respetando dicho margen).
108: *}
109: procedure moveMargin(n:integer);
110:
111: {*
112:     Establece "línea de escritura" en la línea
113:     memorizada por memoLine y situa el
114:     cursor al inicio de la misma (respetando
115:     el margen).
116: *}
117: procedure returnToLine;
118:
119: {*
120:     Sitúa el cursor en la posición almacenada

```

```

121:      por memoXY.
122: *}
123: procedure returnToXY;
124:
125: {*
126:      Establece el formato del texto a su valor por defecto.
127: *}
128: procedure resetFormat;
129:
130: {*
131:      Hace sonar la campana del sistema (de estar esta activada) e
132:      imprime el mensaje error bajo el título de la aplicación.
133: *}
134: procedure setError(error:string);
135:
136: {*
137:      Establece el formato del texto enviado a la terminal
138:      a partir de la llamada a este procedimiento. Format es
139:      un entero entre 1 y 47 (incluidos). Más información
140:      en la especificación del estándar ANSI X3.64
141: *}
142: procedure setFormat(format:integer);
143:
144: {*
145:      Imprime el mensaje m bajo el título de la aplicación.
146: *}
147: procedure setMssg(m:string);
148:
149: {*
150:      Establece title como título de la aplicación, rodeado
151:      por la decoración indicada por las constantes
152:      TITLE_LEFT_DECORATION y TITLE_RIGHT_DECORATION.
153: *}
154: procedure setTitle(title:string);
155:
156: END.
157:
158:
159: MODULE display IMPLEMENTATION;
160:
161: IMPORT
162:
163:      standardoutput;
164:
165: VAR
166:
167:      lineToWriteIn:integer value MARGIN_UP;
168:      marginLeft:integer value MARGIN_LEFT;
169:      memo:integer value 1;
170:
171: procedure clear;
172: begin
173:
174:      write(chr(1), ESC, 'H', ESC, '2J');
175:
176: end;
177:
178: procedure clearDisplay;
179: begin
180:

```

```

181:     marginLeft := MARGIN_LEFT;
182:     lineToWriteIn := MARGIN_UP;
183:
184:     clear;
185:
186:     goToXY(MARGIN_LEFT, MARGIN_UP);
187:
188: end;
189:
190: procedure clearDown;
191: begin
192:
193:     write(ESC, 'J');
194:
195: end;
196:
197: procedure clearLine;
198: begin
199:
200:     goToXY(marginLeft, lineToWriteIn);
201:
202:     write(ESC, 'K');
203:
204: end;
205:
206: procedure clearMssg;
207: var
208:
209:     i:integer;
210:
211: begin
212:
213:     memoXY;
214:
215:     for i := 2 to MARGIN_UP - 1 do
216:     begin
217:
218:         goToXY(1, i);
219:         write(ESC, 'K');
220:
221:     end;
222:
223:     returnToXY;
224:
225: end;
226:
227: procedure downLines;
228: begin
229:
230:     lineToWriteIn := lineToWriteIn + lines;
231:     writeln;
232:     goToXY(marginLeft, lineToWriteIn);
233:
234: end;
235:
236: procedure goToXY;
237: begin
238:
239:     write(ESC, y:0, ';', x:0, 'H');
240:

```

```

241: end;
242:
243: procedure memoLine;
244: begin
245:
246:     memo := lineToWriteIn;
247:
248: end;
249:
250: procedure memoXY;
251: begin
252:
253:     write(ESC, 's');
254:
255: end;
256:
257: procedure moveHorizontally;
258: begin
259:
260:     if n > 0 then write(ESC, n:0, 'C')
261:     else write(ESC, abs(n):0, 'D');
262:
263: end;
264:
265: procedure moveMargin;
266: begin
267:
268:     marginLeft := marginLeft + n;
269:     if marginLeft < 1 then marginLeft := 1;
270:
271:     goToXY(marginLeft, lineToWriteIn);
272:
273: end;
274:
275: procedure returnToLine;
276: begin
277:
278:     lineToWriteIn := memo;
279:     goToXY(marginLeft, lineToWriteIn);
280:
281: end;
282:
283: procedure returnToXY;
284: begin
285:
286:     write(ESC, 'u');
287:
288: end;
289:
290: procedure resetFormat;
291: begin
292:
293:     write(ESC, 0:0, 'm');
294:
295: end;
296:
297: procedure setError;
298: begin
299:
300:     clearMssg;

```

```

301:     memoXY;
302:     goToXY(1, 2);
303:     setFormat(FOREGROUND_RED);
304:     write(chr(7), 'Error: ', error);
305:     resetFormat;
306:     returnToXY;
307:
308: end;
309:
310: procedure setFormat;
311: begin
312:
313:     if (format > 0) and (format < 48) then write(ESC, format:0, 'm');
314:
315: end;
316:
317: procedure setMssg;
318: begin
319:
320:     clearMssg;
321:     memoXY;
322:     goToXY(1, 2);
323:     setFormat(FOREGROUND_BLUE);
324:     write(m);
325:     resetFormat;
326:     returnToXY;
327:
328: end;
329:
330: procedure setTitle;
331: begin
332:
333:     memoXY;
334:     goToXY(1, 1);
335:     write(TITLE_LEFT_DECORATION, title, TITLE_RIGHT_DECORATION);
336:     returnToXY;
337:
338: end;
339:
340: to end do writeln;
341:
342: END.

```

```

1: MODULE filteredinput INTERFACE;
2:
3: {*
4:     Este módulo proporciona funciones y procedimientos para
5:     obtener datos del usuario.
6:
7:     Todos los procedimientos imprimen sus propios formularios, y
8:     filtran la entrada de los datos para evitar errores de formato
9:     y contenido, hablando tanto en términos informáticos como en lo
10:    relacionado con la utilidad de la aplicación.
11: *}
12:
13: export filteredinput = (editStudent, MONTHS, newStudent, readChar, YES, YES_NO);
14:
15: IMPORT
16:
17:     filesystem;
18:
19: CONST
20:
21:     MONTHS = ['f', 'j', 's', 'd', 'F', 'J', 'S', 'D'];
22:     YES     = ['s', 'S'];
23:     YES_NO  = ['s', 'n', 'S', 'N'];
24:
25: {*
26:     Maneja la entrada de datos para editar un alumno. Permite
27:     buscarlo y cambiar todos los datos del mismo. El valor devuelto
28:     será 5 si el alumno ha sido editado correctamente, 6 si no
29:     se ha cargado ningún alumno o 7 si la base de datos está
30:     vacía.
31: *}
32: function editStudent:integer;
33:
34: {*
35:     Pide el login del alumno s^. search indica si se está haciendo
36:     una búsqueda (true) o no (false), y notNull indica si se permite
37:     dejar el login en blanco (false, por ejemplo al editar un alumno,
38:     pues dejar los campos en blanco significa no modificar su valor
39:     anterior) o no (true).
40: *}
41: procedure getStudentLogin(s:PtrToStudent; search, notNull:boolean);
42:
43: {*
44:     Pide la nota de la convocatoria option para el alumno
45:     s^. t indica si es la nota teórica (true) o la práctica (false).
46:     Para que sea válido, el valor de la nota debe estar entre 0 y
47:     10 (incluidos) y tener un decimal como máximo.
48: *}
49: procedure getStudentMark(s:PtrToStudent; option:char; t:boolean);
50:
51: {*
52:     Llama a getStudentMark para pedir las notas teórica y
53:     práctica de la convocatoria option para el alumno s^.
54: *}
55: procedure getStudentMarks(s:PtrToStudent; option:char);
56:
57: {*
58:     Pregunta al usuario la convocatoria para la cual
59:     se pedirán posteriormente las notas. s es un puntero
60:     al alumno en cuestión.

```



```

61: *}
62: procedure getStudentMarksQC(s:PtrToStudent);
63:
64: {*
65:     Pregunta al usuario si desea introducir las notas para el
66:     alumno s^. first indica si es la primera vez que se
67:     hace la pregunta (true) o no (false). La función devuelve
68:     true si el usuario ha decidido introducirlas, o false
69:     en el caso contrario.
70: *}
71: function getStudentMarksQYN(s:PtrToStudent; first:boolean):boolean;
72:
73: {*
74:     Pide el nombre para el alumno s^.
75: *}
76: procedure getStudentName(s:PtrToStudent);
77:
78: {*
79:     Pide los apellidos para el alumno s^.
80: *}
81: procedure getStudentSurname(s:PtrToStudent);
82:
83: {*
84:     Maneja la entrada de datos para crear un nuevo alumno. Todos
85:     los datos son opcionales menos el login, el cual además
86:     no puede ser el mismo para más de un alumno. La función
87:     devuelve 3 si se ha podido almacenar el nuevo alumno
88:     en la base de datos o 4 en caso contrario.
89: *}
90: function newStudent:integer;
91:
92: {*
93:     Lee un carácter de la entrada estándar, lo
94:     almacena en c y descarta los posibles
95:     caracteres que queden en el buffer.
96: *}
97: procedure readChar(var c:char);
98:
99: {*
100:    Lee una cadena de texto, comprueba que su
101:    formato sea el adecuado para ser un login
102:    y lo almacena en l. search indica si debe
103:    evitar aceptar un login ya existente (false)
104:    o no (true). La función devuelve 1 si el usuario
105:    no ha introducido ningún carácter, 2 si el texto
106:    introducido no tiene 8 caracteres (contando cada
107:    byte como un carácter), 3 si no tiene el formato
108:    correcto, 4 si search es false y ya existe un alumno
109:    con el mismo login o 0 si todo es correcto.
110: *}
111: function readLogin(var l:string; search:boolean):integer;
112:
113: {*
114:    Lee de la entrada estándar hacia r, comprobando
115:    que el texto introducido es un real. Para que sea válido
116:    y r se llene con el valor introducido por el usuario el
117:    mismo debe ser un real, estar comprendido entre max
118:    y min (incluidos) y no tener más de dMax cifras decimales.
119:    Esta función devolverá 0 si todo se ha llevado a cabo
120:    correctamente, 1 si el texto introducido no es un real,

```

```

121:      2 si es mayor que max, 3 si es menor que min o 4
122:      si tiene más cifras decimales que dMax.
123: *}
124: function readReal(var r:real; max, min, dMax:integer):integer;
125:
126: END.
127:
128:
129: MODULE filteredinput IMPLEMENTATION;
130:
131: IMPORT
132:
133:     standardinput;
134:     standardoutput;
135:     display;
136:
137: CONST
138:
139:     LETTERS      = ['a'..'z'];
140:     MARK_MAX     = 10;
141:     MARK_MAX_D   = 1;
142:     MARK_MIN     = 0;
143:     MINUS        = '-';
144:     NINE         = ord('9');
145:     NUMBERS      = ['0'..'9'];
146:     POINT        = '.';
147:     PLUS         = '+';
148:     ZERO        = ord('0');
149:
150: function editStudent;
151: var
152:
153:     b:boolean;
154:     login:string(255);
155:     result:integer;
156:     s:PtrToStudent;
157:
158: begin
159:
160:     {*
161:      data es global, proporcionada por el módulo filesystem.
162:     *}
163:
164:     reset(data^);
165:
166:     if lastPosition(data^) = -1 then
167:     begin
168:
169:         editStudent := 7;
170:
171:     end
172:     else
173:     begin
174:
175:         clearDisplay;
176:         setTitle('Editar alumno');
177:
178:         setMssg('Deja en blanco el campo y pulsa intro para volver al menú principal.');
```

```

181:
182:     result := readLogin(login, true);
183:     if result = 0 then
184:     begin
185:
186:         s := searchStudent(login);
187:         if s = NIL then result := 5;
188:
189:     end;
190:
191:     while result > 1 do
192:     begin
193:
194:         case result of
195:
196:             2: setError('El login del alumno debe tener exactamente 8 caracteres.');
```

2: setError('El login del alumno debe tener exactamente 8 caracteres.');

3: setError('El formato de "login" debe ser aaaaaann, siendo cada "a" una letra minúscula del alfabeto inglés y cada "n" un número.');

5: setError('No existe ningún alumno en la base de datos que tenga ese login.');

```

197:
198:             5: setError('No existe ningún alumno en la base de datos que tenga ese login.');
```

5: setError('No existe ningún alumno en la base de datos que tenga ese login.');

```

199:
200:         end;
201:
202:         clearLine;
203:         write('Login: ');
204:
205:         result := readLogin(login, true);
206:
207:         if result = 0 then
208:         begin
209:
210:             s := searchStudent(login);
211:             if s = NIL then result := 5;
212:
213:         end;
214:
215:     end;
216:
217:     if result = 0 then
218:     begin
219:
220:         downLines(1);
221:
222:         setMssg('Dejar un campo en blanco no modifica el valor anterior del mismo.');
```

setMssg('Dejar un campo en blanco no modifica el valor anterior del mismo.');

```

223:
224:         write('Nombre actual: ', s^.name);
225:         downLines(1);
226:         getStudentName(s);
227:         downLines(1);
228:
229:         write('Apellidos actuales: ', s^.surname);
230:         downLines(1);
231:         getStudentSurname(s);
232:         downLines(1);
233:
234:         write('Login actual: ', s^.login);
235:         downLines(1);
236:         getStudentLogin(s, true, false);
237:         downLines(1);
238:

```

```

239:         memoLine;
240:         b := getStudentMarksQYN(s, true);
241:         while b do
242:             begin
243:
244:                 returnToLine;
245:                 clearDown;
246:                 b := getStudentMarksQYN(s, false);
247:
248:             end;
249:
250:
251:             {*
252:                 beingEdited es global, proporcionada por filesystem.
253:             *}
254:             seekUpdate(data^, beingEdited);
255:             data^^ := s^;
256:             update(data^);
257:
258:             dispose(s);
259:
260:             editStudent := 5;
261:
262:         end
263:     else editStudent := 6;
264:
265: end;
266:
267: end;
268:
269: procedure getStudentLogin;
270: var
271:
272:     login:string(255);
273:     option, result:integer;
274:
275: begin
276:
277:     write('Login: ');
278:
279:     if notNull then option := 0
280:     else option := 1;
281:
282:     result := readLogin(login, search);
283:     while result > option do
284:         begin
285:
286:             case result of
287:
288:                 1: setError('El login no puede dejarse en blanco.');
```

```

298:
299:         result := readLogin(login, search);
300:
301:     end;
302:
303:     if result = 0 then s^.login := login;
304:
305: end;
306:
307: procedure getStudentMark;
308: var
309:
310:     i, savedIterator: integer value 0;
311:     mark: real value 0;
312:     n, result: integer;
313:
314: begin
315:
316:     case option of
317:
318:         'f', 'j': n := 0;
319:         's': n := 1;
320:         'd': n := 2;
321:
322:     end;
323:
324:     if t then
325:     begin
326:
327:         {*
328:         config es global, proporcionada por filesystem.
329:         *}
330:
331:         if config[0] then
332:         begin
333:
334:             for i := 0 to 2 do
335:             begin
336:
337:                 if mark < s^.marks[i].theory then
338:                 begin
339:
340:                     savedIterator := i;
341:                     mark := s^.marks[i].theory;
342:
343:                 end;
344:
345:             end;
346:
347:             if mark >= 5 then
348:             begin
349:
350:                 write('El alumno ya tiene aprobada la parte teórica: ', mark:0:1);
351:
352:                 case savedIterator of
353:
354:                     0: write('f/j');
355:                     1: write('s');
356:                     2: write('d');
357:

```

```

358:         end;
359:
360:         downLines(1);
361:
362:     end;
363:
364: end;
365:
366: write('Nota teórica: ');
367:
368: end
369: else
370: begin
371:
372:     if config[1] then
373:     begin
374:
375:         mark := 0;
376:
377:         for i := 0 to 2 do
378:         begin
379:
380:             if mark < s^.marks[i].practice then
381:             begin
382:
383:                 savedIterator := i;
384:                 mark := s^.marks[i].practice;
385:
386:             end;
387:
388:         end;
389:
390:         if mark >= 5 then
391:         begin
392:
393:             write('El alumno ya tiene aprobada la parte práctica: ', mark:0:1);
394:
395:             case savedIterator of
396:
397:                 0: write('f/');
398:                 1: write('s');
399:                 2: write('d');
400:
401:             end;
402:
403:             downLines(1);
404:
405:         end;
406:
407:     end;
408:
409:     write('Nota práctica: ');
410:
411: end;
412:
413: result := readReal(mark, MARK_MAX, MARK_MIN, MARK_MAX_D);
414:
415: while result > 0 do
416: begin
417:

```

```

418:         case result of
419:
420:             1: setError('Introduce un número entre 0 y 10, con un decimal como máximo.');
```

- 421: 2: setError('El número introducido no puede ser mayor que 10.');
- 422: 3: setError('El número introducido no puede ser menor que cero.');
- 423: 4: setError('El número introducido no puede tener más de una cifra decimal.');

```

424:
425:         end;
426:
427:         clearLine;
428:
429:         if t then write('Nota teórica: ');
430:         else write('Nota práctica: ');
431:
432:         result := readReal(mark, MARK_MAX, MARK_MIN, MARK_MAX_D);
433:
434:     end;
435:
436:     if t then s^.marks[n].theory := mark
437:     else s^.marks[n].practice := mark;
438:
439: end;
440:
441: procedure getStudentMarks;
442: begin
443:
444:     downLines(1);
445:     getStudentMark(s, option, false);
446:     downLines(1);
447:     getStudentMark(s, option, true);
448:
449: end;
450:
451: procedure getStudentMarksQC;
452: var
453:
454:     option:char;
455:
456: begin
457:
458:     downLines(1);
459:     write('Convocatoria [f/j/s/d]: ');
460:     readChar(option);
461:
462:     while not (option in MONTHS) do
463:     begin
464:
465:         setError('Solo se admiten como valores válidos la inicial de los meses Febrero, Junio, Septiembre y
            Diciembre.');
```

- 466: clearLine;
- 467: write('Convocatoria [f/j/s/d]: ');
- 468: readChar(option);

```

469:
470:     end;
471:
472:     getStudentMarks(s, option);
473:
474: end;
475:
476: function getStudentMarksQYN;
```

```

477: var
478:
479:     i, savedIterator:integer;
480:     mark:real value 0;
481:     option:char;
482:
483: begin
484:
485:     if config[0] then
486:     begin
487:
488:         for i := 0 to 2 do
489:         begin
490:
491:             if mark < s^.marks[i].theory then
492:             begin
493:
494:                 savedIterator := i;
495:                 mark := s^.marks[i].theory;
496:
497:             end;
498:
499:         end;
500:
501:         if mark >= 5 then
502:         begin
503:
504:             case savedIterator of
505:
506:                 0: write('El alumno aprobó la parte teórica en Febrero/Junio con un ', mark:0:1);
507:                 1: write('El alumno aprobó la parte teórica en Septiembre con un ', mark:0:1);
508:                 2: write('El alumno aprobó la parte teórica en Diciembre con un ', mark:0:1);
509:
510:             end;
511:
512:             downLines(1);
513:
514:         end;
515:
516:     end;
517:
518:     if config[1] then
519:     begin
520:
521:         mark := 0;
522:
523:         for i := 0 to 2 do
524:         begin
525:
526:             if mark < s^.marks[i].practice then
527:             begin
528:
529:                 savedIterator := i;
530:                 mark := s^.marks[i].practice;
531:
532:             end;
533:
534:         end;
535:
536:         if mark >= 5 then

```



```

537:         begin
538:
539:             case savedIterator of
540:
541:                 0: write('El alumno aprobó la parte práctica en Febrero/Junio con un ', mark:0:1);
542:                 1: write('El alumno aprobó la parte práctica en Septiembre con un ', mark:0:1);
543:                 2: write('El alumno aprobó la parte práctica en Diciembre con un ', mark:0:1);
544:
545:             end;
546:
547:             downLines(1);
548:
549:         end;
550:
551:     end;
552:
553:     if first then write('¿Introducir notas del alumno? [s/n]: ')
554:     else write('¿Seguir introduciendo notas? [s/n]: ');
555:
556:     readChar(option);
557:
558:     while not (option in YES_NO) do
559:         begin
560:
561:             setError('Solo se admiten los valores "s" o "n", para responder "Sí" o "No" a la pregunta,
                    respectivamente.');
```

```

562:             clearLine;
563:             write('¿Introducir notas del alumno? [s/n]: ');
564:             readChar(option);
565:
566:         end;
567:
568:         if option in YES then
569:             begin
570:
571:                 getStudentMarksQC(s);
572:                 getStudentMarksQYN := true;
573:
574:             end
575:         else getStudentMarksQYN := false;
576:
577:     end;
578:
579:     procedure getStudentName;
580:     var
581:
582:         temp:string(255);
583:
584:     begin
585:
586:         write('Nombre: ');
587:         readln(temp);
588:
589:         {
590:             NAME_CAPACITY es una constante proporcionada por filesystem.
591:         }
592:         if length(temp) > NAME_CAPACITY then
593:             begin
594:
595:                 setError('El nombre del alumno no puede superar los 20 caracteres.');
```

```

596:         clearLine;
597:         getStudentName(s);
598:
599:     end
600:     else if temp <> " then s^.name := temp;
601:     ;
602:
603: end;
604:
605: procedure getStudentSurname;
606: var
607:
608:     temp:string(255);
609:
610: begin
611:
612:     write('Apellidos: ');
613:     readln(temp);
614:
615:     {*
616:         SURNAME_CAPACITY es una constante proporcionada por filesystem.
617:     *}
618:     if length(temp) > SURNAME_CAPACITY then
619:     begin
620:
621:         setError('Los apellidos del alumno no pueden superar los 40 caracteres. ');
622:         clearLine;
623:         getStudentSurname(s);
624:
625:     end
626:     else if temp <> " then s^.surname := temp;
627:     ;
628:
629: end;
630:
631: function newStudent;
632: var
633:
634:     b:boolean;
635:     s:PtrToStudent;
636:
637: begin
638:
639:     clearDisplay;
640:     setTitle('Nuevo alumno');
641:     setMssg("Login" es el único dato obligatorio.);
642:
643:     new(s);
644:
645:     {*
646:         Inicializamos todos los campos de s^,
647:         pues podrían contener basura.
648:     *}
649:     s^.name := "";
650:     s^.surname := "";
651:     s^.login := "";
652:     s^.marks[0].theory := 0;
653:     s^.marks[0].practice := 0;
654:     s^.marks[1].theory := 0;
655:     s^.marks[1].practice := 0;

```

```

656:   s^.marks[2].theory := 0;
657:   s^.marks[2].practice := 0;
658:
659:   getStudentName(s);
660:   downLines(1);
661:
662:   getStudentSurname(s);
663:   downLines(1);
664:
665:   getStudentLogin(s, false, true);
666:   downLines(1);
667:
668:   b := getStudentMarksQYN(s, true);
669:   while b do
670:   begin
671:
672:       downLines(-3);
673:       clearDown;
674:       b := getStudentMarksQYN(s, false);
675:
676:   end;
677:
678:   if data = NIL then newStudent := 4
679:   else
680:   begin
681:
682:       extend(data^);
683:       write(data^, s^);
684:
685:       newStudent := 3;
686:
687:   end;
688:
689:   dispose(s);
690:
691: end;
692:
693: procedure readChar;
694: begin
695:
696:   c := input^;
697:   reset(input);
698:
699: end;
700:
701: function readLogin;
702: var
703:
704:   cLetters, cNums: integer value 0;
705:   i: integer;
706:   temp: string(255);
707:
708: begin
709:
710:   write('[   ]');
711:   moveHorizontally(-9);
712:
713:   readln(temp);
714:
715:   if temp = " then readLogin := 1

```

```

716:     else if length(temp) <> 8 then readLogin := 2
717:     else
718:     begin
719:
720:         for i := 1 to 6 do if temp[i] in LETTERS then cLetters := cLetters + 1;
721:         for i := 7 to 8 do if temp[i] in NUMBERS then cNums := cNums + 1;
722:
723:         if (cLetters = 6) and (cNums = 2) then
724:         begin
725:
726:             if not search and studentExists(temp) then readLogin := 4
727:             else
728:             begin
729:
730:                 l := temp;
731:                 readLogin := 0;
732:
733:             end;
734:
735:         end
736:     else readLogin := 3;
737:
738: end;
739: ;
740:
741: end;
742:
743: {*
744:     Esta función es una extensión de otra creada a partir de una
745:     sugerencia hecha en clase por el profesor, la cual en resumidas
746:     cuentas era que creásemos una función que permitiese
747:     leer de la entrada estándar hacia un real, pero evitando
748:     que se produjese un error de detención si lo introducido por
749:     el usuario no era realmente un real. La función original, con
750:     las explicaciones teóricas oportunas para comprender su
751:     funcionamiento, está incluida en ./doc/readReal.pas.
752: *}
753: function readReal;
754: var
755:
756:     c, dCount:integer value 0;
757:     userInput:text;
758:     invalid, decimal:boolean value false;
759:
760: begin
761:
762:     rewrite(userInput);
763:
764:     while not eoln and not invalid do
765:     begin
766:
767:         c := c + 1;
768:         if decimal then dCount := dCount + 1;
769:
770:         read(input, userInput^);
771:
772:         {*
773:             Números del 0 al 9.
774:         *}
775:         if ((ord(userInput^) >= ZERO) and (ord(userInput^) <= NINE)) then put(userInput)

```

```

776:      {*}
777:      El punto, siempre y cuando no haya sido introducido anteriormente y
778:      tampoco estemos hablando de que es el primer caracter.
779:      *}
780:      else if (userInput^ = POINT) and not decimal and (c <> 1) then
781:      begin
782:
783:          put(userInput);
784:          decimal := true;
785:
786:      end
787:      {*}
788:      El signo, positivo o negativo, y pudiendo ocupar únicamente
789:      la primera posición.
790:      *}
791:      else if ((userInput^ = PLUS) or (userInput^ = MINUS)) and (c = 1) then put(userInput)
792:      else invalid := true;
793:
794:
795:
796:  end;
797:
798:  reset(input);
799:  reset(userInput);
800:
801:  if invalid then readReal := 1
802:  else if eof(userInput) then
803:  begin
804:
805:      readReal := 0;
806:
807:      if r = 0 then setMssg('Nota definida a 0 (valor por defecto).')
808:      else setMssg('La nota no ha sido modificada.');

```

```

1: MODULE filesystem INTERFACE;
2:
3: {*
4:     Este módulo proporciona los tipos, procedimientos y funciones
5:     relacionados con los datos de la aplicación (Persistencia en
6:     disco de los datos y tratamiento de los mismos,
7:     carga de configuraciones y tipos personalizados).
8: *}
9:
10: export filesystem = ( beingEdited, config, data, disposeDump, NAME_CAPACITY,
11:                      SURNAME_CAPACITY, newDump, PtrToDump, PtrToStudent,
12:                      searchStudent, studentExists);
13:
14: CONST
15:
16:     NAME_CAPACITY      = 20;
17:     SURNAME_CAPACITY   = 40;
18:
19: TYPE
20:
21:     BooleanStream = bindable file of boolean;
22:     PtrToBooleanStream = ^BooleanStream;
23:
24:     {*
25:         Tipo registro para las notas teórica y práctica,
26:         la global se calcula al mostrarla.
27:     *}
28:     CallMarks = record
29:
30:         theory:real;
31:         practice:real;
32:
33:     end;
34:
35:     {*
36:         Tipo del alumno.
37:     *}
38:     Student = record
39:
40:         name:string(NAME_CAPACITY);
41:         surname:string(SURNAME_CAPACITY);
42:         login:packed array[1..8] of char;
43:         marks:array[0..2] of CallMarks;
44:
45:     end;
46:     PtrToStudent = ^Student;
47:
48:     {*
49:         Secuencia de alumnos. Se utiliza de 0 a MAXINT puesto que al
50:         parecer no existe ninguna diferencia en cuanto a memoria utilizada
51:         por las secuencias de acceso secuencial y las de acceso aleatorio.
52:     *}
53:     StudentStream = bindable file[0..MAXINT] of Student;
54:     PtrToStudentStream = ^StudentStream;
55:

```

```

56:  {*
57:      Variable para el volcado de la secuencia de alumnos
58:      a un array de puntero-a-alumno. Se utiliza un
59:      array pues no es recomendable hacer la ordenación
60:      en disco, y se utilizan punteros puesto que en
61:      la ordenación lo que se moverá serán los punteros y no
62:      la información a la que apuntan, moviendo así menos datos
63:      en memoria.
64:  */
65:  Dump(eod:integer) = array[0..eod] of PtrToStudent;
66:  PtrToDump = ^Dump;
67:
68:  VAR
69:
70:      {El último alumno encontrado por searchStudent.}
71:      beingEdited:integer;
72:      {Almacena la configuración de la aplicación.}
73:      config:array[0..1] of boolean;
74:      {Puntero a la secuencia que actúa como base de datos.}
75:      data:PtrToStudentStream;
76:
77:  {*
78:      Crea una secuencia de booleans y la liga con el fichero name
79:      (en disco, si el fichero no existe lo crea). La función devuelve
80:      un puntero a la secuencia creada o un puntero nulo si ha ocurrido
81:      algún error.
82:  */
83:  function booleanStreamOpen(name:string):PtrToBooleanStream;
84:
85:  {*
86:      Desliga la secuencia f^ de su fichero en disco y libera
87:      el espacio de memoria asignado a la misma.
88:  */
89:  procedure booleanStreamClose(f:PtrToBooleanStream);
90:
91:  {*
92:      Libera el espacio de memoria ocupado por el volcado link^.
93:  */
94:  procedure disposeDump(link:PtrToDump);
95:
96:  {*
97:      Carga la configuración de la aplicación o la crea si no existe.
98:  */
99:  procedure loadConfig;
100:
101:  {*
102:      Crea un volcado de la secuencia data^ (data es global) en link^.
103:  */
104:  procedure newDump(var link:PtrToDump);
105:
106:  {*
107:      Busca un alumno cuyo login coincida con login y si existe devuelve
108:      un puntero al mismo y modifica la variable beingEdited (global) con
109:      la posición de ese alumno en la base de datos. Si no existe devuelve
110:      un puntero nulo.
111:  */
112:  function searchStudent(login:string):PtrToStudent;
113:

```

```

114: {*}
115:     Busca un alumno cuyo login coincida con login y si existe devuelve
116:     true y modifica la variable aux (global solo en la implementación
117:     del módulo) con la posición de ese alumno en la base de datos.
118:     Si no existe devuelve false.
119: *}
120: function studentExists(login:string):boolean;
121:
122: {*}
123:     Crea una secuencia de alumnos (Student) y la liga con el fichero
124:     name (en disco, si el fichero no existe lo crea). La función devuelve
125:     un puntero a la secuencia creada o un puntero nulo si ha ocurrido
126:     algún error.
127: *}
128: function studentStreamOpen(name:string):PtrToStudentStream;
129:
130: {*}
131:     Desliga la secuencia f^ de su fichero en disco y libera
132:     el espacio de memoria asignado a la misma.
133: *}
134: procedure studentStreamClose(f:PtrToStudentStream);
135:
136: END.
137:
138:
139: MODULE filesystem IMPLEMENTATION;
140:
141: IMPORT
142:
143:     filteredinput;
144:     standardoutput;
145:
146: CONST
147:
148:     DATABASE_FILE_NAME = './data';
149:
150: VAR
151:
152:     {*}
153:         Contendrá el último alumno encontrado por studentExists
154:         (o el último alumno de la base de datos si la última
155:         llamada a studentExists devolvió false).
156:     *}
157:     aux:integer;
158:
159: function booleanStreamOpen;
160: var
161:
162:     b:BindingType;
163:     f:PtrToBooleanStream;
164:
165: begin
166:
167:     if length(name) = 0 then booleanStreamOpen := NIL
168:     else
169:         begin
170:
171:             new(f);
172:
173:             unbind(f^);

```



```

174:      b := binding(f^);
175:      b.name := name;
176:      bind(f^, b);
177:      b := binding(f^);
178:
179:      if b.bound then
180:      begin
181:
182:          if not b.existing then rewrite(f^);
183:
184:          booleanStreamOpen := f
185:
186:      end
187:      else
188:      begin
189:
190:          dispose(f);
191:          booleanStreamOpen := NIL
192:
193:      end;
194:
195:  end;
196:
197: end;
198:
199: procedure booleanStreamClose;
200: begin
201:
202:     unbind(f^);
203:     dispose(f);
204:
205: end;
206:
207: procedure disposeDump;
208: var
209:
210:     i:integer;
211:
212: begin
213:
214:     for i := 0 to link^.eod do dispose(link^[i]);
215:
216:     dispose(link);
217:
218: end;
219:
220: procedure loadConfig;
221: var
222:
223:     i:integer value 0;
224:     f:PtrToBooleanStream;
225:     c:char;
226:
227: begin
228:
229:     f := booleanStreamOpen('./config');
230:
231:     if f = NIL then
232:     begin
233:

```

```

234:     write('Error desconocido al intentar acceder al archivo ./config, ');
235:     writeln('no es posible cargar la configuración de la aplicación. ');
236:     write('Pulse intro para continuar. ');
237:     readChar(c);
238:
239: end
240: else
241: begin
242:
243:     reset(f^);
244:
245:     if eof(f^) then
246:     begin
247:
248:         writeln;
249:         writeln('Configuración inicial del programa. ');
250:         writeln;
251:         writeln('¿Debe mantenerse la nota de teoría entre convocatorias? [s/n] ');
252:
253:         readChar(c);
254:
255:         while not (c in YES_NO) do
256:         begin
257:
258:             writeln('Solo se admiten los valores "s" o "n", para responder "Sí" o "No" a la pregunta,
                respectivamente. ');
259:             writeln('¿Debe mantenerse la nota de teoría entre convocatorias? [s/n] ');
260:             readChar(c);
261:
262:         end;
263:
264:         if c in YES then config[0] := true
265:         else config[0] := false;
266:
267:         writeln('¿Debe mantenerse la nota de la práctica entre convocatorias? [s/n] ');
268:
269:         readChar(c);
270:
271:         while not (c in YES_NO) do
272:         begin
273:
274:             writeln('Solo se admiten los valores "s" o "n", para responder "Sí" o "No" a la pregunta,
                respectivamente. ');
275:             writeln('¿Debe mantenerse la nota de la práctica entre convocatorias? [s/n] ');
276:             readChar(c);
277:
278:         end;
279:
280:         if c in YES then config[1] := true
281:         else config[1] := false;
282:
283:         write(f^, config[0]);
284:         write(f^, config[1]);
285:
286:     end
287: else
288: begin
289:
290:     while (i < 2) and (not eof(f^)) do
291:     begin

```

```

292:
293:         read(f^, config[i]);
294:         i := i + 1;
295:
296:     end;
297:
298: end;
299:
300: booleanStreamClose(f);
301:
302: end;
303:
304: end;
305:
306: procedure newDump;
307: var
308:
309:     i, l:integer;
310:     s:PtrToStudent;
311:
312: begin
313:
314:     reset(data^);
315:
316:     l := lastPosition(data^);
317:
318:     if l = -1 then link := NIL
319:     else
320:     begin
321:
322:         new(link, l);
323:
324:         for i := 0 to l do
325:         begin
326:
327:             new(s);
328:
329:             read(data^, s^);
330:
331:             link^[i] := s;
332:
333:         end;
334:
335:     end;
336:
337: end;
338:
339: function searchStudent;
340: var
341:
342:     exists:boolean;
343:     student:PtrToStudent;
344:
345: begin
346:
347:     exists := studentExists(login);
348:
349:     if exists then
350:     begin
351:

```

```

352:         new(student);
353:
354:         beingEdited := aux;
355:         student^ := data^^;
356:         searchStudent := student;
357:
358:     end
359: else searchStudent := NIL;
360:
361: end;
362:
363: function studentExists;
364: var
365:
366:     found:boolean value false;
367:
368: begin
369:
370:     reset(data^);
371:     aux := 0;
372:
373:     while not found and not eof(data^) do
374:     begin
375:
376:         if data^^.login = login then found := true
377:         else
378:         begin
379:
380:             get(data^);
381:             aux := aux + 1;
382:
383:         end;
384:
385:     end;
386:
387:     if found then studentExists := true
388:     else studentExists := false;
389:
390: end;
391:
392: function studentStreamOpen;
393: var
394:
395:     b:BindingType;
396:     f:PtrToStudentStream;
397:
398: begin
399:
400:     if length(name) = 0 then studentStreamOpen := NIL
401:     else
402:     begin
403:
404:         new(f);
405:
406:         unbind(f^);
407:         b := binding(f^);
408:         b.name := name;
409:         bind(f^, b);
410:         b := binding(f^);
411:

```

```

412:         if b.bound then
413:         begin
414:
415:             if not b.existing then rewrite(f^);
416:
417:             studentStreamOpen := f;
418:
419:         end
420:         else
421:         begin
422:
423:             dispose(f);
424:             studentStreamOpen := NIL
425:
426:         end;
427:
428:     end;
429:
430: end;
431:
432: procedure studentStreamClose;
433: begin
434:
435:     unbind(f^);
436:     dispose(f);
437:
438: end;
439:
440: {*
441:     Al cargar por primera vez este módulo se carga la configuración
442:     de la aplicación y se crea un puntero global para acceder a la
443:     base de datos.
444: *}
445: to begin do
446: begin
447:
448:     loadConfig;
449:
450:     data := studentStreamOpen(DATABASE_FILE_NAME);
451:
452:     if data = NIL then
453:     begin
454:
455:         writeln("Error: No es posible acceder al archivo '", DATABASE_FILE_NAME, "', el programa se detendrá
            ahora.");
456:
457:         halt;
458:
459:     end;
460: end;
461:
462: {*
463:     Al terminar el programa se "cierra" la base de datos.
464: *}
465: to end do studentStreamClose(data);
466:
467: END.

```

```

1: MODULE sort INTERFACE;
2:
3: {*
4:     Este módulo proporciona los procedimientos y funciones relacionados
5:     con la ordenación de los datos.
6: *}
7:
8: export sort = (bubbleSortAlphabetical, bubbleSortByMarks);
9:
10: IMPORT
11:
12:     filesystem;
13:
14: {*
15:     Implementa el método de la burbuja para ordenar
16:     el array de alumnos link^ alfabéticamente.
17: *}
18: procedure bubbleSortAlphabetical(link:PtrToDump);
19:
20: {*
21:     Implementa el método de la burbuja para ordenar
22:     el array de alumnos link^ mediante sus notas.
23:     option indica la convocatoria para la cual se
24:     está haciendo el ordenamiento (0 para Febrero/Junio,
25:     1 para Septiembre y 2 para Diciembre).
26: *}
27: procedure bubbleSortByMarks(link:PtrToDump; option:integer);
28:
29: {*
30:     Compara los login a y b y devuelve -1 si a < b,
31:     1 si a > b o 0 si a = b.
32: *}
33: function compareLogin(protected var a, b:array of char):integer;
34:
35: {*
36:     Compara las cadenas a y b y devuelve -1 si a < b,
37:     1 si a > b o 0 si a = b.
38: *}
39: function compareString(protected var a, b:string):integer;
40:
41: {*
42:     Es utilizada por bubbleSortAlphabetical, devuelve true
43:     si a y b deben ser intercambiados o false en caso contrario.
44: *}
45: function needSwapAlphabetical(a, b:PtrToStudent):boolean;
46:
47: {*
48:     Es utilizada por bubbleSortAlphabetical, devuelve true
49:     si a y b deben ser intercambiados o false en caso contrario.
50: *}
51: function needSwapByMarks(a, b:PtrToStudent; option:integer):boolean;
52:
53: {*
54:     Comprueba si el caracter en las posiciones second - 1 y second de
55:     str (2 bytes) está en la variable global wanted y lo intercambia
56:     por su contrapartida en la variable global sanitized, de un byte,
57:     y le asigna el valor 255 al segundo byte.
58: *}
59: procedure sanitizeChar(var str:string; second:integer);
60:

```

```

61:  {*
62:      Pasa a minúsculas los caracteres de str y llama
63:      a sanitizeChar si encuentra un byte susceptible
64:      de ser la primera parte de un caracter de 2 bytes
65:      de los presentes en la variable global wanted.
66:  *}
67:  procedure sanitizeString(var str:string);
68:
69:  {*
70:      Intercambia la zona a la que apuntan a y b.
71:  *}
72:  procedure swap(var a, b:PtrToStudent);
73:
74:  END.
75:
76:
77: MODULE sort IMPLEMENTATION;
78:
79: CONST
80:
81:     {*
82:         Diferencia entre las letras mayúsculas y las minúsculas
83:         del alfabeto inglés en ASCII y UTF-8.
84:     *}
85:     ASCII_UPPER_TO_LOWER  = 32;
86:
87:     {*
88:         Pesos de las notas práctica y teórica a la hora de calcular
89:         la global, mediante una media ponderada.
90:     *}
91:     WEIGHT_PRACTICE       = 2;
92:     WEIGHT_THEORY          = 8;
93:
94:  VAR
95:
96:     {*
97:         Arrays globales para almacenar las correspondencias
98:         entre caracteres de 2 y 1 byte. Son globales para evitar
99:         tener que inicializarlos cada vez que se llama a la función
100:        sanitizeChar (se inicializan al cargar el módulo por
101:        primera vez).
102:    *}
103:    sanitized:array[0..11] of char;
104:    wanted:array[0..11] of string(2);
105:
106:  procedure bubbleSortAlphabetical;
107:
108:  var
109:
110:      i:integer;
111:      ordered:boolean;
112:
113:  begin
114:
115:      repeat
116:
117:          ordered := true;
118:
119:          for i := 0 to link^.eod - 1 do
120:              begin

```

```

120:         if needSwapAlphabetical(link^[i], link^[i + 1]) then
121:             begin
122:
123:                 swap(link^[i], link^[i + 1]);
124:
125:                 ordered := false;
126:
127:             end;
128:
129:         end;
130:
131:     until ordered;
132:
133: end;
134:
135: procedure bubbleSortByMarks;
136: var
137:
138:     i:integer;
139:     ordered:boolean;
140:
141: begin
142:
143:     repeat
144:
145:         ordered := true;
146:
147:         for i := 0 to link^.eod - 1 do
148:             begin
149:
150:                 if needSwapByMarks(link^[i], link^[i + 1], option) then
151:                     begin
152:
153:                         swap(link^[i], link^[i + 1]);
154:
155:                         ordered := false;
156:
157:                     end;
158:
159:                 end;
160:
161:         until ordered;
162:
163:     end;
164:
165: function compareLogin;
166: var
167:
168:     done:integer value 0;
169:     i:integer value 0;
170:
171: begin
172:
173:     while (done = 0) and (i < 8) do
174:         begin
175:
176:             if a[i] < b[i] then done := -1
177:             else if a[i] > b[i] then done := 1;
178:             ;
179:

```



```

180:         i := i + 1;
181:
182:     end;
183:
184:     compareLogin := done;
185:
186: end;
187:
188: function compareString;
189: var
190:
191:     aCopy, bCopy:string(40);
192:
193: begin
194:
195:     aCopy := a;
196:     bCopy := b;
197:
198:     sanitizeString(aCopy);
199:     sanitizeString(bCopy);
200:
201:     if aCopy < bCopy then compareString := -1
202:     else if aCopy > bCopy then compareString := 1
203:     else compareString := 0;
204:     ;
205:
206: end;
207:
208: function needSwapAlphabetical;
209: var
210:
211:     compared:integer;
212:
213: begin
214:
215:     compared := compareString(a^.surname, b^.surname);
216:
217:     if compared = 1 then needSwapAlphabetical := true
218:     else if compared = -1 then needSwapAlphabetical := false
219:     else
220:     begin
221:
222:         compared := compareString(a^.name, b^.name);
223:
224:         if compared = 1 then needSwapAlphabetical := true
225:         else if compared = -1 then needSwapAlphabetical := false
226:         else
227:         begin
228:
229:             if compareLogin(a^.login, b^.login) = 1 then needSwapAlphabetical := true
230:             else needSwapAlphabetical := false;
231:
232:         end;
233:         ;
234:
235:     end;
236:     ;
237:
238: end;
239:

```

```

240: function needSwapByMarks;
241: var
242:
243:     globalA, globalB: real;
244:
245: begin
246:
247:     globalA := (a^.marks[option].practice * WEIGHT_PRACTICE
248:               + a^.marks[option].theory * WEIGHT_THEORY)
249:               / (WEIGHT_PRACTICE + WEIGHT_THEORY);
250:
251:     globalB := (b^.marks[option].practice * WEIGHT_PRACTICE
252:               + b^.marks[option].theory * WEIGHT_THEORY)
253:               / (WEIGHT_PRACTICE + WEIGHT_THEORY);
254:
255:     if globalA > globalB then needSwapByMarks := false
256:     else if globalA < globalB then needSwapByMarks := true
257:     else
258:     begin
259:
260:         if a^.marks[option].theory > b^.marks[option].theory then
261:             needSwapByMarks := false
262:         else if a^.marks[option].theory < b^.marks[option].theory then
263:             needSwapByMarks := true
264:         else
265:         begin
266:
267:             if a^.marks[option].practice > b^.marks[option].practice then
268:                 needSwapByMarks := false
269:             else if a^.marks[option].practice < b^.marks[option].practice then
270:                 needSwapByMarks := true
271:             else
272:                 needSwapByMarks := needSwapAlphabetical(a, b);
273:             ;
274:
275:         end;
276:         ;
277:
278:     end;
279:     ;
280:
281: end;
282:
283: procedure sanitizeChar;
284: var
285:
286:     done: boolean value false;
287:     i: integer value 0;
288:
289: begin
290:
291:     while (not done) and (i < 12) do
292:     begin
293:
294:         if str[second] = wanted[i][2] then
295:         begin
296:
297:             str[second - 1] := sanitized[i];
298:             str[second] := chr(255);
299:

```

```

300:         done := true;
301:
302:     end
303:     else i := i + 1;
304:
305: end;
306:
307: end;
308:
309: procedure sanitizeString;
310: var
311:
312:     i:integer;
313:
314: begin
315:
316:     for i := 1 to length(str) - 1 do
317:     begin
318:
319:         if str[i] = chr(195) then
320:         begin
321:
322:             sanitizeChar(str, i + 1);
323:
324:         end;
325:
326:     end;
327:
328:     for i := 1 to length(str) do
329:     begin
330:
331:         if (ord(str[i]) > 64) and (ord(str[i]) < 91) then
332:         begin
333:
334:             str[i] := chr(ord(str[i]) + ASCII_UPPER_TO_LOWER);
335:
336:         end;
337:
338:     end;
339:
340: end;
341:
342: procedure swap;
343: var
344:
345:     aux:PtrToStudent;
346:
347: begin
348:
349:     aux := a;
350:     a := b;
351:     b := aux;
352:
353: end;
354:

```

```
355: to begin do
356: begin
357:
358:     wanted[0] := 'á';
359:     wanted[1] := 'é';
360:     wanted[2] := 'í';
361:     wanted[3] := 'ó';
362:     wanted[4] := 'ú';
363:     wanted[5] := 'ñ';
364:     wanted[6] := 'Á';
365:     wanted[7] := 'É';
366:     wanted[8] := 'Í';
367:     wanted[9] := 'Ó';
368:     wanted[10] := 'Ú';
369:     wanted[11] := 'Ñ';
370:
371:     sanitized[0] := 'a';
372:     sanitized[1] := 'e';
373:     sanitized[2] := 'i';
374:     sanitized[3] := 'o';
375:     sanitized[4] := 'u';
376:     sanitized[5] := 'n';
377:     sanitized[6] := 'a';
378:     sanitized[7] := 'e';
379:     sanitized[8] := 'i';
380:     sanitized[9] := 'o';
381:     sanitized[10] := 'u';
382:     sanitized[11] := 'n';
383:
384: end;
385:
386: END.
```

```

1: MODULE printlist INTERFACE;
2:
3: {*
4:     Este módulo proporciona el procedimiento para imprimir la lista de
5:     alumnos en la salida estándar.
6: *}
7:
8: export printlist = (printOrdered);
9:
10: IMPORT
11:
12:     filesystem;
13:
14: {*
15:     Imprime una lista con los alumnos del array link^. printMarks indica
16:     si se deben imprimir las notas de alguna convocatoria (0 para
17:     Febrero/Junio, 1 para Septiembre, 2 para Diciembre y < 0 para
18:     no imprimir las notas).
19: *}
20: procedure printOrdered(link:PtrToDump; printMarks:integer);
21:
22: {*
23:     Detiene la impresión de la lista y pide introducción de texto
24:     por parte del usuario mediante la entrada estándar. Devuelve true
25:     si el usuario ha introducido una q/Q o false en caso contrario.
26: *}
27: function stop:boolean;
28:
29: END.
30:
31: MODULE printlist IMPLEMENTATION;
32:
33: IMPORT
34:
35:     standardoutput;
36:     display;
37:     filteredinput;
38:
39: CONST
40:
41:     {*
42:         Máximo número de líneas a imprimir sin llamar a stop.
43:     *}
44:     MAX_LINES          = 15;
45:     {*
46:         Pesos de las notas práctica y teórica a la hora de calcular
47:         la global, mediante una media ponderada.
48:     *}
49:     WEIGHT_PRACTICE = 2;
50:     WEIGHT_THEORY   = 8;
51:
52: procedure printOrdered;
53: var
54:
55:     c, i:integer value 0;
56:     foo:char;
57:     margin:integer value 15;
58:     pause, quit:boolean;
59:     temp:real;
60:

```

```

61: begin
62:
63:   clearDisplay;
64:
65:   case printMarks of
66:
67:     0: setTitle('Listado de alumnos con las notas de Febrero/Junio');
68:     1: setTitle('Listado de alumnos con las notas de Septiembre');
69:     2: setTitle('Listado de alumnos con las notas de Diciembre');
70:
71:     otherwise
72:       begin
73:
74:         setTitle('Listado alfabético de alumnos');
75:         margin := 20;
76:
77:       end;
78:
79:   end;
80:
81:   moveMargin(-8);
82:
83:   repeat
84:
85:     setFormat(BOLD);
86:
87:     write('Apellidos':margin, 'Nombre':margin, 'Login':margin);
88:     if printMarks > -1 then write('Teórica':10, 'Práctica':10, 'Global':10);
89:
90:     resetFormat;
91:     downLines(1);
92:
93:     pause := false;
94:
95:     while (not pause) and (i <= link^.eod) do
96:       begin
97:
98:         if (i mod 2) = 1 then setFormat(REVERSE);
99:
100:        write(link^[i]^^.surname:margin, link^[i]^^.name:margin, link^[i]^^.login:margin);
101:
102:        if printMarks > -1 then
103:          begin
104:
105:            if trunc(link^[i]^^.marks[printMarks].theory) = link^[i]^^.marks[printMarks].theory then
106:              write(link^[i]^^.marks[printMarks].theory:10:0)
107:            else write(link^[i]^^.marks[printMarks].theory:10:1);
108:
109:            if trunc(link^[i]^^.marks[printMarks].practice) = link^[i]^^.marks[printMarks].practice then
110:              write(link^[i]^^.marks[printMarks].practice:10:0)
111:            else write(link^[i]^^.marks[printMarks].practice:10:1);
112:
113:            temp := (link^[i]^^.marks[printMarks].practice * WEIGHT_PRACTICE
114:                    + link^[i]^^.marks[printMarks].theory * WEIGHT_THEORY)
115:                    / (WEIGHT_PRACTICE + WEIGHT_THEORY);
116:
117:            if trunc(temp) = temp then write(temp:10:0)
118:            else write(temp:10:1);
119:
120:            write(' ');

```

```

121:
122:     end;
123:
124:     if (i mod 2) = 1 then resetFormat;
125:
126:     downLines(1);
127:
128:     c := c + 1;
129:     i := i + 1;
130:
131:     if (c = MAX_LINES) and (link^.eod - i > 4) then
132:     begin
133:
134:         c := 0;
135:         pause := true;
136:
137:     end;
138:
139: end;
140:
141: if i = link^.eod + 1 then
142: begin
143:
144:     downLines(1);
145:     write('— Pulsa intro para volver al menú principal —');
146:     readChar(foo);
147:
148:     quit := true;
149:
150: end
151: else quit := stop;
152:
153: until quit;
154:
155: end;
156:
157: function stop;
158: var
159:
160:     option:char;
161:
162: begin
163:
164:     downLines(1);
165:     write('Pulsa intro para continuar (q e intro para volver al menú principal): ');
166:     readChar(option);
167:
168:     if option in ['q', 'Q'] then stop := true
169:     else
170:     begin
171:
172:         stop := false;
173:         clearLine;
174:
175:     end;
176:
177: end;
178:
179: END.

```