1. /\* x o x
2. \* fechas.h funciones para trabajar con fechas (1.1)
3. \* x o x \*/ 4
4. #ifndef FECHAS\_H\_
5. #define FECHAS\_H\_

7

8 #include <stdio.h>

9

10

11 **/\*\***

1. **\* tipo de dato para la fecha, los miembros de la 'struct' podrían estar en**
2. **\* otro orden. En ese caso, este sería el tipo de dado tFecahaDMA**
3. **\* las restantes combinaciones (más habituales), serían los tipos:**
4. **\* tFecahaAMD, tFecahaMDA y se debería disponer de toda la colección**
5. **\* de operaciones**
6. **\*\*/**
7. **typedef struct**
8. {
9. **int** di,
10. me,
11. an;
12. } tFecha;

24

25 **/\*\***

1. **\* función booleana que permite el ingreso de tres enteros para el día, mes**
2. **\* y año sin garantía de que correspondan a una fecha.**
3. **\* contempla el caso en que se quiera mostrar un mensaje distinto del**
4. **\* mensaje por defecto el que se mostrará si recibe NULL.**
5. **\* contempla el caso en que no se quiera mostrar ningún mensaje cuando reciba**
6. **\* una cadena vacía (o con blancos o tabulaciones), en este caso, de ser**
7. **\* necesario, se mostrará el mensaje antes de invocarla.**
8. **\* contempla el caso en que no se quiera ingresar nada, por ejemplo con**
9. **\* cero en alguno de los enteros, con lo que devolverá 0 (cero)**
10. **\*\*/**
11. **int** ingresarFechaDMA(tFecha \*fec, **const char** \*mensajeOpcional); 37

38 **/\*\***

1. **\* muestra una fecha en el formato dia/mes/año**
2. **\*\*/**
3. **void** mostrarFechaDMA(**const** tFecha \*fec); 42

43 **/\*\***

1. **\* Vigencia del Calendario Gregoriano**
2. **\* Al jueves -juliano- 4 de octubre de 1582 le sucede el**
3. **\* viernes -gregoriano- 15 de octubre de 1582.**
4. **\* Diez días desaparecen debido a que ya se habían contado de más en el**
5. **\* calendario juliano.**
6. **\* Si hubiera habido calendario gregoriano:**
7. **\* 01/01/1582 -> VIERNES**
8. **\* 01/01/1581 -> JUEVES**
9. **\* 01/01/1580 -> MARTES**
10. **\* ...**
11. **\* tomaremos como mínimo válido el año 1 (uno), teniendo en cuenta que no**
12. **\* existió el año 0 (cero), 'extrapolando' la vigencia del calendario**
13. **\* gregoriano (su vigencia es -en España, Italia y Portugal, a saber, a `**
14. **\* partir del 15 de octubre de 1582)**
15. **\* por no ser adoptado en forma universal, su vigencia depende del país**
16. **\* NOTA: se ponen topes mínimo y máximo, y nuestros algoritmos quedarán**
17. **\* abiertos a modificaciones (que hasta ahora están más allá)**
18. **\* OTRA: a partir de su vigencia, por una pequeña diferencia, cada 3623 años**
19. **\* habrá un día más y para corregirlo, se dejará de contar un bisiesto**
20. **\* cada 3000 años**
21. **\* esto quedará como tema abierto a quién tenga intereses muy trascendentes, y**
22. **\* deberá tener en cuenta determinaciones astronómicas más precisas**
23. **\* si extrapolamos dentro de un rango razonable nos queda...**
24. **\*\*/**
25. #define AN\_MIN 1
26. #define AN\_MAX 5000

70

71 **/\*\***

1. **\* para determinar si un año es bisiesto se lo puede hacer con una función, o**
2. **\* mejor, como en este caso con un macroreemplazo**
3. **\*\*/**

75 #define esBisiesto( X ) ( ( ( X ) % 4 == 0 && ( X ) % 100 != 0 ) || \

76 ( X ) % 400 == 0 )

77

78 **/\*\***

1. **\* determina si una fecha es válida**
2. **\*\*/**
3. **int** esFechaValida(**const** tFecha \*fec); 82

83 **/\*\***

1. **\* permite el ingreso de una fecha válida (valiéndose de las dos anteriores),**
2. **\* contemplando el caso de que no se quiera ingresarla, además de la**
3. **\* posibilidad de mostrar un mensaje distinto del mensaje por defecto**
4. **\*\*/**
5. **int** ingresarFechaValidaDMA(tFecha \*fec, **const char** \*mensajeOpcional); 89

90 **/\*\***

1. **\* compara dos fechas devolviendo 0 si son iguales, algún valor negativo**
2. **\* si la primera es menor que la segunda, algún valor positivo si la**
3. **\* primera es mayor que la segunda**
4. **\*\*/**
5. **int** compararFecha(**const** tFecha \*fec1, **const** tFecha \*fec2); 96

97 **/\*\***

1. **\* calcula y devuelve el día del año de esa fecha**
2. **\* precondición: que sea una fecha válida**
3. **\* NOTA:**
4. **\* Una fecha juliana se usa a veces para hacer referencia a un formato de**
5. **\* fecha que es una combinación del año actual y el número de días desde el**
6. **\* principio del año. Por ejemplo, 1 de enero de 2007 se representa como**
7. **\* 2007001 y 31 de diciembre de 2007 se representa como 2007365. Tenga en**
8. **\* cuenta que este formato no se basa en el calendario juliano.**
9. **\*\*/**
10. **int** aJuliano(**const** tFecha \*fec); 108

109 **/\*\***

1. **\* la convención, más comunmente utilizada, adoptada a continuación se puede**
2. **\* alterar a gusto del programador, a costa de apartarse del estándar**
3. **\*\*/**
4. #define DOMINGO 0
5. #define LUNES 1
6. #define MARTES 2
7. #define MIERCOLES 3
8. #define JUEVES 4
9. #define VIERNES 5
10. #define SABADO 6

120

121 **/\*\***

1. **\* determina el número de día de la semana de una fecha**
2. **\* se tiene en cuenta que:**
3. **\* - en un lapso de 400 años la cantidad de días es múltiplo de 7**
4. **\* - para cada uno de los años consecutivos de esos intervalos el primero de**
5. **\* enero de cada año comienza en el mismo día**
6. **\* - el primero de enero de los años múltiplo de 400 es SABADO (6)**
7. **\* tomamos como año base el año múltiplo de 400 inmediato anterior al año**
8. **\* pero sólo es necesario calcular la distancia al año base en años**
9. **\* por cada año se suma 1 al nro de día del año base, salvo que**
10. **\* cada cuatro años es bisiesto y se suma uno más, pero cada 100 años**
11. **\* no es bisiesto y se resta 1.**
12. **\* si el año no fuera el año base calculado, se suma 1 porque es bisiesto**
13. **\*\*/**
14. **int** nroDeDiaDeLaSemana(**const** tFecha \*fec); 136

137 **/\*\***

1. **\* determina la cantidad de días desde fecDesde hasta fecHasta, que será**
2. **\* negativa si fecDesde es mayor que fecHasta**
3. **\*\*/**
4. **long** diasEntreFechas(**const** tFecha \*fecDesde, **const** tFecha \*fecHasta); 142

143 **/\*\***

1. **\* calcula la cantidad de días, meses y años entre dos fechas, y lo devuelve**
2. **\* como una fecha**
3. **\*\*/**
4. tFecha calcularEdad(**const** tFecha \*fecDesde, **const** tFecha \*fecHasta); 148

149 **/\*\***

1. **#include <stdlib.h>**
2. **#include <string.h>**

152 **\*\*/**

153

154

155 **/\*\***

1. **\* tipo de dato 'extendido' para la fecha con mes y día en palabras**
2. **\*\* /**
3. **typedef struct**
4. **{**
5. **int di,**
6. **me,**
7. **an;**
8. **int nroDia0Do1Lu;**
9. **char nombreDiaLargo[10];**
10. **char nombreDiaCorto[3];**
11. **char nombreMesLargo[11];**
12. **char nombreMesCorto[4];**
13. **} tFechaEx; \*\*/**

169

170 **/\*\***

171 int generarFechaEx(tFechaEx \*fecEx, const tFecha \*fec);

172 **\*\*/**

173

174

175

176 #endif

177

178

1. /\* x o x
2. \* fechas.c funciones para trabajar con fechas (1.1)
3. \* x o x \*/ 182

183 #include "fechas.h"

184

1. **int** ingresarFechaDMA(tFecha \*fec, **const char** \*mensajeOpcional)
2. {
3. **if**(mensajeOpcional)
4. printf("%s", mensajeOpcional);
5. **else**
6. printf("Fecha (dd/mm/aaaa - 0=No Ingresa): ");
7. fec->di = 0;
8. fec->me = 0;
9. fec->an = 0;
10. fflush(stdin);
11. scanf("%d/%d/%d", &fec->di, &fec->me, &fec->an);
12. **return** fec->di && fec->me && fec->an;

197 }

198

1. **void** mostrarFechaDMA(**const** tFecha \*fec)
2. {

201 printf("%02d/%02d/%04d", fec->di, fec->me, fec->an);

202 }

203

1. **int** esFechaValida(**const** tFecha \*fec)
2. {
3. **static const char** dias[][12] = {

207 { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },

208 { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } };

1. **return** fec->me > 0 && fec->me <= 12 &&
2. fec->an >= AN\_MIN && fec->an <= AN\_MAX &&
3. fec->di > 0 && fec->di <= dias[esBisiesto(fec->an)][fec->me - 1];

212 }

213

1. **int** ingresarFechaValidaDMA(tFecha \*fec, **const char** \*mensajeOpcional)
2. {
3. **do**
4. {
5. **if**(!ingresarFechaDMA(fec, mensajeOpcional))
6. **return** 0;
7. } **while**(!esFechaValida(fec));
8. **return** 1;

222 }

223

1. **int** compararFecha(**const** tFecha \*fec1, **const** tFecha \*fec2)
2. {
3. **int** cmp = fec1->an - fec2->an; 227
4. **if**(cmp)
5. **return** cmp;
6. cmp = fec1->me - fec2->me;
7. **if**(cmp)
8. **return** cmp;
9. **return** fec1->di - fec2->di;

234 }

235

1. **int** aJuliano(**const** tFecha \*fec)
2. {

238 **int** dias[][12] = {

239 { 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334 },

240 { 0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 } };

241

242 **return** dias[esBisiesto(fec->an)][fec->me - 1] + fec->di;

243 }

244

1. **int** nroDeDiaDeLaSemana(**const** tFecha \*fec)
2. {
3. **int** distBase = fec->an % 400; 248
4. **return** ( SABADO - 1 + distBase + distBase / 4 - distBase / 100 +
5. (distBase != 0) + aJuliano(fec) ) % 7;

251 }

252

1. **long** diasEntreFechas(**const** tFecha \*fecDesde, **const** tFecha \*fecHasta)
2. {
3. **int** anBase = fecDesde->an <= fecHasta->an ? fecDesde->an : fecHasta->an,
4. distBaseDesde = fecDesde->an - anBase,
5. distBaseHasta = fecHasta->an - anBase;
6. **long** diasBaseAHasta = distBaseHasta \* 365L + distBaseHasta / 4 -
7. distBaseHasta / 100 + distBaseHasta / 400 +
8. (distBaseHasta != 0) + aJuliano(fecHasta),
9. diasBaseADesde = distBaseDesde \* 365L + distBaseDesde / 4 -
10. distBaseDesde / 100 + distBaseDesde / 400 +
11. (distBaseDesde != 0) + aJuliano(fecDesde);
12. **return** diasBaseAHasta - diasBaseADesde;

265 }

266

1. tFecha calcularEdad(**const** tFecha \*fecDesde, **const** tFecha \*fecHasta)
2. {
3. **static const char** dias[][13] = {

270 { 0, 31, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30 },

271 { 0, 31, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30 } };

1. **/// dic ene feb mar abr may jun jul ago set oct nov**
2. tFecha edad;

274

1. edad = \*fecHasta;
2. **if**( (edad.di -= fecDesde->di) < 0)
3. {
4. edad.di += dias[esBisiesto(edad.an)][edad.me];
5. edad.me--;

280 }

1. **if**( (edad.me -= fecDesde->me) < 0)
2. {
3. edad.me += 12;
4. edad.an--;

285 }

1. edad.an -= fecDesde->an;
2. **return** edad;

288 }

289

290 **/\*\***

1. **int generarFechaEx(tFechaEx \*fecEx, const tFecha \*fec)**
2. **{**
3. **static const char dias[][10] = {**
4. **{ "domingo" }, { "lunes" }, { "martes" }, { "miercoles" },**
5. **{ "jueves" }, { "viernes" }, { "sabado" } };**
6. **static const char meses[][12] = {**
7. **{ "enero" }, { "febrero" }, { "marzo" }, { "abril" },**
8. **{ "mayo" }, { "junio" }, { "julio" }, { "agosto" },**
9. **{ "septiembre" }, { "octubre" }, { "noviembre" }, { "diciembre" } };**
10. **int aux;**

301

1. **memset(fecEx, 0, sizeof(tFechaEx));**
2. **fecEx->di = fec->di;**
3. **fecEx->me = fec->me;**
4. **fecEx->an = fec->an;**
5. **if(!esFechaValida(fec))**
6. **return 0;**
7. **aux = nroDeDiaDeLaSemana(fec);**
8. **fecEx->nroDia0Do1Lu = aux;**
9. **strcpy(fecEx->nombreDiaLargo, dias[aux]);**
10. **\*fecEx->nombreDiaCorto = '\0';**
11. **strncat(fecEx->nombreDiaCorto, dias[aux], 2);**

313

314 aux = fec->me - 1;

315 strcpy(fecEx->nombreMesLargo, meses[aux]);

316 \*fecEx->nombreMesCorto = '\0';

317 strncat(fecEx->nombreMesLargo, meses[aux], 3);

318

319 **return 1;**

320 **} \*\*/**

321

322