TDA Pila

Tpila = reg

Cima: entero

Elem: Arreglo[CMAX] de Tdato

Freg

Proced Pcrear(Ref P:Tpila)

P.cima 🡨 0

Fproced

Func Pvacia(Ref P:Tpila): lógico

Pvacia 🡨 P.cima = 0

FFunc

Proced Pponer(Ref P:Tpila, val x:tdato)

P.cima 🡨 p.cima + 1

P.elem[P.cima] 🡨 X

Fproced

Proced Psacar(Ref P:Tpila, ref X:TDato)

X 🡨 P.elem[P.cima]

p.cima 🡨 p.cima – 1

Fproced

Func Pllena(Ref P:Tpila): lógico

Pllena 🡨 p.cima = CMAX

Ffunc

Pila Dinamica

Tpila = Puntero = ↑nodo

Nodo = reg

Sig: puntero

Info: tdato

Freg

Proced Pcrear(Ref P:Tpila)

P 🡨 nil

Fproced

Func Pvacia(Ref P:Tpila)

Pvacia 🡨 p = nil

Ffunc

Proced Pponer(Ref P:Tpila, val x:Tdato)

Var local: nuevo: puntero

CrearNodo(Nuevo, x)

Nuevo↑.sig 🡨 p

p↑.🡨 Nuevo

Fproced

Proced Psacar(Ref P:Tpila, Ref x:Tdato)

Var local: aux: puntero

X 🡨 p↑.info

Aux 🡨 p

P 🡨 p↑.sig

Disponer(aux)

Fproced

Función Pllena(Ref P:Tpila): lógico

Var local: Nuevo: nodo

Pllena 🡨 memdisponible < sizeof(nuevo)

Ffunc

TDA Cola

Cola estatica

Tcola = reg

Final: entero

Elem: Arreglo[CMAX] de Tdato

Freg

Proced Ccrear(Ref q:Tcola)

q.final 🡨 0

Fproce

Func Cvacia(ref q:tcola): lógico

Cvacia 🡨 q.final = 0

Ffunc

Proced Cponer(ref q:Tcola, Val x:tdato)

q.final 🡨 q.final + 1

q.elem[q.final] 🡨 x

Fproced

Proced Csacar(Ref q:Tcola, ref x:Tdato)

Var local: I: entero

X 🡨 q.elem[1]

Para I desde 1 hasta q.final – 1 hacer

q.elem[i] 🡨 q.elem[i+1]

Fpara

q.final 🡨 q.final – 1

Fproced

Func Cllena(Ref q:Tcola): lógico

Cllena 🡨 q.final = CMAX

FFunc

Cola Estatica circular

Tcola = reg

Frente, final: entero

Elem: arreglo[CMAX] de Tdato

Freg

Proced CCrear(Ref q:Tcola)

q.frente 🡨 1

q.final 🡨 1

FProced

Func Cvacia(Ref q:Tcola): lógico

CVacia 🡨 q.final = q.frente

FFunc

Proced Cponer(Ref q:Tcola, val x:tdato)

Si q.final = CMAX entonces

q.final 🡨 1

sino

q.final 🡨 q.final + 1

Fsi

q.elem[q.final] 🡨 x

Fproced

Proced Csacar(Ref q:Tcola, Ref x:Tdato)

si q.frente = cmax entonces

q.frente 🡨 1

Sino

q.frente 🡨 q.frente + 1

fsi

x 🡨 q.elem[q.frente]

FProced

Func Cllena(ref q:Tcola): lógico

Cllena 🡨 (q.frente = q.final + 1) o (q.frente=1 y q.final=CMAX)

Ffunc

Cola dinámica

Tcola = puntero = ↑nodo

Nodo = reg

Sig: puntero

Info: Tdato

Freg

Proced ccrear(Ref q:Tcola)

Q 🡨 nil

Fproced

Func Cvacia(Ref q:Tcola): lógico

Cvacia 🡨 q = nil

Ffunc

Proced Cponer(Ref q:Tcola, val x:Tdato)

Var local: Nuevo, aux: puntero

CrearNodo(Nuevo, x)

Si q = nil entonces

Q 🡨 Nuevo

Sino

AUX 🡨 Q

Mientras aux↑.sig != nil hacer

Aux 🡨 aux↑.sig

Fmientras

Aux↑.sig 🡨 Nuevo

Fsi

Fproced

Proced Csacar(Ref q:Tcola, ref x:Tdato)

Var local: aux: puntero

Aux 🡨 q

X 🡨q↑.info

Q 🡨q↑.sig

Disponer(Aux)

Fproced

Func Cllena(ref q:Tcola): lógico

Var local: Nuevo: nodo

Cllena 🡨 memdisponible < sizeof(Nuevo)

Ffunc

TDA LISTA

Lista estatica con corrimiento

Tlista = reg

Tope, actual: entero

Elem: arreglo[CMAX] de Tdato

Freg

Proced lcrear(Ref L:Tlista)

L.tope 🡨 0

Fproced

Func Lvacia(ref L:Tlista): lógico

Lvacia 🡨 L.tope = 0

Ffunc

Proced Lppio(Ref L:Tlista)

L.actual 🡨 1

Fproced

Proced Linfo(ref l:Tlista, ref x:Tdato)

X 🡨 l.elem[l.actual]

Fproced

Proced Lsig(ref L:Tlista)

L.actual 🡨 l.actual + 1

Fproced

Func Lfin(ref L:Tlista): lógico

Lfin 🡨 l.actual > l.tope

Ffunc

Func Lllena(ref L:Tlista): lógico

Lllena 🡨 L.tope = CMAX

Ffunc

Proced lmodificar(ref L:Tlista, val x:tdato)

l.elem[l.actual] 🡨 x

Fproced

Proced Linsertarppio(Ref L:Tlista, val x:TDato)

Var local: i: entero

Para I desde l.tope hasta 1 (Paso - 1)hacer

L.elem[i+1] 🡨 l.elem[i]

Fpara

l.elem[1] 🡨 x

L.tope 🡨 l.tope + 1

Fproced

Proced Linsertarfin(ref L:Tlista, val x:Tdato)

L.tope 🡨 l.tope + 1

L.elem[L.tope] 🡨 x

FProced

Proced Linsertarorden(ref L:Tlista, val x:Tdato, Val ord: Caracter)

Var local: I, pos: entero

I 🡨 1

Mientras i != 0 y (l.elem[i].clave > x.clave y ord=’D’) o (L.elem[i].clave < x.clave y ord=’A’) entonces

I 🡨 i + 1

FMientras

Para pos desde L.tope hasta i hacer (Paso - 1)

L.elem[pos + 1] 🡨 l.elem[Pos]

Fpara

L.elem[pos] 🡨 x

l.tope 🡨 l.tope + 1

Fproced

Proced Lbuscarordenado (Ref L:Tlista, val x:Tclave, ref Existe: lógico)

Var local: pr, ult: entero, Central: entero, orden: caracter

Existe 🡨 falso

Si l.tope ¡= 0 entonces

Si L.elem[1].clave > l.elem[l.tope].clave entonces

Orden 🡨 ‘D’

Sino

Orden 🡨 ‘A’

Fsi

Pr 🡨 1

Ult 🡨 l.tope

Si orden = ‘A’ entonces

Mientras pr <= ult y No existe hacer

CentraL 🡨 ( pr + ult ) Div 2

Si l.elem[central]. Clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 central

sino

Si l.elem[Central].clave > x entonces

Ult 🡨 central – 1

Sino

Pr 🡨 central + 1

Fsi

Fsi

Fmientras

Sino

Mientras pr <= ult y No Existe hacer

Central 🡨 (pr + ult) div 2

Si L.elem[Central].clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 central

sino

Si L.elem[Central].clave > x entonces

Pr 🡨 central + 1

Sino

Ult 🡨 central – 1

Fsi

Fsi

Fmientras

Fsi

Fproced

Proced Lborrarppio(ref L:Tlista)

Var local: i: entero

Para i desde 1 hasta l.tope – 1 hacer

L.elem[i] 🡨 l.elem[i+1]

Fpara

L.tope 🡨 l.tope + 1

FProced

Proced Lborrarfin(Ref L:Tlista)

L.tope 🡨 l.tope – 1

Fproced

Proced LborrarActual(ref L:Tlista)

Var local: I: entero

Para I desde l.actual hasta l.tope – 1 hacer

l.elem[i] 🡨 l.elem[i +1]

Fpara

l.tope 🡨 l.tope – 1

Fproced

Lista estatica Sin corrimiento

Tlista = reg

Pocup, plibre, actual: puntero

Elem: arreglo[Cmax] de Telem

Freg

Telem = Reg

Info: tdato

Sig: entero

Freg

Proced Lcrear(ref L:TLista)

Var local: i: entero

l.pocup 🡨 0

l.plibre 🡨 1

Para i desde 1 hasta CMAX – 1 hacer

l.elem[i].sig 🡨 i + 1

Fpara

L.elem[CMAX].sig 🡨 0

Fproced

Func Lvacia(Ref L:Tlista): lógico

Lvacia 🡨 l.pocup = 0

Ffunc

Proced Lppio(ref L:Tlista)

L.actual 🡨 l.pocup

Fproced

Proced linfo(ref L:Tlista, ref x:TDato)

X 🡨 l.elem[l.actua].info

Fproced

Proced Lsig(Ref L:Tlista)

L.actual 🡨 l.elem[l.actual].sig

Fproced

Func Lfin(ref l:Tlista)

Lfin 🡨 l.actual = 0

Ffunc

Proced Lmodificar(ref L:Tlista, val x:Tdato)

L.elem[L.actual].info 🡨 x

Fproced

Func Lllena(ref L:Tlista): lógico

Lllena 🡨 l.plibre = 0

Ffunc

Proced Linsertarppio(ref L:Tlista, val x:Tdato)

Var local: aux: entero

Aux 🡨 l.plibre

l.plibre 🡨 l.elem[l.plibre].sig

l.elem[aux].info 🡨 x

L.elem[aux].sig 🡨 l.pocup

L.pocup 🡨 aux

Fproced

Proced Linsertarfin(ref L:TLista, val x:Tdato)

Var local: aux, t: enteros

Aux 🡨 l.plibre

l.plibre 🡨 l.elem[l.plibre].sig

L.elem[aux].info 🡨 x

l.elem[aux].sig 🡨 0

Si l.pocup = 0 entonces

l.pocup 🡨 aux

sino

t 🡨 l.pocup

Mientras L.elem[t].sig ¡= 0 hacer

T 🡨 l.elem[t].sig

Fmientras

l.elem[t].sig 🡨 aux

Fsi

Fproced

Proced Linsertarorden(Ref L:TLista, val x:Tdato, Val orden:Caracter)

Var local: aux, t: entero

Aux 🡨 l.plibre

l.plibre 🡨 l.elem[L.plibre].sig

L.elem[aux].info 🡨 x

Si L.pocup = 0 entonces

L.elem[Aux].sig 🡨 0

l.pocup 🡨 Aux

sino

si (x.clave > l.elem[l.pocup].info.clave y orden = ‘D’) o (x.clave < l.elem[l.pocup].info.clave y orden = ‘A’) entonces

l.elem[aux].sig 🡨 L.pocup

l.pocup 🡨 aux

sino

t 🡨 l.pocup

Mientras (l.elem[t].sig ¡= 0) y ((l.elem[l.elem[t].sig].info.clave < x.clave y orden = ’A’) o (l.elem[l.elem[t].sig].info.clave > x.clave y orden = ’D’)) hacer

T 🡨 l.elem[t].sig

Fmientras

l.elem[aux].sig 🡨 l.elem[t].sig

l.elem[t].sig 🡨 aux

fsi

fsi  
FProced

Proced LbuscarOrdenado(Ref l:Tlista, val x:Tclave, Ref Existe: lógico)

Var local: aux, t: entero

Existe 🡨 falso

Si l.pocup ¡= 0 entonces

Si l.elem[l.pocup].info.clave = x entonces

l.actual 🡨 l.pocup

Existe 🡨 verdadero

Sino

T 🡨 l.pocup

Mientras (l.elem[t].sig ¡= 0) y (L.elem[l.elem[t].sig].info.clave = l.elem[l.pocup].info.clave) hacer

T 🡨 l.elem[t].sig

Fmientras

Si l.elem[t].sig ¡= 0 entonces

Si l.elem[l.pocup].info.clave > l.elem[l.elem[t].sig].info.clave entonces

J 🡨 l.elem[t].sig

Mientras j ¡= 0 y l.elem[j].info.clave > x hacer

j 🡨 l.elem[j].sig

Fmientras

Sino

J 🡨 l.elem[j].sig

Mientras j ¡= 0 y l.elem[j].info.clave < x hacer

j 🡨 l.elem[j].sig

Fmientras

Fsi

Si j ¡= 0 y l.elem[j].info.clave = x entonces

l.actual 🡨 j

Existe 🡨 verdadero

Fsi

Fsi

Fsi

Fsi

Fproced

Proced Lborrarppio(ref L:TLista)

Var local: aux: entero

Aux 🡨 l.pocup

l.opcup 🡨 l.elem[l.pocup].sig

l.elem[Aux].sig 🡨 l.plibre

l.plibre 🡨 aux

Fproced

Proced Lborrarfin(Ref L:Tlista)

Var local: aux, t:entero

Si l.elem[l.pocup].sig = 0 entonces

Aux 🡨 l.pocup

l.pocup 🡨 0

sino

t 🡨 l.pocup

Mientras l.elem[l.elem[t].sig].sig ¡= 0 hacer

T 🡨 l.elem[t].sig

Fmientras

Aux 🡨l.elem[t].sig

L.elem[t].sig 🡨 0

Fsi

l.elem[aux].sig 🡨 l.plibre

l.plibre 🡨 aux

Fproced

Proced LborrarActual(Ref L:Tlista)

Var local: aux: entero

Aux 🡨 l.actual

Si l.pocup = l.actual entonces

l.pocup 🡨 l.elem[l.pocup].sig

Sino

T 🡨 l.pocup

Mientras l.elem[t].sig != l.actual entonces

T 🡨 l.elem[t].sig

Fmientras

l.elem[t].sig 🡨 l.elem[l.actual].sig

Fsi

L.elem[aux].sig 🡨 l.plibre

l.plibre 🡨 aux

L.actual 🡨 l.elem[l.actual].sig

Fproced

Lista simplemente enlazada

Tlista = reg

Actual, cab: puntero

Freg

Nodo = reg

Sig: puntero

Info: Tdato

Freg

Proced Lcrear(Ref L:Tlista)

L.cab 🡨nil

Fproced

Func Lvacia(ref L:Tlista): lógico

Lvacia 🡨 l.cab = nil

Ffunc

Proced Lppio(ref L:Tlista)

L.actual 🡨 l.cab

Fproced

Proced Linfo(ref L:Tlista, Ref x:tdato)

X 🡨 l.actual↑.info

Fproced

Proced Lsig(ref L:Tlista)

l.actual 🡨 l.actual↑.Sig

Fproced

Func Lfin(ref L:Tlista): lógico

LFin 🡨 l.actual = nil

Ffunc

Proced Lmodificar(ref L:Tlista, val X:Tdato)

L.actual↑.info 🡨 x

Fproced

Func Lllena(ref l:Tlista): lógico

Var local: nuevo: nodo

Lllena 🡨 Memdisponible < sizeof(Nuevo)

Ffunc

Proced CrearNodo(ref L:Tlista, val X:tdato)

New(Nuevo)

Nuevo↑.sig 🡨 nil

Nuevo↑.info 🡨 x

Fproced

Proced Linsertarppio(Ref L:Tlista, val x:Tdato)

Var local: Nuevo: puntero

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 nuevo

Sino

Nuevo↑.sig 🡨 L.cab

l.cab 🡨 Nuevo

Fsi

Fproced

Proced Linsertarfin(ref L:Tlista, val x:Tdato)

Var local: Nuevo, aux: punter

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

Sino

Aux 🡨 l.cab

Mientras aux↑.sig != nil entonces

Aux 🡨 aux↑.sig

Fmientras

aux↑.sig 🡨 Nuevo

Fsi

Fproced

Proced Linsertarorden(ref l:Tlista, val x:Tdato, val orden: caracter)

Var local: Nuevo, aux: puntero

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

Sino

Si (x.clave > l.cab↑.info.clave y orden=’D’) y (x.clave < l.cab↑.info.clave y orden=’A’) entonces

Nuevo↑.sig 🡨 l.cab

l.cab 🡨 Nuevo

Sino

Aux 🡨 l.cab

Mientras (aux↑.sig != Nil) y ((aux↑.sig↑.info.clave > x.clave y orden = ‘D’) o (aux↑.sig↑.info.clave < x.clave y orden = ’A’)) hacer

Aux 🡨 aux↑.sig

Fmientras

Nuevo↑.sig 🡨 aux↑.sig

aux↑.sig 🡨 Nuevo

Fsi

Fsi

Fproced

Proced LBuscarOrdenado(Ref L:Tlista, val x:Tclave, ref existe: lógico)

Var local: aux: puntero

Existe 🡨 falso

Si l.cab != nil entonces

Si l.cab↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 l.cab

Sino

Aux 🡨 l.cab

Mientras (aux != nil) y (aux↑.info.clave = l.cab↑.info.clave) hacer

Aux 🡨 aux↑.sig

Fmientras

Si aux != nil entonces

Si l.cab↑.info.clave > aux↑.info.clave entonces

Mientras aux ! = nil y aux↑.info.clave > x hacer

Aux 🡨 Aux↑.sig

Fmientras

Sino

Mientras aux != nil y aux↑.info.clave < x hacer

Aux 🡨 aux↑.sig

Fmientras

Fsi

Si aux != nil y aux↑.info.clave = x entonces

l.actual 🡨 aux

Existe 🡨 verdadero

Fsi

Fsi

Fsi

Fsi

FProced

Proced Lborrarppio(ref L:Tlista)

Var local: aux: puntero

Aux 🡨 l.cab

l.cab 🡨 l.cab↑.sig

disponer(aux)

Fproced

Proced Lborrarfin(Ref L:Tlista)

Var local: aux: punter

Si l.cab↑.sig = nil entonces

Aux 🡨 l.cab

l.cab 🡨 nil

sino

t 🡨 l.cab

Mientras t↑.sig↑.sig != nil hacer

T 🡨 t↑.sig

Fmientras

Aux 🡨 t↑.sig

t↑.sig 🡨 nil

fsi

disponer(Aux)

Fproced

Proced LborrarActual(ref L:TLista)

Var local: aux: puntero

Aux 🡨 l.actual

Si l.cab = l.actual entonces

Si l.cab↑.sig = nil entonces

l.cab 🡨 nil

L.actual 🡨 nil

sino

l.cab 🡨 l.cab↑.sig

l.actual 🡨 l.actual↑.sig

fsi

sino

t 🡨 l.cab

mientras t↑.sig != l.actual hacer

t 🡨 t↑.sig

fmientras

t↑.sig 🡨 l.actual↑.sig

l.actual 🡨 l.actual↑.sig

Fsi

Disponer(Aux)

Fproced

Lista simplemente enlazada circular

Tlista = reg

Cab, actual: puntero

Freg

Nodo = reg

Sig: puntero

Info: Tdato

Freg

Proced LCrear(Ref l:Tlista)

L.cab 🡨 nil

Fproced

Func Lvacia(ref l:Tlista): lógico

Lvacia 🡨 l.cab = nil

Ffunc

Proced Lppio(ref L:Tlista)

l.actual 🡨 l.cab

Fproced

Proced Linfo(ref l:Tlista, Ref x:Tdato)

X 🡨 l.actual↑.info

Fproced

Proced Lsig(ref L:Tlista)

L.actual 🡨 l.actual↑.sig

Fproced

Func Lfin(ref l:Tlista): lógico

Lfin 🡨 l.actual = l.cab

Ffunc

Proced Lmodificar(ref L:Tlista, val x:TDato)

L.actual↑.info 🡨 x

Fproced

Func Lllena(ref L:tlista): lógico

Var local: nuevo:nodo

Lllena 🡨 memdisponible < sizeof(Nuevo)

Ffunc

Proced Linsertarppio(ref L:Tlista, val x:Tdato)

Var local: Nuevo, aux: puntero

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

l.cab↑.sig 🡨 l.cab

Sino

Aux 🡨 l.cab

Mientras aux↑.sig != l.cab hacer

Aux 🡨 aux↑.sig

Fmientras

aux↑.sig 🡨 Nuevo

Nuevo↑.sig 🡨 l.cab

l.cab 🡨 Nuevo

Fsi

FProced

Proced Linsertarfin(ref L:Tlista, val x:tdato)

Var local: Nuevo, aux: punter

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

l.cab↑.sig 🡨 l.cab

sino

aux 🡨 l.cab

Mientras aux↑.sig != l.cab hacer

Aux 🡨 aux↑.sig

Fmientras

aux↑.sig 🡨 Nuevo

nuevo↑.sig 🡨 l.cab

Fsi

Fproced

Proced Linsertarorden(ref L:Tlista, val x:Tdato, val orden: caracter)

Var local: Nuevo, aux: puntero

CrearNodo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 nuevo

l.cab↑.sig 🡨 l.cab

sino

si ((Nuevo↑.info.clave > x.clave y orden=’D’) o (Nuevo↑.info.clave < x.clave y orden=’A’ )) entonces

aux 🡨 l.cab

mientras aux↑.sig != l.cab hacer

aux 🡨 aux↑.sig

fmientras

aux↑.sig 🡨 Nuevo

nuevo↑.sig 🡨 l.cab

l.cab 🡨 Nuevo

Sino

Aux 🡨 l.cab

Mientras aux↑.sig != l.cab y ((aux↑.sig↑.info.clave > x.clave y orden=’D’) o (aux↑.sig↑.info.clave < x.clave y orden=’A’)) hacer

Aux 🡨 aux↑sig

Fmientras

Nuevo↑.sig 🡨 Aux↑.sig

aux↑.sig 🡨 Nuevo

Fsi

Fsi

Fproced

Proced Lbuscarordenado(ref L:Tlista, val x:tclave, Ref Existe: lógico)

Var local: t: puntero

Existe 🡨 falso

Si l.cab !=nil entonces

Si l.cab↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 l.cab

sino

t 🡨 l.cab↑.sig

Mientras t!= l.cab y t↑.info.clave = l.cab↑.info.clave hacer

T 🡨 t↑.sig

Fmientras

Si t != l.cab entonces

Si l.cab↑.info.clave > t↑.info.clave entonces

Mientras t != l.cab y t↑.info.clave > x hacer

t 🡨 t↑.sig

Fmientras

Sino

Mientras t != l.cab y t↑.info.clave < x hacer

t 🡨 t↑.sig

Fmientras

Fsi

Si t!= l.cab y t↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨t

fsi

fsi

fsi

fsi

Fproced

Proced Lborrarppio(Ref L:Tlista)

Var local: Aux, t: puntero

Aux 🡨 l.cab

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

sino

t 🡨 l.cab

mientras t↑.sig != l.cab hacer

t 🡨 t↑.sig

Fmientras

t↑.sig 🡨 l.cab↑.sig

l.cab 🡨 l.cab↑.sig

fsi

Disponer(Aux)

Fproced

Proced Lborrarfin(ref L:Tlista)

Var local: aux: puntero

Si l.cab↑.sig = l.cab entonces

Aux 🡨 l.cab

l.cab 🡨 nil

sino

t 🡨 l.cab

Mientras t↑.sig↑.sig != l.cab hacer

T 🡨 t↑.sig

Fmientras

Aux 🡨 t↑.sig

t↑.sig 🡨 l.cab

Fsi

Disponer(Aux)

Fproced

Proced LborrarActual(ref L:TLista)

Var local: aux: puntero

Aux 🡨 l.actual

Si l.actual = l.cab entonces

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

l.actual 🡨 nil

Sino

T 🡨 l.cab

Mientras t↑.sig != l.cab hacer

T 🡨 t↑.sig

Fmientras

t↑.sig 🡨 l.cab↑.sig

l.cab 🡨 l.cab↑.sig

l.actual 🡨 l.actual↑.sig

fsi

sino

t 🡨 l.cab

Mientras t↑.sig != l.actual hacer

T 🡨 t↑.sig

Fmientras

T↑.sig 🡨 l.actual↑.sig

l.actual 🡨 l.actual↑.sig

Fsi

Disponer(Aux)

Fproced

Lista doblemente enlazada

Tlista = reg

Cab, actual: puntero

Freg

Nodo = reg

Ant, sig: puntero

Info: Tdato

Freg

Proced lcrear(ref L:Tlista)

L.cab 🡨 nil

Fproced

Func Lvacia(Ref L:Tlista): lógico

Lvacia 🡨 l.cab = nil

FFunc

Proced Lppio(ref L:Tlista)

L.actual 🡨 l.cab

Fproced

Proced Linfo(ref L:Tlista, Val x:tdato)

X 🡨 l.actual↑.info

FProced

Proced lSig(ref L:Tlista)

L.actual 🡨 l.actual↑.sig

Fproced

Func Lfin(ref L:Tlista): lógico

LFin 🡨 l.actual = nil

Ffunc

Proced Lmodificar(ref L:Tlista, val x:Tdato)

l.actual↑.info 🡨 x

FProced

Func Lllena(ref L:Tlista): lógico

Var local: Nuevo: Nodo

Lllena 🡨 memdisponible < Sizeof(Nuevo)

Func

Proced Linsertarppio(Ref L:Tlista, val x:Tdato)

Var local: Nuevo: puntero

CrearNodoDoble(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

sino

Nuevo↑.sig 🡨 L.cab

l.cab↑.ant 🡨 Nuevo

l.cab 🡨 Nuevo

Fsi

Fproced

Proced Linsertarfin(ref L:Tlista, val x:Tdato)

Var local: Nuevo, Aux: punter

CrearNodoDoble(Nuevo, x)

Si L.cab = nil entonces

l.cab 🡨 Nuevo

Sino

Aux 🡨 l.cab

Mientras aux↑.sig != nil hacer

Aux 🡨 aux↑.sig

Fmientras

aux↑.sig 🡨 Nuevo

Nuevo↑.ant 🡨 aux

Fsi

Fproced

Proced Linsertarorden(Ref L:Tlista, val x:tdato, val orden:caracter)

Var local: Nuevo, aux: puntero

CrearNodoNuevo(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨 Nuevo

Sino

Si (Nuevo↑.info.clave > x.clave y orden=’D’) o

(Nuevo↑.info.clave < x.info.clave y orden=’A’) entonces

nuevo↑.sig 🡨 l.cab

l.cab↑.ant 🡨 Nuevo

l.cab 🡨 Nuevo

sino

Aux 🡨 l.cab

Mientras (aux↑.sig != nil) y ((aux↑.sig↑.info.clave > x.clave y orden=’D’) o (aux↑.sig↑.info.clave < x.clave y orden=’A’)) hacer

Aux 🡨 aux↑.sig

Fmientras

Nuevo↑.sig 🡨 aux↑.sig

Si aux↑.sig != nil entonces

aux↑.sig↑.ant 🡨 Nuevo

fsi

nuevo↑.ant 🡨 Aux

aux↑.sig 🡨Nuevo

Fsi

Fsi

Fproced

Proced Lbuscarordenado(ref L:Tlista, val x:tdato, ref Existe: lógico)

Var local: nuevo: puntero

Existe 🡨 falso

Si l.cab != nil entonces

Si l.cab↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 l.cab

Sino

T 🡨 l.cab

Mientras (t!= nil) y (t↑.info.clave = l.cab↑.info.clave) hacer

T 🡨 t↑.sig

Fmientras

Si t != nil entonces

Si l.cab↑.info.clave > t↑.info.clave entonces

Mientras t != nil y t↑.info.clave > x hacer

T 🡨 t↑.sig

Fmientras

Sino

Mientras t!= nil y t↑.info.clave < x hacer

T 🡨 t↑.sig

Fmientras

Fsi

Si t!= nil y t↑.info.clave = x entonces

Existe 🡨 verdadero

L.actual 🡨 t

Fsi

Fsi

Fsi

Fsi

FProced

Proced Lborrarppio(Ref l:Tlista)

Var local: aux: puntero

Aux 🡨 l.cab

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

sino

l.cab↑.sig↑.ant 🡨 nil

l.cab 🡨 l.cab↑.sig

Fsi

disponer(Aux)

Fproced

Proced Lborrarfin(ref L:tlista)

Var local: Aux: punter

Si l.cab↑.sig = nil entonces

Aux 🡨 l.cab

l.cab 🡨 nil

sino

t 🡨 l.cab

Mientras t↑.sig↑.sig != nil hacer

T 🡨 t↑.sig

Fmientras

Aux 🡨 t↑.sig

t↑.sig 🡨 nil

Fsi

Disponer(Aux)

Fproced

Proced LborrarActual(ref L:Tlista)

Var local: aux: puntero

Aux 🡨 l.actual

Si l.actual = l.cab entonces

Si l.cab↑.sig = nil entonces

l.cab 🡨 nil

l.actual 🡨 nil

sino

l.cab↑.sig↑.ant 🡨 nil

l.cab 🡨 l.cab↑.sig

l.actual 🡨 l.actual↑.sig

fsi

Sino

Si l.actual↑.sig != nil entonces

l.actual↑.sig↑.ant 🡨 l.actual↑.ant

fsi

l.atual↑.ant↑.sig 🡨 l.actual↑.sig

fsi

disponer(Aux)

Fproced

Lista doblemente enlazada circular

Tlista = reg

Actual, cab: puntero

Freg

Nodo = reg

Ant, sig: puntero

Info: Tdato

Freg

Proced Lcrear(ref L:Tlista)

L.cab 🡨 nil

Fproced

Func Lvacia(Ref L:Tlista): lógico

Lvacia 🡨 l.cab = nil

Ffunc

Proced Lppio(Ref L:Tlista)

l.actual 🡨 l.cab

Fproced

Proced Linfo(Ref L:Tlista, ref x:Tdato)

X 🡨 l.actual↑.info

Fproced

Proced Lsig(ref L:TLista)

L.actual 🡨 l.actual↑.sig

Fproced

Func lfin(ref L:Tlista): lógico

Lfin 🡨 l.actual = l.cab

Ffunc

Proced Lmodificar(Ref L:Tlista, val x:TDato)

L.actual↑.info 🡨 x

Fproced

Func Lllena(ref L:Tlista): lógico

Var local: Nuevo: nodo

Lllena 🡨 memdisponible < sizeof(nuevo)

Ffunc

Proced Linsertarppio(ref L:Tlista, val x:Tdato)

Var local: nuevo: puntero

CrearNodoDoble(Nuevo, x)

Si l.cab = nil entonces

l.cab 🡨Nuevo

l.cab↑.sig 🡨 l.cab

l.cab↑.ant 🡨 l.cab

sino

l.cab↑.ant↑.sig 🡨 Nuevo

Nuevo↑.ant 🡨 l.cab↑.ant

Nuevo↑.sig 🡨 l.cab

l.cab↑.ant 🡨 Nuevo

l.cab 🡨 Nuevo

Fsi

Fproced

Proced Linsertarfin(ref L:Tlista, val x:tdato)

Var local: Nuevo: puntero

Si l.cab = nil entonces

l.cab 🡨 Nuevo

l.cab↑.sig 🡨 l.cab

l.cab↑.ant 🡨 l.cab

sino

l.cab↑.ant↑.sig 🡨 Nuevo

Nuevo↑.ant 🡨 l.cab↑.ant

l.cab↑.ant 🡨 Nuevo

Nuevo↑.sig 🡨 l.cab

Fsi

Fproced

Proced Linsertarorden(ref L:Tlista, val x:Tdato, val orden: caracter)

Var local: Nuevo: puntero

CrearNodoDoble(Nuevo, x)

Si l.cab = nil entonces

l.cab🡨 Nuevo

l.cab↑.sig 🡨 Nuevo

l.cab↑.ant 🡨 Nuevo

sino

si (x.clave > l.cab↑.info.clave y orden = ‘D’) o (x.clave < l.cab↑.info.clave y orden = ‘A’) entonces

l.cab↑.ant↑.sig 🡨 Nuevo

Nuevo↑.ant 🡨 L.cab↑.ant

Nuevo↑.sig 🡨 l.cab

l.cab↑.ant 🡨 Nuevo

l.cab 🡨 Nuevo

sino

t 🡨 l.cab

Mientras t↑.sig != l.cab y ((t↑.sig↑.info.clave > x.clave y orden=’D’) o (t↑.sig↑.info.clave < x.clave y orden = ‘A’)) entonces

T 🡨 t↑.sig

Fmientras

Nuevo↑.sig 🡨 t↑.sig

t↑.sig↑.ant 🡨 Nuevo

nuevo↑.ant 🡨 t

t↑.sig 🡨 Nuevo

Fsi

Fsi

FProced

Proced Lbuscarordenado(Ref L:Tlista, val x:Tclave, Ref Existe: lógico)

Var local: pr, ult:puntero, orden:carácter

Existe 🡨falso

Si l.cab!= nil entonces

Si l.cab↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 l.cab

sino

pr 🡨 l.cab

Ult 🡨 l.cab↑.ant

Si pr↑.info.clave > ult↑.info.clave entonces

Mientras (Pr↑.info.clave > ult↑.info.clave) y (pr↑.info.clave > x) y (ult↑.info.clave < x) hacer

Pr 🡨 pr↑.sig

Ult 🡨 ult↑.ant

Fmientras

Sino

Mientras (pr↑.info.clave < ult↑.info.clave) y (Pr↑.info.clave < x) y (ult↑.info.clave > x) hacer

Pr 🡨 pr↑.sig

Ult 🡨 ult↑.sig

Fmientras

Fsi

Si pr↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 pr

Sino

Si ult↑.info.clave = x entonces

Existe 🡨 verdadero

l.actual 🡨 ult

fsi

fsi

fsi

fsi

Fproced

Proced LborrarPpio(Ref L:Tlista)

Var local: aux: puntero

Aux 🡨 l.cab

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

sino

l.cab↑.ant↑.sig 🡨 l.cab↑.sig

l.cab↑.sig↑.ant 🡨 l.cab↑.ant

l.cab 🡨 l.cab↑.sig

fsi

disponer(Aux)

Fproced

Proced Lborrarfin(Ref L:Tlista)

Var local: aux: puntero

Aux 🡨 l.cab↑.ant

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

sino

aux↑.ant↑.sig 🡨 l.cab

l.cab↑.ant 🡨 aux↑.ant

fsi

disponer(aux)

Fproced

Proced Lborraractual(ref L:Tlista)

Var local: aux: puntero

Aux 🡨 l.actual

Si l.cab = l.actual entonces

Si l.cab↑.sig = l.cab entonces

l.cab 🡨 nil

l.actual 🡨 nil

sino

l.cab↑.ant↑.sig 🡨 l.cab↑.sig

l.cab↑.sig↑.ant 🡨 l.cab↑.ant

l.cab 🡨 l.cab↑.sig

l.actual 🡨 l.actual↑.sig

fsi

sino

aux↑.ant↑.sig 🡨 l.actual↑.sig

aux↑.sig↑.ant 🡨 l.actual↑.ant

l.actual 🡨 l.actual↑.sig

fsi

disponer(Aux)

Fproced

TDA FECHA

Gregoriano Registro

Tfecha = Reg

Dia: entero

Mes: entero

Anio: entero

Freg

Func Fdia(ref F:Tfecha): entero

Fdia 🡨 f.dia

Ffunc

Func Fmes(Ref F:Tfecha): entero

Fmes 🡨 f.mes

Ffunc

Func Fanio(ref F:Tfecha): entero

Fanio 🡨 f.anio

FFunc

Proced Fsumardias(Ref F:tfecha, val n: entero)

Var local: v: entero

Si N > 0 entonces

f.dia 🡨 f.dia + n

Mientras f.dia > cantdias(f.mes, f.anio) hacer

f.dia 🡨 f.dia – cantdias(f.mes, f.anio)

f.mes 🡨 f.mes + 1

si f.mes = 13 entonces

f.anio 🡨 f.anio + 1

f.mes 🡨 1

fsi

fmientras

sino

v 🡨 n\*(-1)

Mientras V > 0 hacer

f.dia 🡨 f.dia – 1

si f.dia = 0 entonces

f.mes 🡨 f.mes – 1

si f.mes = 0 entonces

f.anio 🡨 f.anio - 1

f.mes 🡨 12

fsi

f.dia 🡨 cantdias(F.mes, f.anio)

fsi

v 🡨 v – 1

fmientras

fsi

Fproced

Func CantDias(val m, a: entero): entero

Var local: días: entero

Si m = 4 o m = 6 o m= 9 o m= 11 entonces

Días 🡨 30

Sino

Si m = a entonces

Si EsBisiesto(a) entonces

Días 🡨 29

Sino

Días 🡨 28

Fsi

Sino

Dias 🡨 31

Fsi

Fsi

CantDias 🡨 días

Ffunc

Func Esbisiesto(val a: entero): lógico

Var local: es: lógico

Es 🡨 falso

Si (a resto 400 = 0) o (a resto 4 = 0 y a resto 100 ¡= 0) entonces

Es 🡨 verdadero

Fsi

EsBisiesto 🡨 es

Ffunc

Func Fválida(Ref F:Tfecha): lógico

Var local: m: arreglo[12] de enteros, es: lógico

Es 🡨 falso

Cargar(m, f)

Si f.anio > 1500 entonces

Si F.mes >= 1 y F.mes <= 12 entonces

Si F.dia >= 1 y F.dia <= m[F.mes] entonces

Es 🡨 verdadero

Fsi

Fsi

Fsi

Fvalida 🡨 es

Ffunc

Proced Cargar(Ref m: arreglo[12] de enteros, ref F:Tfecha)

Si (f.anio resto 400 = 0) o (F.anio resto 4 = 0 y f.anio Resto 100 ¡= 0) entonces

M[2] 🡨 29

Sino

M[2] 🡨 28

Fsi

M[1] 🡨 31; M[3] 🡨 31; M[5] 🡨 31; M[7] 🡨 31; M[8] 🡨31; M[10] 🡨 31; M[12] 🡨 31; M[4] 🡨 30; M[6] 🡨 30; M[6] 🡨 30;

M[11] 🡨 30

Fproced

Func FdiferenciaDias(Ref F1,F2: Tfecha): entero

Var local: Fmayor, Fmenor: entero

Fmayor 🡨 SuperJuliano(F1)

Fmenor 🡨 SuperJuliano(F2)

FDiferenciaDias 🡨 Fmayor – Fmenor

Ffunc

Func superjuliano(ref F: tfecha): entero

Var local: anioAux, mesAux, diaAux: entero

diaAux 🡨 0

mesaux 🡨 1

anioaux 🡨 1

mientras anioaux < f.anio hacer

si esbisiesto(anioAux) entonces

diaaux 🡨 diaaux + 366

sino

diaaux 🡨 diaaux + 365

fsi

anioaux 🡨 anioaux + 1

fmientras

mientras mesaux < f.mes hacer

diaaux 🡨 diaaux + cantdias(mesaux, f.anio)

mesaux 🡨 mesaux + 1

Fmientras

Diaaux 🡨 diaaux + f.dia

Superjuliano 🡨 diaaux

Ffunc

Proced FconvertirFlegibleTfecha(Val Fent: Flegible, Ref Fsal:Tfecha, ref error: lógico)

Convertir(Subcadena(Fent, 1,2), fsal.dia, error)

Si No error entonces

Convertir(Subcadena(Fent, 4, 2), fsal.mes, error)

Si no error entonces

Convertir(Subcadena(fent, 7, 4), fsal.anio, error)

Fsi

Fsi

Fproced

Proced FConvertirTfechaFlegible(ref Fent:Tfecha, ref Fsal: flegible)

Si fent.dia < 10 entonces

Fsal 🡨 ‘0’ + str(fent.dia) + ‘/’

Sino

Fsal 🡨 str(fent.dia) + ‘/’

Fsi

Si fent.mes < 10 entonces

Fsal 🡨 fsal + ‘0’ + str(fent.mes) + ‘/’

Sino

Fsal 🡨 fsal + str(fent.mes) + ‘/’

Fsi

Fsal 🡨 Fsal + str(fent.anio)

Fproced

Func Fmayor(ref F1, F2:tfecha): lógico

Si f1.anio > f2.anio entonces

Fmayor 🡨 verdadero

Sino

Si f1.anio < f2.anio entonces

Fmayor 🡨 falso

Sino

Si f1.mes > f2.mes entonces

Fmayor 🡨 verdadero

Sino

Si f1.mes < f2.mes entonces

Fmayor 🡨falso

Sino

Si f1.dia > f2.dia entonces

Fmayor 🡨 verdadero

Sino

Fmayor 🡨 falso

Fsi

Fsi

Fsi

Fsi

Fsi

Fproced

Gregoriano Cadena

Tfecha = cadena[8] // ‘aaaammdd’

Func Fdia(ref F:Tfecha): entero

Var local: idia: entero, error: lógico

Convertir(subcaden(f, 7,2), idia, error)

Fdia 🡨 idia

Ffunc

Func Fmes(Ref F:Tfecha): entero

Var local: imes: entero, error: lógico

Convertir(Subcadena(f, 5, 2), imes, error)

Fmes 🡨 imes

Ffunc

Func Fanio(ref F:tfecha): entero

Var local: ianio: entero, error: lógico

Convertir(subcadena(f, 1, 4), ianio, error)

Fanio 🡨 ianio

Ffunc

Func Fmayor(ref F1, F2:Tfecha): lógico

Fmayor 🡨 F1 > F2

Ffunc

Func Fvalida(ref F:Tfecha): lógico

Var local: M: arreglo[12] de enteros, ianio, imes, idia: entero, es: lógico

Es 🡨 falso

Convertir(Subcadena(1, 4), ianio, error)

SI no error entonces

Convertir(Subcadena(5, 2), imes, error)

Si No error entonces

Convertir(Subcadena(7, 2), idia, error)

Si no error entonces

CargarCadena(m, f)

Si ianio > 1500 entonces

Si imes >= 1 y imes <= 12 entonces

Si idia >= 1 y idia <= m[f.mes] entonces

Es 🡨 verdadero

Fsi

Fsi

Fsi

Fsi

Fsi

Fsi

Fvalida 🡨 Es

FProced

Proced CargarCadena(Ref m: arreglo[12] de enteros, ref F:Tfecha)

Var local: ianio: entero, error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

Si (ianio Resto 400 = 0) y (ianio Resto 4 = 0 y ianio resto 100 ¡= 0) entonces

M[2] 🡨 29

Sino

M[2] 🡨 28

Fsi

M[1] 🡨 31; M[3] 🡨 31; M[5] 🡨 31; M[7] 🡨 31; M[8] 🡨31; M[10] 🡨 31; M[12] 🡨31; M[4] 🡨30; M[6] 🡨 30; M[9] 🡨 30; M[11] 🡨 30

Fproced

Proced sumardias(ref F:tfecha, val n:entero)

Var local: imes, ianio, idia, v: entero, error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

Convertir(subcadena(f, 5, 2), imes, error)

Convertir(subcadena(f, 7, 2), idia, error)

Si n > 0 entonces

Idia 🡨 idia + n

Mientras idia > cantdias(Imes, ianio) hacer

Idia 🡨 idia – cantdias(imes, ianio)

Imes 🡨 imes + 1

Si imes = 13 entonces

Ianio 🡨 ianio + 1

Imes 🡨 1

Fsi

Fmientras

Sino

V 🡨 n \* (- 1)

Mientras v > 0 hacer

Idia 🡨 idia – 1

Si idia = 0 entonces

Imes 🡨 imes – 1

Si imes = 0 entonces

Ianio 🡨 ianio – 1

Imes 🡨12

Fsi

Idia 🡨 cantdias(imes, ianio)

Fsi

V 🡨 v – 1

Fmientras

Fsi

F 🡨 str(ianio)

Si imes < 10 entonces

F 🡨 f + ‘0’ + str(imes)

Sino

F 🡨 f + str(imes)

Fsi

Si idia < 10 entonces

F 🡨 f + ‘0’ + str(idia)

Sino

F 🡨 f + str(idia)

Fsi

Fproced

Proced FconvertirFlegibleTfecha(val fent: flegible, ref Fsal:Tfecha, ref error: lógico)

Var local: error: lógico, imes, ianio, idia: entero

Convertir(subcadena(fent, 7, 4), ianio, error)

Si No error entonces

Convertir(Subcadena(fent, 4, 2), imes, error)

Si No error entonces

Convetir(Subcadena(fent, 1, 2), idia, error)

Si no Error entonces

Fsal 🡨 subcadena(fent, 7, 4) + subcadena(fent, 4, 2) + subcadena(fent, 1, 2)

Fsi

Fsi

Fsi

Fproced

Proced FconvertirTfechaFlegible(Ref Fent:Tfecha, ref Fsal: flegible)

Fsal 🡨 subcadena(fent, 7, 2) + ‘/’ + subcadena(fent, 5, 2) + ‘/’ + subcadena(fent, 1, 4)

Fproced

Func FdiferenciaDias(ref F1, F2: tfecha): entero

Var local: DiaMayor, DiaMenor: entero

DiaMayor 🡨 SuperJulianoCadena(F1)

DiaMenor 🡨 SuperJulianoCadena(F2)

Fdiferenciadias 🡨 DiaMayor - DiaMenor

Ffunc

Func SuperJulianoCadena(ref F:Tfecha): entero

Var local: imes, idia, ianio, diaAux, mesAux, AnioAux: entero, error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

Convertir(subcadena(f, 5, 2), imes, error)

Convertir(subcadena(f, 7, 2), idia, error)

diaAux 🡨 0

mesAux 🡨 1

AnioAux 🡨 1

Mientras anioAux < ianio hacer

Si esbisiesto(anioAux) entonces

DiaAux 🡨 diaAux + 366

Sino

DiaAux 🡨 diaAux + 365

Fsi

Anioaux 🡨 anioaux + 1

Fmientras

Mientras mesAux < imes hacer

DiaAux 🡨 diaAux + cantdias(mesaux, ianio)

mesAux 🡨 mesAux + 1

fmientras

diaAux 🡨 diaAux + idia

superjulianocadena 🡨 diaaux

Ffunc

Gregoriano Registro

Tfecha = reg

Dia: entero

Mes: entero

Anio: entero

Freg

Func Fdia(ref F:Tfecha): entero

Fdia 🡨f.dia

Ffunc

Func Fmes(ref F:Tfecha): entero

Fmes 🡨 f.mes

Ffunc

Func Fanio(ref F:Tfecha): entero

Fanio 🡨 f.anio

Ffunc

Proced Sumardias(ref F:tfecha, val n:entero)

Var local: v: entero

Si n > 0 entonces

f.dia 🡨 f.dia + n

mientras f.dia > cantdias(f.mes, f.anio) hacer

f.dia 🡨 f.dia – cantdias(f.mes, f.anio)

f.mes 🡨 f.mes + 1

si f.mes = 13 entonces

f.anio 🡨 f.anio + 1

f.mes 🡨 1

fsi

fmientras

sino

v 🡨 n\*(-1)

mientras V > 0 hacer

f.dia 🡨 f.dia – 1

si f.dia = 0 entonces

f.mes 🡨 f.mes – 1

si f.mes = 0 entonces

f.anio 🡨 f.anio

f.mes 🡨 12

fsi

f.dia 🡨 cantdias(f.mes, f.anio)

fsi

v 🡨 v – 1

fmientras

fsi

Fproced

Func CantDias(val m, a: entero): entero

Var locales: días: entero

Si m=4 o m=6 o m=9 o m=11 entonces

Días 🡨 30

Sino

Si m= 2 entonces

Si EsBisiesto(a) entonces

Días 🡨 29

Sino

Días 🡨 28

Fsi

Sino

Dias 🡨 31

Fsi

Fsi

CantDias 🡨 días

Ffunc

Func EsBisiesto(ref F:Tfecha): lógico

Var local: es: lógico

Es 🡨 falso

Si (a resto 400 = 0) o (a resto 4 = 0 y a resto 100 ¡= 0) entonces

Es 🡨 verdadero

Fsi

EsBisiesto 🡨 es

Ffunc

Func EsValida(ref F:tfecha): lógico

Var local: m: arreglo[12] de enteros, es: lógico

Es 🡨 falso

Cargar(m, f)

Si F.anio > 1500 entonces

Si f.mes >= 1 y f.mes <= 12 entonces

Si f.dia >= 1 y f.dia <= m[F.mes] entonces

Es 🡨 verdadero

Fsi

Fsi

Fsi

EsValida 🡨 Es

Ffunc

Proced Cargar(ref m: arreglo[12] de enteros, Ref F:Tfecha)

Si (f.anio resto 400 = 0) o (f.anio Resto 4 = 0 y f.anio resto 100 ¡= 0) entonces

M[2] 🡨 29

Sino

M[2] 28

Fsi

M[1] 🡨 31; m[3] 🡨 31; m[5] 🡨 31; m[7] 🡨 31; m[8] 🡨 31; m m[10] 🡨31; m[12] 🡨 31; m[4] 🡨 30; m[6] 🡨30; m[9] 🡨 30;

M[11] 🡨 30

Fproced

Funcion FdiferenciaDias(ref F1, F2: entero): entero

Var local: fmayor, fmenor

Fmayor 🡨 superjuliano(f1)

Fmenor 🡨 superjuliano(f2)

FdiferenciaDias 🡨 fmayor – fmenor

Ffunc

Func SuperJuliano(ref F:Tfecha): entero

Var local: diaAux, mesAux, anioAux: entero

diaAux 🡨0

mesAux 🡨 1

AnioAux 🡨 1

Mientras anioAux < f.anio hacer

Si EsBisiesto(Anioaux) entonces

DiaAux 🡨 DiaAux + 366

Sino

DiaAux 🡨 DiaAux + 365

Fsi

AnioAux 🡨 AnioAux + 1

Fmientras

Mientras mesAux < f.mes hacer

DiaAux 🡨 DiaAux + cantDias(MesAux, f.anio)

mesAux 🡨 mesAux + 1

Fmientras

DiaAux 🡨 DiaAux + f.dia

SuperJuliano 🡨 DiaAux

Ffunc

Proced FconvertirflegibleTfecha(val Fent: flegible, ref fsal: Tfecha, ref Error: lógico)

Convertir(Subcadena(fent, 7, 4), fsal.anio, error)

Si no Error entonces

Convetir(subcadena(fent, 4, 2), fsal.mes, error)

Si no Error entonces

Convertit(subcadena(fent, 1, 2), fsal.dia, error)

Fsi

Fsi

Fproced

Proced FconvertirTfechaFlegible(ref Fent:Tfecha, ref fsal: flegible)

Si fent.dia < 10 entonces

Fsal 🡨 ‘0’ + str(fent.dia) + ‘/’

Sino

Fsal 🡨 str(fent.dia) + ‘/’

Fsi

Si fent.mes < 10 entonces

Fsal 🡨 fsal + ‘0’ + str(fent.mes) + ‘/’

Sino

Fsal 🡨 fsal + str(fent.mes) + ‘/’

Fsi

Fsal 🡨 fsal + str(fent.anio)

Fproced

Func Fmayor(ref F1, F2: Tfecha): lógico

Si F1.anio > F2.anio entonces

Fmayor 🡨 verdadero

Sino

Si F1.anio < F2.anio entonces

Fmayor 🡨 falso

Sino

Si f1.mes > f2.mes entonces

Fmayor 🡨 verdadero

Sino

Si F1.mes < F2.mes entonces

Fmayor 🡨 falso

Sino

Si f1.dia > f2.dia entonces

Fmayor 🡨 verdadero

Sino

Fmayor 🡨 falso

Fsi

Fsi

Fsi

Fsi

Fsi

Fproced

Gregoriano cadena

Tfecha = cadena[8] / ‘aaaammdd’

Func Fdia(ref F:Tfecha): entero

Var local: idia: entero, error: lógico

Convertir(subcadena(f, 7, 2), idia, error)

Fdia 🡨 idia

Ffunc

Func Fmes(Ref F:Tfecha): entero

Var local: imes: entero, error: lógico

Convertir(subcadena(f, 5, 2), imes, error)

Fmes 🡨 imes

Ffunc

Func Fanio(ref F:Tfecha): entero

Var local: ianio: entero, error: Lógico

Convertir(subcadena(f, 1, 4), ianio, error)

Fanio 🡨 f.anio

Ffunc

Func Fmayor(ref F1,F2:Tfecha): lógico

Fmayor 🡨 F1 > F2

Ffunc

Func Fvalida(ref F:Tfecha): lógico

Var local: m: arreglo[12] de enteros, ianio, imes, idia: entero, es: lógico

ES 🡨 falso

Convertir(subcadena(f, 1, 4), ianio, error)

Si No error entonces

Convertir(subcadena(f, 5, 2), imes, error)

Si No error entonces

Convertir(Subcadena(f, 7, 2), idia, error)

Si No error entonces

CargarCadena(m, f)

Si ianio > 1500 entonces

Si imes >= 1 y imes <= 12 entonces

Si idia >= 1 y idia <= m[imes] entonces

Es 🡨 verdadero

Fsi

Fsi

Fsi

Fsi

Fsi

Fsi

Fvalida 🡨 es

Fproced

Proced cargarcadena(ref M: arreglo[12] de enteros, Ref F:Tfecha)

Var local: ianio: entero; error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

si (ianio resto 400 = 0) o (ianio resto 4 = 0 y ianio resto 100 ¡= 0) entonces

m[2] 🡨 29

sino

m[2] 🡨 28

fsi

m[1] 🡨 31; m[3] 🡨 31; m[5] 🡨31; m[7] 🡨 31; m[8] 🡨 31; m[10] 🡨 31; m[12] 🡨 31; m[4] 🡨 30; m[6] 🡨 30; m[9] 🡨 30; m[11] 🡨 31

Fproced

Proced SumarDias(ref F:tfecha, val n: entero)

Var local: v, idia, imes, ianio: entero, error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

Convertir(subcadena(f, 5, 2), imes, error)

Convertir(Subcadena(f, 7, 2), idia, error)

Si n > 0 entonces

Idia 🡨 idia + n

Mientras idia > cantdias(imes, ianio) hacer

Idia 🡨 idia – cantdias(imes, ianio)

Imes 🡨 imes + 1

Si imes = 13 entonces

Ianio 🡨 ianio + 1

Imes 🡨 1

Fsi

Fmientras

Sino

V 🡨 n \* (-1)

Mientras V > 0 hacer

Idia 🡨 idia - 1

Si idia = 0 entonces

Imes 🡨 imes – 1

Si imes = 0 entonces

Ianio 🡨 ianio – 1

Imes 🡨 12

Fsi

Idia 🡨 cantdias(imes, ianio)

Fsi

V 🡨 v – 1

Fmientras

Fsi

F 🡨 str(ianio)

Si imes < 10 entonces

F 🡨 f + ‘0’ + str(imes) + ‘/’

Sino

F 🡨 f + str(imes) + ‘/’

Fsi

Si idia < 10 entonces

F 🡨 f + ‘0’ + str(idia)

Sino

F 🡨 f + str(idia)

Fsi

Fproced

Proced FconvertirFlegibleTfecha(Val Fent:flegible, ref Fsal:tfecha, Ref Error: lógico)

Var local: ianio, imes, idia: entero, error: lógico

Convertir(subcadena(fent, 7, 4), ianio, error)

Si No error entonces

Convertir(Subcadena(fent, 4, 2), imes , error)

Si no error entonces

Convertir(subcadena(fent, 1, 2), idia, error)

Si no error entonces

Fsal 🡨 Subcadena(fent, 7, 4) + subcadena(fent, 4, 2) + subcadena(fent, 1, 2)

Fsi

Fsi

Fsi

Fproced

Proced FconvertirTfechaFlegible(ref Fent:tfecha, ref Fsal: Flegible)

Fsal 🡨 subcadena(fent, 7, 2) + ‘/’ + subcadena(fent, 5,2) + ‘/’ +

Subcadena(fent, 1, 4)

Fproced

Func FdiferenciaDias(Ref F1, F2: Tfecha): entero

Var local: Fmayor, Fmenor: entero

Fmayor 🡨 SuperJulianoCadena(F1)

Fmenor 🡨 SuperJulianoCadena(F2)

FdiferenciaDias 🡨 Fmayor – Fmenor

Ffunc

Func SuperJulianoCadena(ref F:tfecha): entero

Var local: idia, imes, ianio, diaaux, mesaux, anioaux: entero, error: lógico

Convertir(Subcadena(f, 1, 4), ianio, error)

Convertir(subcadena(f, 5, 2), imes, error)

Convertir(subcadena(f, 7, 2), idia, error)

DiaAux 🡨 0

MesAux 🡨 1

AnioAux 🡨 1

Mientras AnioAux < ianio hacer

Si esBisiesto(anioAux) entonces

diaAux 🡨 diaAux + 366

sino

DiaAux 🡨 diaAux + 365

Fsi

anioAux 🡨 anioAux + 1

Fmientras

Mientras mesAux < imes hacer

DiaAux 🡨 DiaAux + cantDias(MesAux, ianio)

MesAux 🡨 MesAux + 1

Fmientras

DiaAux 🡨 diaAux + idia

SuperJulianoCadena 🡨 DiaAux

Ffunc