INFORMÁTICA APLICADA curso 2006-2007
PRÁCTICA 4: ESTRUCTURAS

Los objetivos de esta práctica son:

(1) Declarar estructuras en lenguaje C de forma correcta. Esto nos permitirá definir nuevos tipos de datos que no existen en el lenguaje.

(2) Utilizar de forma correcta las estructuras definidas. Esta posibilidad del lenguaje nos permite manejar información abstracta de forma más sencilla.

Introducción

En esta práctica nos proponemos realizar un programa en C que sea capaz de avisarnos de fechas señaladas con antelación (p.e. cumpleaños, aniversarios…). Para ello utilizaremos los conceptos de estructuras en C vistos en teoría. Además, haremos uso de las funciones de la librería estándar de C `<time.h>` que nos permitirán conocer la fecha y hora actual desde nuestro programa. Finalmente, se usarán también las funciones de E/S de ficheros que ya conocemos de prácticas anteriores.

Paso 1. Definir la estructura de datos a utilizar

Vamos a empezar por declarar la *estructura de datos* necesaria para almacenar la siguiente información relativa a una persona:

- Nombre (máximo 50 caracteres).
- Dirección (máximo 50 caracteres).
- Año de nacimiento (tres enteros).
- Gustos y aficiones (máximo 50 caracteres).

Para hacer esto, podemos definir la siguiente estructura de datos en C:

```
#define TAMMAX 50
struct amigo{
        char nombre[TAMMAX];
        char direccion[TAMMAX];
        int dia, mes, anyo; //fecha de nacimiento
        char aficiones[TAMMAX];
};
```

Como se puede observar, dentro de esta estructura están definidas las siguientes variables:

- **char nombre[TAMMAX]**: Es un array de caracteres (cadena de caracteres). Almacena el nombre del amigo (p.e. “Pedro López Martínez”).
- **char direccion[TAMMAX]**: Es un array de caracteres. Almacena la dirección del amigo (p.e. “c/Antonio Machado 45”).
- **int dia, mes y anyo:** Día, mes y año de nacimiento.
- **char aficiones[TAMMAX]**: Es un array de caracteres. Almacena los gustos de la persona (p.e. Música pop, atletismo…).

Definiremos la estructura amigo en el fichero de cabecera de la práctica, en la sección: “`Definición de tipos de datos y clases`”.

Paso 2. Uso de la estructura

Para almacenar la información relativa a varias personas, podemos declarar diferentes variables de tipo `amigo`.

```
int main(void)
{
        struct amigo amigo1, amigo2;
        printf("\nNombre del alumno: ");
        gets(amigo1.nombre);
        printf("\nDireccion: ");
        gets(amigo1.direccion);
        …
}
```

Sin embargo, si necesitamos almacenar un mayor número de elementos de tipo `struct amigo` esta no es la manera más conveniente de hacerlo. Para ello podemos utilizar un *array de estructuras*. Lo podremos declarar de la siguiente manera:

```
int main(void)
{
        struct amigo amigos[50];
        printf("\nNombre: ");
        gets(amigos[i].nombre);
        printf("\nDireccion: ");
        gets(amigo1.direccion);
        …
}
```

Escribir `struct amigo` cada vez que deseamos una variable de ese tipo es bastante tedioso. Es práctica común utilizar la sintaxis `typedef` para utilizar un nombre más corto.

```
#define TAMMAX 50
struct amigo{
        char nombre[TAMMAX];
        char direccion[TAMMAX];
        char fecha[TAMMAX];
        int dia, mes, anyo;
        char aficiones[TAMMAX];
};
typedef struct amigo amigo; // typedef nombreAntiguo nombreNuevo;
```

Hecho esto, podremos declarar variables de tipo `struct amigo` de la siguiente manera:

```
int main(void)
{
        amigo amigo1;
        amigo amigos[50];

        printf("\nNombre: ");
        gets(amigos[i].nombre); //Recoge el nombre por teclado
        printf("\nDireccion: ");
        gets(amigo1.direccion);//Recoge la dirección por teclado
```

```
        …
}
```

Nótese que, en el código anterior, hemos limitado el número de variables de tipo amigo a 50. Si queremos manejar un número variable de estructuras, deberemos utilizar un puntero y las funciones para reserva de memoria dinámica.

```
int main(void)
{
        amigo *pa;
        int n;

        printf("\nIntroduzca el número de estructuras a almacenar: ");
        scanf(“%d”, &n);
        //Reservamos memoria para n estructuras de tipo amigo
        pa = (amigo*)malloc(n*sizeof(amigo));
        …


        …
        free(pa);
}
```

Paso 3. Introducir los datos

La siguiente función pedirá la información relativa a un amigo por teclado y la almacenará en una estructura de tipo `amigo`.

```
void IntroduceDatos(amigo *pa)
{
        printf("\nNombre: ");
        gets(pa->nombre);
        printf("\nDirección: ");
        gets(pa->direccion);
        printf("\nAnyo de nacimiento: ");
        scanf("%d", &pa->anyo);
…
}
```

Se debe prestar atención a que a la función se le pasa la dirección de una variable de tipo `amigo`. De esta manera, podemos modificar los datos de una variable declarada en un entorno diferente (p.e. declarada en la función `main`).

Complete el código de la función `IntroduceDatos` para recoger el resto de información de la estructura.

Paso 4. Estructurar el código

Las siguientes funciones auxiliares nos ayudarán a estructurar el código de manera más comprensible y compacta.

Realice las funciones que se describen a continuación:

- **void MostrarDatos(amigo a)**: Función que muestra la información relativa a **una única** estructura de tipo `amigo`.
- **void ListaTodos(amigo *v, int n)**: Muestra por pantalla todos las estructuras de tipo `amigo` declaradas en el programa. Se le pasa como primer parámetro la dirección del primer elemento del array de estructuras.
- **void OrdenaPorAnyos(amigo *v, int n)**: Ordena el listado de estructuras en funcion del año de nacimiento. Emplea en este procedimiento el método de ordenación "burbuja".
- **int ComparaFechas(amigo am, int margendias)**: Esta función compara la fecha actual con la fecha del aniversario almacenada en la estructura amigo. Deberá tomar como argumento una estructura de tipo amigo y un margen de días. La función deberá devolver 1 si faltan `margendias` o menos hasta la fecha del aniversario almacenada en la estructura amigo. Devolverá 0 en caso contrario. Para hacer esto se recomienda utilizar las funciones de la librería `<time.h>`. A continuación se lista alguna de las funciones de la librería:

  time_t time (time_t * timer);
  Esta función nos devuelve el número de segundos transcurridos desde las 00:00 horas del 1 de Enero de 1970 usando el reloj del sistema. Al mismo tiempo almacena este número de segundos en la dirección apuntada por `timer`.

  struct tm * localtime (const time_t *** timer ***);
  Esta funcion convierte el tipo `time_t` a una estructura tm definida de la siguiente manera:

```
struct tm {
   int tm_hour;   /* hora (0 - 23) */
   int tm_isdst;
   int tm_mday;   /* dia del mes (1 - 31) */
   int tm_min;    /* minutos (0 - 59) */
   int tm_mon;    /* mes (0 - 11 : 0 = Enero) */
   int tm_sec;    /* segundos (0 - 59) */
   int tm_wday;   /*  dia de la semana(0 - 6 : 0 = domingo)*/
   int tm_yday;   /* dia del año (0 - 365) */
   int tm_year;   /* Año (a partir de 1900) */
};
```

  char * asctime (const struct tm * *tmptr* **);**

  Convierte una estructra `tm` a una cadena (string). La estructura de la cadena devuelta tiene el formato siguiente: `Www Mmm dd hh:mm:ss yyyy`

  time_t mktime (struct tm * ptm);

  Convierte una estructura `tm` a un tipo `time_t`. Además, comprueba los valores dentro de la estructura tm ajustando los valores si no están dentro del rango posible o son incorrectos. Después los traduce a un valor `time_t`,

indicando los segundos transcurridos desde el 1 de enero de 1970. Los valores originales de `tm_wday` y `tm_yday` son ignorados y recalculados en función del resto de parámetros.

Uso: El siguiente ejemplo muestra el uso de las funciones de la librería `<time.h>`.

```
time_t rawtime, seconds;
struct tm * timeinfo;
int diasAnyoHoy, diasAnyoFuturo, temp;

//Obtenemos el número de segundos desde 01/01/1970, se almacenan también en rawtime
seconds =  time(&rawtime);
printf ("\nHan transcurrido %ld segundos desde el 1 de Enero de 1970", seconds);
printf ("\nHan transcurrido %ld horas desde el 1 de Enero de 1970", seconds/3600);

//Obtenemos la fecha actual y la almacenamos en la estructura timeinfo
timeinfo = localtime ( &rawtime );

//Convertimos la fecha a una cadena con asctime.
printf("\nLa fecha y la hora actual es: %s", asctime(timeinfo) );

//Almacenamos el dia del año de hoy (0-365)
diasAnyoHoy = timeinfo->tm_yday;

//Modificamos los campos de la estructura
printf("\nIntroduzca una fecha futura. Dia: ");
scanf("%d", &temp);
timeinfo->tm_mday=temp;

printf("Mes: ");
scanf("%d", &temp);
timeinfo->tm_mon=temp-1;

//recalculamos el dia del año de hoy
 mktime(timeinfo);

//Lo almacenamos
diasAnyoFuturo = timeinfo->tm_yday;

printf("\nFaltan %d dias para esa fecha", diasAnyoFuturo-diasAnyoHoy);
```

Salida:

```
Han transcurrido 1158324827 segundos desde el 1 de Enero de 1970
Han transcurrido 321756 horas desde el 1 de Enero de 1970
La fecha y la hora actual es: Fri Sep 15 14:53:47 2006

Introduzca una fecha futura. Dia: 06
Mes: 11

Faltan 52 dias para esa fecha
```

Paso 5. Primera versión del programa

En este apartado intentaremos resumir el funcionamiento del programa y su estructura hasta el momento. La cabecera del programa deberá tener una apariencia similar a la siguiente

```
//*****************************************************************
//
//     NOMBRE DEL ARCHIVO:       practica4.h
//     MÓDULO:
//     FECHA:       08/09/2006
//     AUTOR:        Arturo Gil
//     DESCRIPCIÓN: Fichero de cabecera de practica4.c
```

```
//*********************************************************************
#ifndef  __PRACTICA4_H__
#define  __PRACTICA4_H__

//---------------------------------------------------------------------
// Bloque de definiciones  comunes
//---------------------------------------------------------------------
// Definición de constantes
#define TAMMAX 50
#define MARGENDIAS 15

// Definición de macros
// Definición de tipos de datos y clases
struct amigo{
       char nombre[TAMMAX];
       char direccion[TAMMAX];
       int anyo, mes, dia;
       char aficiones[TAMMAX];
};
typedef struct amigo amigo; // typedef nombreAntiguo nombreNuevo;

// Prototipos de funciones globales
void IntroduceDatos(amigo *pa);
void MuestraDatos(amigo a);
void ListaTodos(amigo *v, int n);
void OrdenaPorAnyo(amigo *v, int n);
int ComparaFechas(amigo am, int margendias);
//---------------------------------------------------------------------
#ifndef      __PRACTICA4_C__
//---------------------------------------------------------------------
// Bloque de definiciones externas
// Variables Globales externas
//---------------------------------------------------------------------
#else
//---------------------------------------------------------------------
// Bloque de definiciones internas
// Prototipos de funciones internas
void IntroduceDatos(amigo *pa);
void MuestraDatos(amigo a);
void ListaTodos(amigo *v, int n);
void OrdenaPorAnyo(amigo *v, int n);
int ComparaFechas(amigo am, int margendias);
//---------------------------------------------------------------------
#endif       // Fin bloque de definiciones internas
#endif       // Fin bloque de la cabecera
```

La función `main` tendrá una apariencia similar a la siguiente:

```
//*********************************************************************
//     NOMBRE DEL ARCHIVO:       practica4.c
//     MÓDULO:
//     FECHA:
//     AUTOR:                  Arturo Gil
//     DESCRIPCIÓN:
//*********************************************************************
#define  __PRACTICA4_C__

// Librerías Estándar
#include <stdio.h>

// Otras librerías
// Encabezado Módulo-Proyecto
// Encabezado del archivo
#include "..\inc\practica4.h"

// Variables Globales externas
// Variables Globales internas

//*********************************************************************
//     Nombre Función: main
//     Variables globales:
//     Variables de salida:
//     Comentarios (parámetros):
```

```
//*********************************************************************
void main(void)
{
       int n, result;
       int fin=0, opcion, i;
        amigo *pamigos;

      printf("Numero de estructuras que desea almacenar: ");
      scanf("%d", &n);
      pamigos = (amigo *)malloc(n*sizeof(amigo));

       while(!fin) {
              printf("1. Introducir datos\n");
              printf("2. Lista datos\n");
              printf("3. Lista fechas proximas\n");
              printf("4. Ordena por anyos la lista\n");
              printf("5. FIN\n");

              scanf("%d", &opcion);
              switch(opcion)
              {
              case 1:
                     for (i=0; i<n; i++) {
                            IntroduceDatos(&pamigos[i]);
                     }
                     break;
              case 2:
                     ListaTodos(pamigos,n);
                     break;
              case 3:
                     for (i=0; i<n; i++) {
                            result=ComparaFechas(pamigos[i], MARGENDIAS);
                            if(result)
                                   MostrarDatos(pamigos[i]);
                     }
                     break;

              case 4:
                     OrdenaPorAnyos(pamigos,n);
                     break;
              case 5:
                     fin = 1;
                     break;
              }
       }
       free(pamigos); //Liberamos memoria
}


void IntroduceDatos(amigo *pa)
{
       printf("\nNombre: ");
       gets(pa->nombre);
       printf("\nDirección: ");
       gets(pa->direccion);
       printf("\nAnyo de nacimiento: ");
       scanf("%d", &pa->anyo);
       printf("\nFecha de nacimiento: ");
       scanf("%s", pa->fecha);
       printf("\nAficiones: ");
       gets(pa->aficiones);
}

void MuestraDatos(amigo a)
{
       /*Ejercicio*/
}


void ListaTodos(amigo *v, int n)
{
       /*Ejercicio*/
}

void OrdenaPorAnyo(amigo *v, int n)
```

```
{
	/*Ejercicio*/

	/* Nota: Se recomienda utilizar el método burbuja para la ordenación.
	Nótese además que se deberá comparar v[i].anyo y v[i+1].anyo para la
	ordenación (equivalente a *(v).anyo y *(v+i).anyo )*/
}
```

El programa principal empezará por solicitar el número de estructuras de tipo amigo que para las que se desea introducir datos. A continuación muestra un menú con las siguientes opciones:

1. Introducir datos
2. Lista todos los datos
3. Lista fechas próximas
4. Ordena por años
5. FIN

Complete el código de la práctica. Compile el fichero practica4.c y compruebe su funcionamiento.

Por el momento el programa que hemos realizado no tiene demasiada utilidad ¿Por qué? Todos los datos que introducimos se pierden al finalizar el programa, ya que únicamente se guardan en memoria de programa. En el siguiente apartado le añadiremos la funcionalidad que le falta.

Paso 6. Segunda versión del programa (opcional)

Se propone en este apartado añadirle mayor funcionalidad al programa. Para ello se propone realizar dos funciones que nos permitan leer y escribir los datos en un fichero. Elegiremos el siguiente formato para escribir los datos en un fichero de texto.

```
3

José Pérez Hernández
c/Avda. de la Universidad
02 09 1980
Correr, nadar

Homer Simpson
Evergreen Terrace 742, Springfield
02 12 1950
Comer, beber

Francisco García Pérez
c/Unamuno
02 09 1945
Leer, escuchar música
```

Así pues, el primer número escrito indica el número de entradas que existen en el fichero. Se propone al alumno que dé un nombre fijo al fichero (p.e `datos.txt`), de manera que se guarden y se lean los datos siempre del mismo fichero.

Se propone al alumno que cree tres funciones:

- **int NumDatosFichero():** Esta función devuelve el número de entradas que existe en el fichero.

- **int LeerDatosFichero(amigo*):** A esta función se le pasa un array de estructuras de tipo amigo vacías. Abrirá el fichero y, a continuación, leerá los datos y los guardará en los campos correspondientes de cada elemento del array. Finalmente cerrará el fichero.

- **int EscribirDatosFichero(amigo *a, int n):** Esta función escribirá los datos de un array de estructuras de tipo amigo en un fichero. Al principio del fichero se deberá escribir el número de estructuras existentes.

Al ejecutar el programa se deberán cargar en memoria todos los datos existentes en el fichero `datos.txt`. Se deberá liberar memoria para 50 elementos más con el objeto de que se puedan introducir más datos por teclado. Al finalizar el programa se deberán guardar todos los datos en el fichero. Se propone el siguiente programa como ejemplo:

```
#define  __PRACTICA4_C__

// Librerías Estándar
#include <stdio.h>

// Otras librerías
// Encabezado Módulo-Proyecto
// Encabezado del archivo
#include "..\inc\practica4.h"

// Variables Globales externas
// Variables Globales internas

//*********************************************************************
//     Nombre Función: main
//     Variables globales:
//     Variables de salida:
//     Comentarios (parámetros):
//*********************************************************************

void main(void)
{
       int n, nmax, result;
       int fin=0, opcion, i;
      amigo *pamigos;

      n =  NumDatosFichero(); //Abre datos.txt y devuelve el número de estructuras
       nmax = n + TAMMAX;
       //Se reserva memoria para 50 estructuras más
      pamigos = (amigo *)malloc((nmax)*sizeof(amigo));

       //Cargamos los datos del fichero
      LeerDatosFichero(pamigos);

       while(!fin) {
              printf("1. Introducir datos\n");
              printf("2. Lista datos\n");
              printf("3. Lista fechas proximas\n");
              printf("4. Ordena por anyos la lista\n");
              printf("5. FIN\n");

              scanf("%d", &opcion);
              switch(opcion)
              {
              case 1:
                     //Añadimos un elemento más
                     n++;
                     if(n <= nmax){
```

```
                    printf("\nIntroduciendo datos para elemento %d", n);
                    IntroduceDatos(&pamigos[n-1]);
                    }
                    break;
             case 2:
                    ListaTodos(pamigos,n);
                    break;
             case 3:
                    for (i=0; i<n; i++) {
                           result=ComparaFechas(pamigos[i], MARGENDIAS);
                           if(result)
                                  MostrarDatos(pamigos[i]);
                    }
                    break;

             case 3:
                    OrdenaPorAnyos(pamigos,n);
                    break;
             case 6:
                    fin = 1;
                    break;
             }
      }

       EscribirDatosFichero(pamigos, n);
      free(pamigos); //Liberamos memoria
}
```

APÉNDICE: Estructuras en C

Una estructura en C es una agrupación de variables, de tipos de datos generalmente diferentes que se agrupan bajo un mismo nombre. Las estructuras nos permiten una mayor encapsulación de la información dentro de un programa, con lo que simplifica la tarea de programar. Las estructuras nos permitirán organizar información compleja dentro del programa.

Declaración: El código que se muestra a continuación define una estructura `struct amigo`.

```
#define TAMMAX 20
struct amigo{
        char nombre[TAMMAX];
        char direccion[TAMMAX];
        int anyo;
        char aficiones[TAMMAX];
};
```

Dentro de esta estructura están encapsuladas varias variables:
`char nombre[TAMMAX]`: Es un array de caracteres (cadena de caracteres). Almacena el nombre del amigo (p.e. “Pedro López Martínez”).
`char direccion[TAMMAX]`: Es un array de caracteres. Almacena la dirección del amigo (p.e. “c/Antonio Machado 45”).
`int anyo`: Es una variable de tipo entero. Almacena el año de nacimiento de la persona.
`char aficiones[TAMMAX]`: Es un array de caracteres. Almacena los gustos de la persona (p.e. Música pop, atletismo…).

En este momento, podemos declarar una estructura de tipo amigo y utilizarla dentro de un programa:

```
#define TAMMAX 20
struct amigo{
        char nombre[TAMMAX];
        char apellidos[TAMMAX];
        int anyoNacimiento;
        char fechaNacimiento[TAMMAX];
        char aficiones[TAMMAX];
};

int main(void)
{
      struct amigo amigoEjemplo;

      printf("Introduce nombre:\n");
      gets(amigoEjemplo.nombre);

      printf("Introduce los apellidos %s:", amigoEjemplo.nombre);
      gets(amigoEjemplo.apellidos);

      printf("Introduce el anyo de nacimiento:");
      scanf("%d",&amigoEjemplo.anyoNacimiento);

      printf("Introduce la fecha de nacimiento:");
      scanf("%s",amigoEjemplo.fechaNacimiento);
```

```
      printf("Introduce sus aficiones:”);
      gets(amigoEjemplo.aficiones);

      …
```

En negrita hemos indicado el momento en el que declaramos una variable de tipo `struct amigo`. Por lo tanto, el nuevo tipo de datos que hemos definido se llama `struct amigo`. Para hacer más fácil la declaración de estructuras, es una práctica habitual utilizar una sentencia typedef como la siguiente:

```
#define TAMMAX 20
struct amigo{
       char nombre[TAMMAX];
       char apellidos[TAMMAX];
       int anyoNacimiento;
       char fechaNacimiento[TAMMAX];
       char aficiones[TAMMAX];
};
typedef struct amigo amigo;

int main(void)
{
      struct amigo amigoEjemplo;

      printf("Introduce nombre:\n");
      gets(amigoEjemplo.nombre);

      printf("Introduce los apellidos %s:”, amigoEjemplo.nombre);
      gets(amigoEjemplo.apellidos);

      printf("Introduce el anyo de nacimiento:”);
      scanf(“%d”,&amigoEjemplo.anyoNacimiento);

      printf("Introduce la fecha de nacimiento:”);
      scanf(“%s”,amigoEjemplo.fechaNacimiento);

      printf("Introduce sus aficiones:”);
      gets(amigoEjemplo.aficiones);

      …
```