12. Arrays dinámicos.

Ejemplo en C

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>

typedef struct
{
  char Nombre[80];
  int  Edad;
} Persona;

typedef struct
{
  Persona* Gente;
  int NumPersonas;
} Personas;

void IniPersonas (Personas* X)
{
    X->Gente = NULL;
    X->NumPersonas = 0;
}

void FinPersonas (Personas* X)
{
    free (X->Gente);
    X->Gente = NULL;
    X->NumPersonas = 0;
}

void AgregarPersona (Personas* X, Persona* P)
{
    X->Gente = (Persona*) realloc (X->Gente,sizeof(Persona)*(X->NumPersonas+1));
    X->Gente[X->NumPersonas++] = *P;
}

void EliminarPersona (Personas* X, int Indice)
{
    int i;

    for (i=Indice ; i<X->NumPersonas-1 ; i++) X->Gente[i] = X->Gente[i+1];
    X->Gente = (Persona*) realloc (X->Gente,sizeof(Persona)*(--X->NumPersonas));
}

void ImprimirPersonas (Personas* X)
{
    int i;

    printf ("Listado de personas:\n");
    for (i=0 ; i<X->NumPersonas ; i++)
        printf ("%d - %s\n",X->Gente[i].Edad,X->Gente[i].Nombre);
    printf ("\n");
}

void main (int argc, char* argv[])
{
    Personas K;
    IniPersonas (&K);

    Persona A;
    A.Edad = 15;
    strcpy (A.Nombre,"Manolo");
    AgregarPersona (&K,&A);

    A.Edad = 25;
    strcpy (A.Nombre,"Filomeno");
    AgregarPersona (&K,&A);

    A.Edad = 48;
    strcpy (A.Nombre,"Martirio");
    AgregarPersona (&K,&A);

    ImprimirPersonas (&K);
    EliminarPersona (&K,1);
    ImprimirPersonas (&K);
    FinPersonas (&K);
}
```

El ejemplo anterior muestra el uso frecuente de un array dinámico de estructuras.

En C, el manejo de este tipo de estructuras involucra `malloc` (reservar memoria), `realloc` (reasignar el espacio asignado), `free` (liberar espacio), y copias entre elementos.

C# considera estas operaciones como “no seguras”, y ofrece tipos de datos para realizar estas tareas de forma segura. Además, dichos tipos de datos son mucho más fáciles de utilizar.

Ejemplo en C#

```
using System;
using System.Collections;

namespace Simple
{
      class CSimple
      {
        class Persona
        {
            public int Edad;
            public string Nombre;
        }

        static void ImprimirPersonas (ArrayList K)
        {
            Console.WriteLine ("Listado de personas:");
            for (int i=0 ; i<K.Count ; i++)
            {
                Persona P = (Persona) K[i];
                Console.WriteLine ("{0} - {1}",P.Edad,P.Nombre);
            }
            Console.WriteLine ();
        }

        [STAThread]
        static void Main(string[] args)
        {
            ArrayList K = new ArrayList ();

            Persona P = new Persona();
            P.Edad = 15;
            P.Nombre = "Manolo";
            K.Add (P);

            P = new Persona();
            P.Edad = 25;
            P.Nombre = "Filomeno";
            K.Add (P);

            P = new Persona();
            P.Edad = 48;
            P.Nombre = "Martirio";
            K.Add (P);

            ImprimirPersonas (K);
            K.RemoveAt (1);
            ImprimirPersonas (K);
        }
      }
}
```

Vemos que un objeto `ArrayList` tiene las operaciones necesarias para agregar y eliminar elementos, igual que el tratamiento implementado por `malloc`, `realloc` y `free`. Un objeto `ArrayList` “se inicializa solo” y “se destruye solo”. Pero esto es un tema sobre el que incidiremos profundamente cuando veamos la programación orientada al objeto, ahora no es relevante.

13. Acceso a ficheros de texto en modo lectura.

Ejemplo en C

```
#include <stdio.h>
#include <stdlib.h>

void main (int argc, char* argv[])
{
    FILE* F;
    char c;

    F = fopen ("C:\\Texto.txt","r");
    while (!feof(F))
    {
        fscanf (F,"%c",&c);
        printf ("%c",c);
    }
    fclose (F);
}
```

Ejemplo en C#

```
using System;
using System.IO;

namespace Simple
{
      class CSimple
      {
            [STAThread]
            static void Main(string[] args)
            {
            FileStream F = new FileStream ("C:\\Texto.txt",FileMode.Open,
                                                          FileAccess.Read);
            while (F.Position < F.Length)
            {
                char c = (char) F.ReadByte ();
                Console.Write (c);
            }
            F.Close ();
        }
      }
}
```

El ejemplo anterior muestra cómo leer carácter a carácter un fichero de texto. Aquí terminan las semejanzas entre C y C# en este aspecto. La función `fscanf` permite ir leyendo datos separados por espacios, convirtiendo los tipos de la forma deseada. Sin embargo, un objeto `FileStream` sólo permite leer caracteres o bloques de ellos, no es capaz de ir leyendo de separador en separador.

Ejemplo en C#

```
        static void Main(string[] args)
        {
            StreamReader F = new StreamReader ("C:\\Texto.txt");
            for (;;)
            {
                string s = F.ReadLine ();
                if (s == null) break;
                Console.WriteLine ("Linea: " + s);
            }
            F.Close ();
        }
```

El ejemplo anterior usa un objeto `StreamReader` para leer el fichero de texto, línea a línea.

Ejemplo en C#

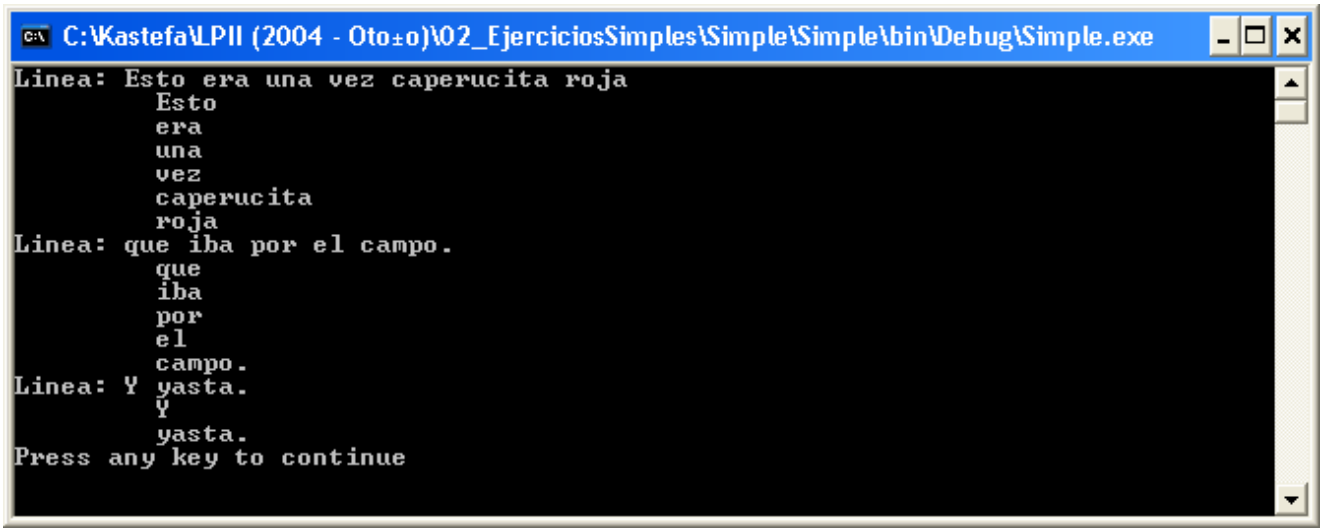
```
using System;
using System.IO;

namespace Simple
{
      class CSimple
      {
         [STAThread]
         static void Main(string[] args)
         {
            StreamReader F = new StreamReader ("C:\\Texto.txt");
            for (;;)
            {
                string s = F.ReadLine ();
                if (s == null) break;

                Console.WriteLine ("Linea: " + s);
                string[] Palabras = s.Split (' ');
                for (int i=0 ; i<Palabras.Length ; i++)
                    Console.WriteLine ("         " + Palabras[i]);
            }
            F.Close ();
        }
      }
}
```

El ejemplo anterior va leyendo del fichero línea a línea, y utiliza el método `Split` del objeto `string` para separar las palabras en un nuevo array de cadenas de caracteres. La salida de este programa muestra línea a línea el fichero, y para cada línea muestra las palabras, separadas unas de otras.

A partir de aquí, y con las operaciones del objeto `Convert` (`Convert.ToInt32`, `Convert.ToDouble`, etc), se puede emular el funcionamiento de un `fscanf` con formato.



14. Acceso a ficheros de texto en modo escritura.

Ejemplo en C

```
#include <stdio.h>
#include <stdlib.h>

void main (int argc, char* argv[])
{
    FILE* F;
    int i = 4;
    double j = 45.67;
    char k[] = "Hola";

    F = fopen ("C:\\TextoSalida.txt","wt");
    fprintf (F,"Esto es una cosa a escribir\n");
    fprintf (F,"Esto es otra cosa, que tambien salta de linea\n");
    fprintf (F,"Impresion de datos: %d %f %s\n",i,j,k);
    fclose (F);
}
```

Ejemplo en C#

```
using System;
using System.IO;

namespace Simple
{
      class CSimple
      {
        [STAThread]
        static void Main(string[] args)
        {
            StreamWriter F = new StreamWriter ("C:\\TextoSalida.txt");

            F.Write ("Esto es una cosa a escribir\r\n");
            F.WriteLine ("Esto es otra cosa, que tambien salta de linea");

            int i = 4;
            double j = 45.67;
            string k = "Hola";
            F.WriteLine ("Impresion de datos: {0} {1} {2}",i,j,k);
            F.Close ();
        }
      }
}
```

`StreamWriter` es el tipo de datos (objeto) de C# que permite imprimir en un fichero de texto de forma semejante a `fprintf` en C.