2023

Programación II

Archivos y punteros

ARCHIVOS

En la actualidad, un programa dedicado en forma exclusiva al manejo de variables, incapaz de registrar y recuperar información generada por sí mismo a lo largo de diferentes corridas, disminuye considerablemente su calidad: imposibilidad de compartir la información con otros sistemas, obligación de ejecutar nuevamente los procesos para obtener resultados alcanzados en ejecuciones previas ocasionando una importante pérdida de tiempo, etc.
Por ello, el resguardo de la información es crucial en todo software y el medio de implementarlo es a través de archivos ya sean planos o bases de datos (tema no tratado en este curso).
Los archivos pueden ser clasificados según el tipo de acceso y formato de la información almacenada.

CLASIFICACIÓN POR TIPO DE ACCESO

1. **Archivos Secuenciales:** En los archivos secuenciales, para obtener una línea específica del archivo, se deberán leer todas las líneas anteriores a ella. Esto se debe a que en este tipo de archivos no es posible acceder a un dato (línea) específico en forma directa, en consecuencia, la escritura de datos siempre se realiza al final del archivo**.**

2. **Archivos Directos:** Los archivos de acceso directo (aleatorio), permiten almacenar en ellos registros que poseen una estructura establecida manteniendo el tipo de dato de cada uno de sus campos. Similar al concepto de arreglos de estructuras, tema tratado en la materia Laboratorio I. La gran diferencia reside, como se puede apreciar, en que la información almacenada en un archivo no se pierde al finalizar la ejecución del programa, como si ocurre al manejar arreglos en memoria.

La característica de reconocer los tipos de datos almacenados es posible gracias a que la información almacenada en este tipo de archivos se encuentra en formato binario.
La principal cualidad de los archivos directos es que permiten acceder a un registro determinado sin necesidad de leer todos los anteriores.
Se debe destacar que:

- Toda operación de lectura o escritura desplazara el puntero del archivo a la próxima posición.
- Será posible ubicar al puntero en una posición determinada gracias a una función de desplazamiento.
- Para el borrado de un registro determinado se pude recurrir a una de las siguientes alternativas: marcar el registro como eliminado (dando un determinado valor al mismo dentro de un campo reservado para tal fin) y de esta manera omitirlo durante la generación de los listados y consultas. Otro camino es copiando todos los registros del archivo, excepto el registro en cuestión, para luego eliminar el archivo original y renombrar el nuevo archivo generado.

CLASIFICACIÓN POR FORMATO DE ALMACENAMIENTO

1. **Archivos de Texto:** En memoria el almacenamiento de cada caracter requiere 2 bytes (16 bits), el mismo espacio requerido para almacenar un entero corto (short). En los archivos de texto cada caracter es almacenado individualmente por lo que pueden ser leídos, interpretados, por cualquier procesador de texto. De esta forma, si en un archivo se desea almacenar la palabra “peras” ésta ocupará 10 bytes (cada caracter ocupa 2 bytes), lo mismo ocurre al tratar de almacenar el numero entero 15789 el cual también ocupará 10 bytes ya que se lo almacena como un conjunto de caracteres y no como el valor que realmente representa.

Manipulación de un Archivo de Texto Secuencial

```
//incluir el namespace System.IO
static void Main(string[] args)
{
FileStream Archivo;
StreamWriter StreamGrabar;
StreamReader StreamLeer;
string Cadena;
char Opcion;

Console.WriteLine("Ingrese Opcion (1-Crear 2-Agregar 3-Listar 4-Salir):");
Opcion = Console.ReadKey().KeyChar;
```

```
switch (Opcion)
{    case '1':
     {
        Console.WriteLine("\nIngrese la primer linea a almacenar en el nuevo archivo.");
        Cadena = Console.ReadLine();
        Archivo = new FileStream("Lista.txt",FileMode.Create);
        StreamGrabar = new StreamWriter(Archivo);
        StreamGrabar.WriteLine(Cadena);
        StreamGrabar.Close();
        Archivo.Close();
        break;
     }
   case '2':
     {
        Console.WriteLine("\nIngrese una nueva linea a agregar en el archivo.");
        Cadena = Console.ReadLine();
        Archivo = new FileStream("Lista.txt",FileMode.Append);
        StreamGrabar = new StreamWriter(Archivo);
        StreamGrabar.WriteLine(Cadena);
        StreamGrabar.Close();
        Archivo.Close();
        break;
     }
   case '3':
     {
        if (File.Exists("Lista.txt"))
        {
           Console.WriteLine("\nContenido del Archivo:");
           Archivo = new FileStream("Lista.txt", FileMode.Open);
           StreamLeer = new StreamReader(Archivo);
           while (StreamLeer.EndOfStream == false)
           {
              Cadena = StreamLeer.ReadLine();
              Console.WriteLine(Cadena);
           }
           StreamLeer.Close();
           Archivo.Close();
        }
        else
        {
           Console.WriteLine("\nEl archivo no existe.");
        }
        break;
     }
}
Console.ReadKey();
}
```

2. **Archivos Binarios:** En ciertas ocasiones puede ser útil o imprescindible almacenar directamente el contenido de la memoria en un archivo. A este tipo de archivos se los denomina binarios, en ellos, al almacenar la palabra “peras” se ocuparán 10 bytes (2 bytes por caracter) pero al almacenar el número entero corto (short) 15789 solo se ocuparán 2 bytes. El “inconveniente” con estos archivos es que no son legibles directamente desde un procesador de texto.

Por esta razón el tipo de archivo a utilizar depende de la información a almacenar y su finalidad. Si se desea un archivo de uso general, que pueda ser leído por cualquier usuario por medio de un procesador de texto se recurrirá a un archivo de texto, en cambio, si se desea almacenar una imagen, una estructura con diferentes tipos de datos, etc. se optara por un archivo binario.

Manipulación de un Archivo Binario Secuencial

```
//incluir el namespace System.IO static
void Main(string[] args)
{
   FileStream Archivo;
   BinaryWriter BinarioGrabar;
   BinaryReader BinarioLeer;
   string Cadena;
   int NDoc;
   string Apellido;
   string Nombre;
   float Sueldo;
   char Opcion;

   Console.WriteLine("Ingrese Opcion (1-Crear 2-Agregar 3-Listar 4-Salir):");
   Opcion = Console.ReadKey().KeyChar;

   switch (Opcion)
   {
     case '1':
        {
           Archivo = new FileStream("Lista.txt", FileMode.Create);
           BinarioGrabar = new BinaryWriter(Archivo);
           Console.WriteLine("\nIngrese el Nro Doc.:");
           Cadena = Console.ReadLine();
           NDoc = int.Parse(Cadena);
           BinarioGrabar.Write(NDoc);
           Console.WriteLine("\nIngrese el Apellido:");
           Apellido = Console.ReadLine();
           BinarioGrabar.Write(Apellido);
           Console.WriteLine("\nIngrese el Nombre:");
           Nombre = Console.ReadLine();
           BinarioGrabar.Write(Nombre);
           Console.WriteLine("\nIngrese el Sueldo:");
           Cadena = Console.ReadLine();
           Sueldo = float.Parse(Cadena);
           BinarioGrabar.Write(Sueldo);

           BinarioGrabar.Close();
           Archivo.Close();
           break;
        }
     case '2':
        {
           Archivo = new FileStream("Lista.txt", FileMode.Append);
           BinarioGrabar = new BinaryWriter(Archivo);

           Console.WriteLine("\nIngrese el Nro Doc.:");
           Cadena = Console.ReadLine();
```

```
            NDoc = int.Parse(Cadena);
            BinarioGrabar.Write(NDoc);
            Console.WriteLine("\nIngrese el Apellido:");
            Apellido = Console.ReadLine();
            BinarioGrabar.Write(Apellido);
            Console.WriteLine("\nIngrese el Nombre:");
            Nombre = Console.ReadLine();
            BinarioGrabar.Write(Nombre);
            Console.WriteLine("\nIngrese el Sueldo:");
            Cadena = Console.ReadLine();
            Sueldo = float.Parse(Cadena);
            BinarioGrabar.Write(Sueldo);

            BinarioGrabar.Close();
            Archivo.Close();
            break;
        }
```

```
      case '3':
        {
          if (File.Exists("Lista.txt"))
          {
            Console.WriteLine("\nContenido del Archivo:");
            Archivo = new FileStream("Lista.txt", FileMode.Open);
            BinarioLeer = new BinaryReader(Archivo);

            while (BinarioLeer.PeekChar() != -1)
            {
              NDoc = BinarioLeer.ReadInt32();
              Apellido = BinarioLeer.ReadString();
              Nombre = BinarioLeer.ReadString();
              Sueldo = BinarioLeer.ReadSingle();
              Console.WriteLine("{0}-{1}-{2}-{3}", NDoc, Apellido, Nombre, Sueldo);
            }
            BinarioLeer.Close();
            Archivo.Close();
          }
        else
          {
            Console.WriteLine("\nEl archivo no existe.");
          }

          break;
        }
    }
    Console.ReadKey();
}
```

PUNTEROS

A través de una tabla de direcciones, el nombre de cada variable se asocia a una dirección de memoria, en la cual se encuentra el valor contenido por la variable. Este valor deberá corresponderse con el tipo de dato especificado en la declaración de la variable. De esta forma no es necesario recordar la dirección en la que se encuentra un valor determinado, es suficiente con invocar el nombre que tiene asociado esa dirección.
Un puntero es una variable, que como todas ellas cuenta con su propia dirección en memoria, pero se diferencia de las mismas en que su contenido es una dirección a otra variable.
Al momento de la declaración se deberá especificar el tipo de dato de la variable a la cual apuntará precedido por el signo "*":

TipoDeDato *NombreDeVariablePuntero

Es posible declarar punteros a punteros, estos de declaran con 2 signos "*":

TipoDeDato **NombreDeVariablePuntero

Para la manipulación de los punteros existen 2 operadores:

- **&:** permite conocer la dirección de una variable.
- ***:** permite devolver/asignar el contenido de la variable a la que apunta.

Ejemplo

```
unsafe static void Main(string[] args)
{
   float SueldoEmpleado, SueldoJefe;
   float *Importe;
   SueldoEmpleado = 1000;
   SueldoJefe = 1800;
   Importe = &SueldoEmpleado; //Importe contiene la direccion de SueldoEmpleado
   *Importe = 1150; //el empleado recibio un aumento de $150, la variable SueldoEmpleado = 1150
   Importe = &SueldoJefe; //Importe contiene la direccion de SueldoJefe
   *Importe = *Importe + 200; //el jefe recibio un aumento de $200, la variable SueldoJefe = 2000
}
```

No es posible:

- Utilizar el operador "&" con valores constantes ni expresiones.
- Modificar la dirección de una variable.
- Asignación entre punteros de diferentes tipos de datos, para realizarlo se debe efectuar previamente una conversión de tipos:

Ejemplo

```
unsafe static void Main(string[] args)
{
   longSueldoEmpleado;
   long *ImporteA;
   long *ImporteB;
   int *ImporteC;
   int **ImporteD;
   SueldoEmpleado = 1150;
   ImporteA = &SueldoEmpleado;
   ImporteB = ImporteA; //punteros del mismo tipo de dato, ImporteB apunta a la misma direccion que apunta
ImporteA
   ImporteC = (int *) ImporteA; //punteros de diferente tipo de dato, ImporteC apunta a la misma direccion que
apunta ImporteA
   ImporteC = (int *) &SueldoEmpleado;//puntero y variable de diferente tipo de dato, ImporteC apunta a la direccion
de SueldoEmpleado
   ImporteD = &ImporteC; //ImporteD apunta a la direccion que apunta ImporteC
   **ImporteD = 1350; //SueldoEmpleado=1350
}
```

Las operaciones que se pueden efectuar en los punteros son suma y resta, esto permite desplazarse hacia posiciones de memoria que se encuentren por delante y por detrás de la dirección a la que apunta. La cantidad de bytes que se desplazará por cada unidad dependerá del tipo de dato con el que fue declarado. Esta característica permite utilizar los punteros como elementos para recorrer arreglos, entre otras alternativas.
El nombre de un arreglo es un puntero a la 1er posición del mismo.

Ejemplo

```
unsafe static void Main(string[] args)
{
float[] Sueldos;
Sueldos = new float[4];
Sueldos[0] = 1000;
Sueldos[1] = 350;
Sueldos[2] = 750;
Sueldos[3] = 950;

fixed (float* PunteroSueldo = &Sueldos[0]) //tambien podria expresarse como PunteroSueldo=Sueldos
{
   Console.WriteLine(*PunteroSueldo); // presentara por pantalla el contenido de Sueldos[0]
   Console.WriteLine(*(PunteroSueldo + 1)); // presentara por pantalla el contenido de Sueldos[1]
   Console.WriteLine(*(PunteroSueldo + 2)); // presentara por pantalla el contenido de Sueldos[2]
   Console.WriteLine(*(PunteroSueldo + 3)); // presentara por pantalla el contenido de Sueldos[3]
}

fixed (float* PunteroSueldo = &Sueldos[2])
{
   Console.WriteLine(*PunteroSueldo); // presentara por pantalla el contenido de Sueldos[2]
   Console.WriteLine(*(PunteroSueldo + 1)); // presentara por pantalla el contenido de Sueldos[3]
}

fixed (float* PunteroSueldo = &Sueldos[1], PunteroSueldo2 = &Sueldos[3])
{
   Console.WriteLine(PunteroSueldo[-1]); //presenta por pantalla el contenido de Sueldos[0]
   Console.WriteLine(PunteroSueldo[0]); //presenta por pantalla el contenido de Sueldos[1]
   Console.WriteLine(PunteroSueldo[2]); //presenta por pantalla el contenido de Sueldos[3]
   Console.WriteLine(PunteroSueldo2 - PunteroSueldo); //presenta por pantalla la cant. de elementos entre ambos
(31=2)
}

Console.ReadKey();
}
```