Estructuras de Datos

Manejo de Archivos

Prof. Luis Garreta

Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana – Cali

10 de noviembre de 2017

Memoria Interna o RAM

- Hasta el momento se presentaron diferentes estructuras de datos, cada una de ellas clasificadas de acuerdo a distintos criterios.
- Sin embargo, todas las estructuras tienen una particularidad en común, son internas y temporales, es decir, son definidas en un algoritmo y ocupan memoria RAM.
- Estas se utilizan en ejecución y todos los valores contenidos en ellas se pierden, a lo sumo, cuando el algoritmo termina de ejecutarse.
- La memoria interna o RAM de una computadora es denominada memoria principal y sus principales características son:
  - a) Tiene capacidad limitada
  - b) Es memoria volátil ya que al apagar la computadora o al cortarse la luz, se pierde la información que en ella reside.
  - c) Es memoria electrónica, por lo cual su velocidad de acceso es muy elevada (memoria de acceso aleatorio)

Memoria Auxiliar o Secundaria

- Para determinados problemas es necesario disponer de un tipo de estructura de datos que permita guardar su información en soporte no volátil o temporal (digamos en disco, cinta, etc.), y de esa forma preservarla aunque el programa finalice.
- La memoria secundaria, auxiliar o externa se refiere a medios de almacenamiento que están ubicados fuera de la memoria RAM, y que son capaces de retener información luego de que un programa finaliza o luego de apagar la computadora, como por ejemplo: disco rígido, diskette, cinta, cd, etc.
- Dichos dispositivos secundarios almacenan información en medios magnéticos (disco rígido, cintas, diskettes, etc.) o medios ópticos (cd, zip, dvd, etc.), esto hace mucho más lenta su operación.
- La velocidad de acceso a la memoria afecta notablemente la eficiencia de un algoritmo.
- El tiempo de acceso a memoria RAM está en el orden de nanosegundos (10-9 seg), en tanto que el acceso a disco rígido está medido en milisegundos (10-3 seg).

Estructuras de Datos en Memoria Secundaria: Archivos

- Las estructuras de datos que se asocian con un dispositivo de memoria auxiliar permanente donde almacenan la información se denominan archivos.
- Usaremos archivos:
  - Cuando la cantidad de datos que manipula el programa muy grande y no hay suficiente espacio en la RAM
  - Cuando el resultado o la salida de un programa sirva de entrada para otro programa, por lo cual es necesario guardarlo en forma permanente
  - Cuando la carga sucesiva de datos se torna engorrosa, pesada y lleva demasiado tiempo, por lo cual conviene guardarla en forma permanente.

Definición de Archivos

Un archivo es una colección de datos del mismo tipo relacionados entre sí, almacenados en una unidad de memoria secundaria, con un determinado diseño o estructura.

- Para dar diseño a los archivos se usan los **registros**: físicamente un archivo se almacena como una sucesión de datos estructurados por el diseño de un registro.
- La información se guarda con el formato especificado por el registro, y se recupera con ese mismo formato.
- De esta manera, un registro será la mínima unidad de información transferible entre un archivo y un programa.
- Así podemos decir que "**un conjunto de registros con ciertos aspectos en común, y organizados para algún propósito particular, constituyen un archivo**".

Tipos de Archivos

- Al final, todos los archivos son secuencias de bytes.
- Sin embargo, se pueden distinguir dos tipos de archivos de acuerdo a su contenio: Texto y Binarios.
- **Archivos Tipo Texto:**
  - Se caracterizan porque guardan líneas de caracteres o secuencias de caracteres terminados en salto de línea ("\n")
  - El tipo de caracteres se representa el código legible ya sea código ASCII o más reciente en Unicode.
  - Ejemplos: archivos fuentes .c, .cpp, .h, .java, .py, .txt, etc.
- **Archivos Tipo Binario:**
  - Guardan la información en bytes y el si

Archivos Tipo Texto

- Son aquellos que pueden contener cualquier clase de datos y de tal manera que son "entendibles" por la gente.
- Los datos en un archivo de texto se almacenan usando el código ASCII en el cual cada carácter es representado por un simple byte.
- Debido a que los archivos de texto utilizan el código ASCII, se pueden desplegar o imprimir.
- En este tipo de archivos, todos sus datos se almacenan como cadenas de caracteres,
- Es decir, los números se almacenan con su representación ASCII y no su representación numérica.
- Por lo tanto no se pueden realizar operaciones matemáticas directamente con ellos.
- Por ejemplo: Si se guarda el dato 3.141592 en un archivo de texto, se almacena como "3.141592" y nótese que $3.141592 \neq$ "3.141592"
- Ejemplos de archivos tipo texto: archivos fuentes .c, .cpp, .h, .java, .py, .txt, etc.

Archivos Tipo Binario

- Este tipo de archivos almacenan los datos numéricos con su representación binaria.
- Pueden ser archivos que contienen instrucciones en lenguaje máquina listas para ser ejecutadas.
- Por ejemplo, un programa en C++ tiene las instrucciones almacenadas en un archivo de texto llamado programa fuente, pero una vez que se compila se traslada a un archivo ejecutable tipo binario (en lenguaje máquina), que es directamente entendido por la computadora.
- En este tipo de archivos también se pueden almacenar diferentes tipos de datos incluyendo datos numéricos que se graban con su representación binaria (no con su representación ASCII).
- Por esto, cuando se despliegan con un editor de textos aparecen caracteres raros que no se interpretan.
- Por ejemplo, si se guarda el dato 27 en un archivo binario, se almacena como 00001111 y no como “27”.

Organización de los archivos

La organización de un archivo define la manera en que los registros se distribuyen en el almacenamiento secundario:

- 1- Secuencial:
- 2- Indexado o directo:
- 3- Secuencial indexado:

Archivos Secuenciales

Un archivo secuencial consiste de un conjunto de registros almacenados consecutivamente, de manera que para acceder al n-ésimo registro, se debe acceder previamente a los (n – 1) anteriores. Los registros se graban en forma consecutiva en el orden que se ingresan, y se recuperan en el mismo ordenamiento

Archivos Indexados o Directos

Un archivo indexado está formado por un conjunto de registros donde el ordenamiento físico no se corresponde con el ordenamiento lógico; esto es, se recuperan por su posición dentro del archivo (o por clave) sin necesidad de recorrer los anteriores.

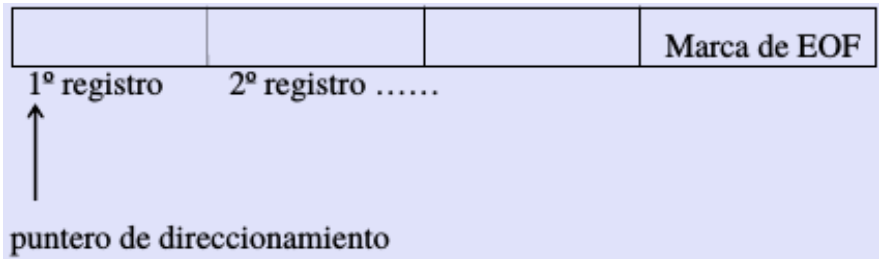
Archivos Secuenciales indexados

Un archivo particionado utiliza estructuras de datos auxiliares para permitir un acceso pseudo-directo y combina las dos técnicas anteriores. Se agrupan registros, donde cada grupo se accede directamente (por posición o clave), y dentro de cada grupo los registros se recuperan secuencialmente (ej. La guía telefónica: voy directamente a la letra de comienzo del apellido, y a partir de allí busco uno a uno hasta encontrar el apellido).

Operaciones básicas sobre archivos

Las operaciones elementales para trabajar con archivos son:

- **Open**: Abrir un archivo para procesarlo
- **Close**: Cerrar el archivo luego de terminar de operar con él
- **Read**: Leer los datos almacenados en el mismo o recuperarlos
- **Write**: Escribir los nuevos datos en un archivo o modificar los ya existentes
- **Seek**: Ubicar el puntero de direcionamient del archivo en una posición determinada.



Archivos en C++

Para C++, todos los objetos de archivos perteneces a una de las siguientes clases:

Class	**Description**
ifstream	Objects belong to this class are associated with files opened for input purposes
ofstream	Objects belong to this class are associated with files opened for output
fstream	Objects belong to this class are associated with files opened for input and output purposes.

Open: Abrir un archivo

Open

void open(const char *name, int mode, int access)

Si open no lleva ninguna bandera, por defecto se abre el archivo como **tipo texto**
De lo contrario debe colocar en el modo de apertura la bandera **ios::binary**

Donde:

- name: cadena con el nombre del archivo
- mode: bandera con el modo de apertura, algunos son:
  - **ios::in** Open for input, with open() member function of the ifstream variable.
  - **ios::out** Open for output, with open() member function of the ofstream variable.
  - **ios::binary** Binary file, if not present, the file is opened as an ASCII file as default.
  - **ios::app** Append data to the end of the output file.
- access: Constante dependiente del sistema operativo (0, 1, 2, o 4)

Metodos para Lectura/Escritura en Archivos Texto

- Para leer caracter por caracter: **objetoArchivo.get (char ch);**
- Para leer linea por línea: **objectoArchivo.getline (char [] buffer, int sizeBuffer);**
- Para leer con formato: **objeto archivo >> variableEntrada;**
- Para escribir con formato: **objeto archivo << variableSalida;**

Ejemplo01: Lectura y Escritura archivos Texto

```
#include <fstream>
#include <stdlib.h>
#include <iostream>
using namespace std;

// Lectura-Escritura simple archivos.txt
// Lectura caracterxcaracter

int main(void) {
    ifstream inputfile;
    ofstream outputfile;
    char chs;

    inputfile.open ("archivoparaleer.txt"); // !!debe estar creado
    outputfile.open("archivoparaescribir.txt");

    while (!inputfile.eof()) {
        inputfile.get(chs); // Lectura de caracter
        cout<<chs;
    outputfile << chs;  // Escritura del caracter
        }
    }
    inputfile.close();
    outputfile.close();
}
```

Ejemplo02: Lectura con formato archivo texto

```
#include <iostream>
#include <fstream>
#include <stdlib.h>

// Lectura de cada linea el entero y la cadena en dos variables

using namespace std;

main(void) {
    ifstream inputfile;
    int indice;
    string palabra;

    inputfile.open ("archivo-indices.txt");
    /* 0 car
     * 1 green
     * 2 blue
     * 3 red
     * 4 cat
     */

  while (!inputfile.eof()) {
    inputfile >> indice >> palabra;  // lectura de un entero y un string
    cout << "Indice es: " << indice;
    cout << "  Palabra es: " << palabra;
    cout << endl;
  }
  inputfile.close();
}
```

Métodos para manejo de archivos binarios y acceso directo

Función	**Sintaxis**	**Descripción**
ubicar para lectura	seekg(desplazamiento, origen)	Mueve el puntero del archivo a la posición donde va a leer
ubicar para escritura	seekp(desplazamiento, origen)	Mueve el puntero del archivo a la posición donde va a escribir.
escribir	write((char *) &Obj, sizeof(Obj));	Escribe al archivo el buffer apuntado por **Obj** la cantidad de bytes del objetos
leer	read((char *) &Obj, sizeof(Obj));	Lee del archivo la cantidad y la deja en el objeto **Obj**

Para la ubicación del puntero del archivo:

- Origen en una constante entera que dice a partir de donde se realizará el desplazamiento (número de bytes) del puntero

Ejemplo03: Escritura de Registros en Archivos Binarios

```
#include <fstream>
#include <string.h>
using namespace std;

// Crea tres registros y escribe a un archivo binario

typedef struct Estudiante {
    char codigo [10];
    char nombre [50];
};

int main () {
  Estudiante e1, e2, e3;
  strcpy (e1.codigo, "abcd"); strcpy (e1.nombre, "Julio Moncada");

  strcpy (e2.codigo, "1030"); strcpy (e2.nombre, "Tulio Ramirez");

  strcpy (e3.codigo, "1040"); strcpy (e3.nombre, "Maria Belenna");

  ofstream salida;
  salida.open ("estudiantes.bin", ios::binary);

  salida.write ((char *)&e1, sizeof (e1));
  salida.write ((char *)&e2, sizeof (e2));
  salida.write ((char *)&e3, sizeof (e3));

  salida.close ();
}
```

Lectura de Acceso Directo en un Archivo Binario

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
using namespace std;

// Lee del archivo binario "estudiantes.bin" los
// el segundo y tercer registro

typedef struct Estudiante {
    char codigo [10];
    char nombre [50];
};

int main () {
  Estudiante e2, e3;
  ifstream in;
  in.open ("estudiantes.bin", ios::in|ios::binary);

  in.seekg (2*60,ios::beg);
  in.read ((char *)&e2, sizeof (e2));
  cout << e2.codigo <<", "<<e2.nombre << endl;

  in.seekg (1*60,ios::beg);
  in.read ((char *)&e3, sizeof (e2));
  cout << e3.codigo <<", "<<e3.nombre << endl;


  in.close ();
}
```

Metodos de Verificación de Errores y Final de Archivo

Function	**Explanation:**
bad()	Returns True if an Error occurs while reading or writing Operation. (Example: in case we try to write to a file that is not open for writing or if the device where we try to write has no space left).
fail()	Returns true in the same cases as bad() plus in case that a format error happens. (as trying to read an integer number and an alphabetical character is received:).
eof()	Returns true if a file opened for reading has reached the end.

Ejemplo05: Verificación de Errores de Apertura y Final de Archivo

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
using namespace std;

// Verificacion de erorres y final de archivo
main(void) {
    ifstream inputfile;
    ofstream outputfile;
    char chs;
    inputfile.open ("archivoparaleer.txt");
    outputfile.open("archivoparaescribir.txt");
    if (inputfile.fail () == true || outputfile.fail()==true) { // Verificacion
        cout << "Error abriendo archivos " << endl;
        exit (1);
    }

    while (!inputfile.eof()) {  // Mientras no llegue al final
        nputfile.get(chs);
        if (!inputfile.eof()) { // Mientras no esté en el final
            cout<<chs;
            outputfile<<chs;
        }
    }
    inputfile.close();
    outputfile.close();
}
```