ESTRUCTURA DE DATOS I
LABORATORIO 01

LUIS EDUARDO LLERENA TRUJILLO
ELKIN RICARDO SANTIS PALENCIA
JOHAN ANDRES VERGARA VEGA

Lic:
SEBASTIAN RACEDO

UNIVERSIDAD DEL NORTE
BARRANQUILLA, ATLÁNTICO
2
0
2
2

Punto 1.

Inicio

Buffer
Clientes
Dispositivo Disco
Reg_Clientes
Cedula N[10]
Nombre X[20]
Dir X[20]
Celular N[20]
Email X[30]
FinBuffer

Buffer
Facturas
Dispositivo Disco
Reg_Facturas
CedulaF N[10]
Numfactura N[10]
CodProdutor N[10]
Cantidad N[5]
FinBuffer

Buffer
Productos
Dispositivo Disco
Reg_Productos
Codigo N[10]
Descripcion X[100]
Precio N[10]
FinBuffer

Buffer
Deudas
Dispositivo Disco
Reg_Deudas

```
        CedulaD N[10]
        NombreD X[20]
        CelularD N[12]
        Totaldeudas N[12]
FinBuffer

Buffer
Codigo
Dispositivo Disco
Reg_Codigo
        NroCod   N[3]
        contador N[1]
FinBuffer

Abrir Clientes(lectura)
Abrir Facturas(lectura)
Abrir Produtos(lectura)
Abrir Deudas(Escritura)
Abrir Codigo(Escritura)

registrado <- falso
Leer Reg_Clientes
Leer Reg_Facturas
totaldeudas <- 0
MQ (Not(EOF(Clientes)) And Not(EOF(Facturas))) hacer
        encontrado <- falso
        Si (Cedula = CedulaF) entonces
                Si (Not(registrado)) entonces
                        CedulaD <- Cedula
                        NombreD <- Nombre
                        CelularD <- Celular
                        registrado <- verdadero
        Finsi
        Cerrar Productos
        Abrir Productos(lectura)
        Leer Reg_Productos
        MQ (Not(EOF(Productos) And Not(encontrado)) hacer
```

```
            Si(CodProdutor = Codigo) entonces
                encontrado <- verdadero
                totaldeudas <- totaldeudas + cantidad*Precio
                Leer Reg_facturas
            SiNo
            Leer Reg_Productos
            FinSi
        FinMQ
        SiNo
            Si (CedulaF > Cedula) entonces
                Escribir Reg_Deudas
                Leer Reg_Clientes
                totaldeudas <- 0
                registrado <- falso
            SiNo
                 Leer Reg_facturas
            Finsi
    Finsi
FinMQ

Cerrar Facturas
Cerrar Productos
Abrir Productos(Lectura)
Abrir Factura (Lectura)
Leer Reg_Productos

MQ (Not(EOF(Productos))) hacer
        Cerrar Facturas
        Abrir Facturas (lectura)
        NroCod <- Codigo
        MQ (EOF(Facturas)) hacer
            Si(Codigo = CodProductor) entonces
                con <- con + 1
                Leer Reg_Facturas
            SiNo
                Leer Reg_Facturas
            FinSi
```

```
        FinMQ
        contador <- con
         Escribir Reg_Codigo
         Leer Reg_Productos
    FinMQ
    Cerrar Codigo
    Abrir Codigo (Lectura)
    Leer Reg_Codigo
    mayor <- 0
    MQ (Not(Eof(Codigo))) hacer
        Si (contador > mayor) entonces
                mayor <- contador
         FinSi
        Leer Reg_Codigo
    FinMQ
    Cerrar Codigo
    Abrir Codigo (Lectura)
    Leer Reg_Codigo
    MQ (Not(Eof(Codigo))) hacer
        Si (mayor = Contador) entonces
                Escibrir "El Código del producto con mayor número de clientes es: ", NroCod
        SiNo
                Leer Reg_Codigo
        Finsi
    FinMQ

Fin
```

Punto 2.

Manipulación de archivos en Python

Abrir archivos:

Para abrir un archivo en python usamos la función open(). Esta devuelve el contenido del archivo como un objeto de python.

Syntax: open(file_name, mode)
file_name: es el nombre y la ruta del archivo
mode: este parámetro es un string que se utiliza para especificar el modo en el que debe abrirse el archivo.

> 'r': esta cadena se utiliza para leer (sólo) el archivo. Se pasa por defecto si no se suministra ningún parámetro y devuelve un error si no existe tal archivo.
>
> 'w': esta cadena se utiliza para escribir en/sobre el archivo. Si el archivo con el nombre suministrado no existe, crea uno por ti.
>
> 'x': esta cadena se utiliza para crear un archivo específico.
>
> 'a': abrir para escribir, añadiendo al final del archivo si existe
>
> 'b': Esta cadena se utiliza cuando el usuario quiere manejar el archivo en modo binario. Generalmente se utiliza para manejar archivos de imagen.
>
> 't': text mode (default) : Esta cadena se utiliza para manejar archivos en modo texto. Por defecto, la función open() utiliza el modo texto.

Leer archivos

Con open() tendremos ya en fichero el contenido del documento listo para usar, y podemos imprimir su contenido con read(). El siguiente código imprime todo el fichero.

Syntax: print(file_name.read())

```
#Contenido de la primera línea
#Contenido de la segunda línea
```

El método readlines() devuelve una lista que contiene cada línea del archivo como un elemento de la lista.

Utilice el parámetro x para limitar el número de líneas devueltas. Si el número total de bytes devueltos supera el número especificado, no se devuelven más líneas.

Syntax: print(file_name.readline(x))

Escribir en archivo

Ya hemos visto cómo crear y leer un archivo . Veamos ahora cómo podemos añadir contenido. El método de archivo de Python write() escribe una cadena str en el archivo. No hay valor de retorno.

Syntax*:* file_name.write(str)
str - Es la cadena que se escribirá en el archivo.

Cerrar archivo

Es muy importante el uso de .close() ya que si dejamos el fichero abierto, podríamos llegar a tener un comportamiento inesperado que queremos evitar. Por lo tanto, siempre que se abre un archivo es necesario cerrarlo cuando hayamos acabado.

Syntax: file_name.close()

Para agregar contenido al final del archivo en Python utilizamos el mismo método filename.readline() pero ahora con la cadena 'a+' como parámetro el cual agrega el contenido al final del archivo y si la ruta no existe, creará un archivo nuevo.

```
#Abrimos el archivo
archivo = open("data.txt", "r")

#Leemos el archivo y mostramos los registros en pantalla
data = archivo.readlines()
print (data)

#Escribimos sobre el archivo
for linea in data:
    archivo.write(linea)

#No olvidemos cerrar el archivo
archivo.close()
```

Manipulación de archivos en JavaScript

El entorno NodeJS se utiliza para diferentes scripts que incluyen el manejo de archivos. NodeJS no es más que un entorno para ejecutar código JavaScript.

Abrir archivos:

Para abrir un archivo en JavaScript, usamos el método fs.open el cual tomará dos argumentos, la ruta y el modo. El argumento “ruta” se utiliza para localizar el archivo.

Syntax: fs.open(file_name, mode)

file_name: Es el nombre y la ruta del archivo.
mode: se utiliza para abrir el archivo de diferentes modos como adjuntar, escribir y leyendo.

'r'	Abre un archivo en modo lectura.
'un'	Abre un archivo en modo adjunto.
'w'	Abre un archivo en modo de escritura.
'a +'	Abre un archivo en modo de adición y lectura.
'w +'	Abre un archivo en modo de escritura y lectura.
'r +'	Abre un archivo en modo lectura y escritura.

Leer archivos:

Para leer archivos en JavaScript usamos el método fs.readFile() el cual leerá el contenido del archivo y arrojará un error si el archivo no existe en la ruta dada.

Syntax: fs.readFile(file_name, callback)
file_name: Es el nombre y la ruta del archivo.
callback: Función donde detectaremos si hay un error en la ejecución del método.

Escribir en archivos:

Para escribir en archivos usamos el método fs.writeFile() el cual escribirá el contenido en el archivo. Si el archivo no existe en la ruta dada, se creará uno nuevo.

Syntax: fs.writeFile(file_name, content, callback)

file_name: Es el nombre y la ruta del archivo.
content: Es el contenido a escribir en el archivo.
callback: Función donde detectaremos si hay un error en la ejecución del método.

Para agregar contenido al final del archivo en JavaScript utilizamos el método fs.appendFile() el cual agrega el contenido al final del archivo y si la ruta no existe, creará un archivo nuevo.

Syntax: fs.writeappendFile(file_name, content, callback)

file_name: Es el nombre y la ruta del archivo
content: Es el contenido a escribir en el archivo
callback: Función donde detectaremos si hay un error en la ejecución del método

En JavaScript tenemos métodos misceláneos que nos permitirán eliminar y renombrar un archivo, los cuales son fs.unlink() y fs.rename() respectivamente.

Cerrar archivo

En JavaScript no es necesario cerrar los archivos ya que cada uno de los métodos que usamos para el manejo de archivos realiza este proceso al finalizar el callback.

Syntax: none

```
const fs = require("fs");

let data = fs.readFileSync("data.txt", "utf-8").split("\r\n");
for (let i = 0; i < data.length; i++) {
  if (data[i].includes("Melania")) {
    data.splice(i, 1);
  }
}
let text = "";
for (let i = 0; i < data.length; i++) {
  text += data[i] + "\r\n";
}

fs.writeFile("data.txt", text, () => {
  console.log("Done");
});
```

Manipulación de archivos en C++

El archivo de cabecera **fstream.h** define las clases **ifstream**, **oftream** y **fstream** para operaciones de lectura, escritura y lectura/escritura en archivos respectivamente. Para trabajar con archivos debemos crear objetos de éstas clases de acuerdo a las operaciones que deseamos efectuar.

Abrir archivos.

Declaramos un objeto de la clase ofstream, después utilizamos la función miembro open para abrir el archivo.

Syntax:

```
offstream file
file.open("file_name.txt", ios::in);
```

- ios::app Operaciones de añadidura.
- ios::ate Coloca el apuntador del archivo al final del mismo.
- ios::in Operaciones de lectura. Esta es la opción por defecto para objetos de la clase ifstream.
- ios::out Operaciones de escritura. Esta es la opción por defecto para objetos de la clase ofstream.
- ios::nocreate Si el archivo no existe se suspende la operación.
- ios::noreplace Crea un archivo, si existe uno con el mismo nombre la operación se suspende.
- ios::trunc Crea un archivo, si existe uno con el mismo nombre lo borra.
- ios::binary Operaciones binarias.

Leer archivo

Para abrir un archivo y realizar operaciones de lectura se crea un objeto de la clase ifstream y se procede prácticamente de la misma forma que lo expuesto en el apartado anterior. Después de abrir el archivo se puede leer su contenido utilizando las funciones miembro de la clase(***getline( )***)ifstream. Cuando se lee un archivo, por lo general se empieza al principio del mismo y se leerá su contenido hasta que se encuentre el final del archivo. Para determinar si se ha llegado al final del archivo se puede utilizar la función ***EOF*** como condición de un

bucle ***While***. Además se puede utilizar la función miembro fail para detectar un error al abrir el archivo.

Escribir en archivo

Para abrir un archivo y realizar operaciones de escritura se crea un objeto de la clase ofstream, se inicializa con el método .open() y finalmente mediante operadores de asignación escribimos el contenido al archivo

Syntax:

ofstream archivo;
archivo.open("file.txt", ios::out);
archivo << "contenido" << endl;

Cerrar archivo

Es muy importante el uso de .close() ya que si dejamos el fichero abierto, podríamos llegar a tener un comportamiento inesperado que queremos evitar. Por lo tanto, siempre que se abre un archivo es necesario cerrarlo cuando hayamos acabado.

Syntax: file_name.close()

```
11  int main()
12  {
13  ifstream archivo;
14      ofstream archivod("temp.txt", ios::out);
15      string texto, texto2;
16
17      archivo.open("../data.txt", ios::in);
18
19      if (archivo.fail())
20      {
21          cout << "no se pudo abrir el archivo";
22          exit(1);
23      }
24      string nombre = "Jane Chadwick";
25      while (!archivo.eof())
26      {
27          getline(archivo, texto);
28          if (!(texto.find(nombre) != string::npos))
29          {
30
31              archivod << texto << endl;
32          }
33      }
34
35      archivo.close();
36      remove("../data.txt");
37      rename("temp.txt", "../data.txt");
38
39      return 0;
40  }
```

¿Todos los lenguajes de programación usan los mismos métodos para trabajar con archivos (tanto para lectura como escritura)?
Si, en todos los lenguajes de programación se usan los mismos métodos para trabajar con archivos ya que aunque tengan estructura diferentes y varían en cuanto sus sintaxis, todos tienen un método para abrir un archivo, uno para leer el archivo y uno para leer el archivo.

Para ejemplificar esto hemos tomado como muestras los lenguajes JavaScript, Python y C ++. A continuación podemos ver estos métodos en cada lenguaje.

ABRIR UN ARCHIVO

JavaScript:

```javascript
const fs = require("fs");

fs.open("sample.txt", "w", (err, file) => {
  if (err) throw err;
  console.log(file);
});
```

C++

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

int main()
{
    ofstream file;

    file.open("file.txt", ios::in);

    return 0;
}
```

Python

```python
##contenido = open("nombre-del-archivo.txt", "modo")

## Agregarás contenido y lo podrás leer
contenido = open("nombre-del-archivo.txt", "a+")

## Solo podrás leer el contenido
contenido = open("nombre-del-archivo.txt", "r")
```

LEER UN ARCHIVO

JavaScript

```javascript
const fs = require("fs");

fs.readFile("sample.txt", (err, data) => {
  if (err) throw err;
  console.log(data.toString());
});
```

C++

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

int main()
{
    ifstream archivo;
    string texto;
    archivo.open("datos.txt", ios::in);
    if (archivo.fail())
    {
        cout << "no se pudo abrir el archivo";
        exit(1);
    }
    while (!archivo.eof())
    {
        getline(archivo, texto);
        cout << texto << endl;
    }
    archivo.close();
    return 0;
}
```

Python

```python
## Abrimos en modo solo lectura
contenido = open("nombres.txt", "r")

## Recorremos y mostramos cada línea enumerada
contador = 1
print(contenido.read())
```

ESCRIBIR UN ARCHIVO

JavaScript

```javascript
const fs = require("fs");

fs.writeFile("sample.txt", "Writing content", (err) => {
  if (err) throw err;
  console.log("Completed! xd");
});
```

Python

```python
## La lista de nombres a escribir
nombres = ["Juan", "Susana", "Andrea", "Melanie", "Andrés"]

## Abrimos en modo escritura
archivo = open("nombres.txt", "w")

## Recorremos la lista y agregamos cada nombre al archivo
for nombre in nombres:
    archivo.write(nombre + "\n")

## Muy importante cerrar el archivo.
archivo.close()
```

C++

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

using namespace std;


int main()
{
    ofstream archivo;

    archivo.open("file.txt", ios::out);

    if (archivo.fail())
    {
        cout << "no se pudo abrir el archivo";
        exit(1);
    }

    archivo << "contenido" << endl;

    archivo.close();
    return 0;
}
```

Experimentos

Tiempo de ejecución

Lectura de archivos

Tiempo de ejecución - Leer archivos			
	C++	JavaScript	Python
T1	0,844	0,707	0,142
T2	1,036	0,271	0,119
T3	1,035	0,249	0,112
T4	0,922	0,279	0,119
T5	0,963	0,268	0,112
T6	1,047	0,268	0,122
T7	1,1	0,236	0,115
T8	1,119	0,248	0,112
T9	1,057	0,252	0,111
T10	1,086	0,242	0,097
Promedio	1,0209	0,302	0,1161

Como podemos observar, el programa que mejor se desempeñó fue el codificado en Python, el cual menos tiempo en promedio demoró en realizar la tarea. Por el contrario, el más demorado fue C++.

Búsqueda en archivos

Tiempo de ejecución - Buscar en archivos			
	C++	JavaScript	Python
T1	1,225	0,137	0,184
T2	0,961	0,114	0,11
T3	0,935	0,121	0,11
T4	0,947	0,116	0,12
T5	1,264	0,122	0,111
T6	1,002	0,128	0,109
T7	0,929	0,12	0,115
T8	1,028	0,114	0,112
T9	0,95	0,145	0,111
T10	0,942	0,107	0,108
Promedio	1,0183	0,1224	0,119

Como podemos observar, el programa que mejor se desempeñó fue el codificado en Python, el cual menos tiempo en promedio demoró en realizar la tarea. Por el contrario, el más demorado fue C++.

Eliminar un registro

Tiempo de ejecución - Eliminar un registro			
	C++	JavaScript	Python
T1	2,075	0,129	0,189
T2	2,3	0,151	0,157
T3	2,012	0,134	0,325
T4	2,185	1,967	0,325
T5	2,845	0,276	0,153
T6	2,116	0,333	0,153
T7	2,472	0,266	0,21
T8	2,408	0,339	0,139
T9	1,899	0,668	0,144
T10	1,973	0,936	0,371
Promedio	2,2285	0,5199	0,2166

Como podemos observar, el programa que mejor se desempeñó fue el codificado en Python, el cual menos tiempo en promedio demoró en realizar la tarea. Por el contrario, el más demorado fue C++.

Complejidad

Para comparar la complejidad del script en cada lenguaje, debemos tener en cuenta múltiples variables, una de ellas es la cantidad de líneas empleadas en cada uno.

Líneas empleadas

Lineas empleadas			
	C++	Javascript	Python
Abrir un archivo	9	5	1
Leer un archivo	22	6	1
Escribir en un archivo	18	5	2

Como se puede ver, python se corona nuevamente como ganador al emplear menos líneas de código para abrir, leer, y escribir un archivo. Y una vez más C + + ocupa el tercer lugar en esta prueba.

Importes de paquetes

Otro factor a tener en cuenta para comparar la complejidad de nuestro script en cada lenguaje es la cantidad de paquetes o librerías que es necesario importar para el funcionamiento de nuestro código. esto lo podemos ver al notar como en python no es necesario importar ninguna librería, mientras que tanto en JavaScript como en C++ es necesario importar las librerías de File System(fs para JavaScript y fstream para C++).

Dificultad para borrar un registro

Uno de los procesos que puede ser más determinante para nuestra comparación es el de eliminar un registro dado. mientras que en JavaScript y Python tenemos la facilidad de borrar el elemento de un array(JavaScript) o una lista(Python) y posteriormente reescribir el archivo en C++ nos vemos en la necesidad de crear un nuevo archivo para guardar todos los registros excepto el que queremos eliminar para posteriormente eliminar nuestro archivo inicial y renombrar con el nombre de nuestro archivo inicial el archivo que acabamos de crear.

En qué se diferencian (si es que se diferencian) los algoritmos que trabajan sobre archivos en la actualidad de los vistos en clase? Explicar su respuesta.

Se diferencian solo sintácticamente ya que a pesar que los métodos tienen nombres y estructuras diferentes realizan la misma función.

Ejemplo:

Abrir archivo.
Si bien para trabajar con un archivo el primer paso es asignarlo, esta operación no es suficiente para operar con dicho archivo. Se debe realizar la apertura del mismo indicando el tipo de operaciones que se van a realizar sobre él: solo lectura (entrada) o escritura (salida). Los lenguajes de programación poseen instrucciones específicas para tal fin.
En pseudocódigo la usaremos así:
Abrir *file_name* (tipo de operación)
Pero en general todos los lenguajes funcionan de igual manera hacen lo mismo.