UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE INGENIERÍA Y CIENCIAS
CC3001 - 1 Algoritmos y Estructuras de Datos

TAREA 2
Expanción

Daniela Campos
danielacamposfischer@gmail.com

Profesor:
Patricio Poblete

Fecha:
24 de Abril del 2017

Índice

1 Introducción **2**

2 Análisis del Problema **3**
2.1 Problema . . . 3
2.2 Suposiciones sobre el Problema . . . 3
2.3 Casos Bordes . . . 3

3 Solución del Problema **4**
3.1 Clases y Métodos Utilizados . . . 4
3.1.1 Clase Expand . . . 4
3.1.2 Método main . . . 4
3.1.3 Método check . . . 4
3.1.4 Método lector . . . 4
3.1.5 Clase ListaEnlazada . . . 5
3.1.6 Método ListaEnlazada . . . 5
3.1.7 Método concatenate . . . 5
3.1.8 Método length . . . 5
3.1.9 Método get . . . 5
3.1.10 Método contains . . . 5
3.2 Ejemplos Entradas y Salidas . . . 6
3.2.1 Entrada 1: . . . 6
3.2.2 Salida 1: . . . 6
3.2.3 Entrada 2: . . . 7
3.2.4 Salida 2: . . . 7
3.2.5 Entrada 3: . . . 7
3.2.6 Salida 3: . . . 7

4 Modo de Uso **8**

5 Resultados y Análisis **9**

6 Anexos **10**

1 Introducción

En computación, al escribir un programa, se puede hacer referencia a otros archivos, para esto, se requiere un compilador. Al compilar el programa, el compilador deberá expandir las referencias a otros archivos, reemplazandolas con el código correcto.

En el presente informe se explicarán los pasos que se siguieron para crear un programa que funcione como compilador. El programa leerá un archivo y lo irá imprimiendo en la pantalla, utilizando la salida estandar y realizando los reemplazos correspondientes cuando exista una referencia a otro archivo.

2 Análisis del Problema

2.1 Problema

$$\frac{2x^30}{y}$$

El problema propuesto, consiste en crear un programa que funcione como un compilador, es decir, que vaya leyendo un archivo, imprimiendolo en la pantalla y reemplazando las referencias a otros archivos con lo correspondiente al caso.

Las referencias a otros archivos con las que se trabajará seran denotadas de la siguiente forma <<<**nombrearchivo**>>>. Ejemplos de los archivos con los que se puede trabajar se encuentran en los Anexos 1,2,3,4.

El programa debería ser capaz de recibir un listado de archivos y compilarlos todos, evitando así que se tengan que operar de a uno los archivos. Al listado de nombres de archivos se les deberá aplicar el proceso de expanción de manera recursiva.

2.2 Suposiciones sobre el Problema

El programa debería ser capaz de tomar un listado grande de nombres de archivos y además estos deben ser tomados como argumentos del método **main** y no a traves de la entrada estandar.

2.3 Casos Bordes

El programa podría tratar de compilar archivos que contengan referencias circulares, como los siguiente archivo:

```
El archivo 2 dice :
<<<archivo2>>>, y el archivo 6
dice: <<<archivo6>>>
aqui termina el archivo 1.
```

Figura 1: archivo5.txt

```
El archivo  6 contiene al
archivo 5 aqui : <<<archivo5>>>
```

Figura 2: archivo6.txt

En este caso, el Archivo 5 hace referencia al Archivo 5 y este a su vez, hace referencia al Archivo 5, produciendo así un ciclo infinito de llamadas recursivas y provocando un **StackOverFlowException**.

3 Solución del Problema

3.1 Clases y Métodos Utilizados

3.1.1 Clase Expand

Esta clase contiene el código que compila y expande los archivos entregados. Contiene los métodos **main**, **lector** y **check**.

3.1.2 Método main

Este método se encuentra en la **Clase Expand**. Toma como argumento un listado de archivos y para cada archivo en el listado, crea una Lista Enlazada, la cual ayuda al método check a verificar si existe una referencia circular, si es que esta existe, se imprime un mensaje de advertencia en la pantalla, de otra manera, se imprime el contenido del archivo con los reemplazos correspondientes a referencias a otros archivos.[1]

3.1.3 Método check

Este método se encuentra en la **Clase Expand**. Toma como argumento un archivo en forma de String y una Lista Enlazada que contiene los archivos que ya han sido leidos y el archivo actual. Utiliza métodos de la librería IO para leer los archivos.

Atravieza cada linea del archivo y si encuentra una referencia a otro, chequea si se encuentra en la Lista Enlazada, si es el caso retorna un -1, si ese no es el caso, aplica recursivamente el método check al archivo al cual se está haciendo referencia. Si al haber atravezado todo el archivo y sus respectivas referencias no ha encontrado una referencia circular, retorna un 0.[2]

3.1.4 Método lector

Este método se encuentra en la Clase Expand. Toma como argumento un archivo en forma de String. Utiliza los métodos de la librería IO para leer los archivos.

Atravieza cada linea del archivo imprimiendo lo que se encuentra en esta. Si encuentra una referencia a otro archivo, la corta del archivo original y luego llama de manera recursiva al método lector para el archivo al cual se hizo referencia.[3]

[1] El código del método **main** puede ser encontrado en el Anexo 5.
[2] El código del método check puede ser encontrado en el Anexo 6
[3] El código del método lector puede ser encontrado en el Anexo 7

3.1.5 Clase ListaEnlazada

Esta clase contiene el código que permite crear una lista enlazada. Contiene un constructor y los métodos **concatenate**, **length**, **get** y **contains**.[4]

3.1.6 Método ListaEnlazada

Este método se encuentra en la **Clase ListaEnlazada**. Es el constructor de la lista y toma como argumento a un String.

3.1.7 Método concatenate

Este método se encuentra en la **Clase ListaEnlazada**. Toma como argumento una Lista Enlazada y la concantena con la lista a la cual se le está aplicando el método.

3.1.8 Método length

Este método se encuentra en la **Clase ListaEnlazada**. No toma argumentos y retorna el largo de la lista enlazada al cual es aplicado.

3.1.9 Método get

Este método se encuentra en la **Clase ListaEnlazada**. Toma como argumento un Integer i y retorna el valor que se encuentra en el índice i de la Lista Enlazada.

3.1.10 Método contains

Este método se encuentra en la **Clase ListaEnlazada**. Toma como argumento un String y chequea si este se encuentra en la Lista Enlazada, de ser el caso, retorna su índice, si no, retorna un -1.

[4]El código para la Clase ListaEnlazada se encuentra en el Anexo 8

3.2 Ejemplos Entradas y Salidas

A continuación se mostraran resultados obtenidos al ejecutar el programa con diferentes entradas:

3.2.1 Entrada 1:

```
La Clase Expand contiene
a los metodos: <<<archivob>>>
y utiliza a la
Clase ListaEnlazada para chequear
si existe una referencia circular
de la forma: <<<archivoc>>>
```

Figura 3: archivoa.txt

```
main, check, lector
```

Figura 4: archivob.txt

```
archivo1 llama a archivo2
y archivo 2 llama a archivo1.
```

Figura 5: archivoc.txt

3.2.2 Salida 1:

```
> java Expand archivoa.txt
La Clase Expand contiene
a los metodos: main, check, lector
y utiliza a la
Clase ListaEnlazada para chequear
si existe una referencia circular
de la forma: archivo1 llama a archivo2
y archivo 2 llama a archivo1.
```

Figura 6: Compilación archivoa.txt

3.2.3 Entrada 2:

```
main, check, lector
```

Figura 7: archivob.txt

```
archivo1 llama a archivo2
y archivo 2 llama a archivo1.
```

Figura 8: archivoc.txt

3.2.4 Salida 2:

```
> java Expand archivob.txt archivoc.txt
main, check, lector
archivo1 llama a archivo2
y archivo 2 llama a archivo1.
```

Figura 9: Compilación archivob.txt y archivoc.txt

3.2.5 Entrada 3:

```
El archivo e dice:
<<<archivoe>>>
```

Figura 10: archivod.txt

```
El archivo d dice:
<<<archivod>>>
```

Figura 11: archivoe.txt

3.2.6 Salida 3:

```
> java Expand archivod.txt
Se ha encontrado una referencia circular en archivod.txt
```

Figura 12: Compilación archivod.txt

4 Modo de Uso

Se deben guardar en una misma carpeta las clases Expand y ListaEnlazada, junto con los archivos que quieren ser compilados. Para ejecutar el programa desde el terminal, se debe hacer de la siguiente manera:

```
> java Expand archivo.txt
```

Si se desean compilar varios archivos, se debe proceder como sigue:

```
> java Expand archivo1.txt archivo2.txt archivo3.txt
```

5 Resultados y Análisis

Se puede apreciar que el programa funciona bien, ya que logra compilar un listado de archivos, independiente de su tamaño. Además logra detectar cuando existe una referencia circular independiente de si es directa ($archivo1 \Rightarrow archivo2 \Rightarrow archivo1$) o indirecta ($archivo1 \Rightarrow archivo2 \Rightarrow archivo3 \Rightarrow archivo1$).

Como era solicitado, el programa utiliza recursión para leer e imprimir los contenidos de un archivo utilizando el método **lector**. Pero además, la ultiliza para chequear referencias circulares en el método **check**.

6 Anexos

1. **Anexo 1 :** archivo1.txt

```
El archivo 2 dice :
<<<archivo2>>>, y el archivo 3
dice: <<<archivo3>>>
aqui termina el archivo 1.
```

2. **Anexo 2 :** archivo2.txt

```
[ contenido del archivo 2]
```

3. **Anexo 3 :** archivo3.txt

```
El archivo 3 contiene al
archivo 4 aqui: <<<archivo4>>>
```

4. **Anexo 4 :** archivo4.txt

```
[ contenido del archivo 4]
```

5. **Anexo 5 : Método main**

```
public static void main(String[] args){
        for (String s: args) {
                ListaEnlazada archivos = new ListaEnlazada(s);
                if (check(s, archivos) == 0){
                lector(s);
                }
                else {
                        System.out.println("Referencia circular ...");
                }
        }
}
```

6. **Anexo 6 : Método check**

```
public static int check(String archivo, ListaEnlazada archivos){
        String line;
        int b = 0;
        try{
            FileReader readableFile = new FileReader(archivo);
            BufferedReader reader = new BufferedReader(readableFile);
            line = reader.readLine();
            while(line!=null){
                int a = 0;
                boolean hayArchivo = false;
                for(int i = 0; i < line.length(); i++){
                    if(a > 0){
                        a--;
                        continue;
                    }
                    if(line.charAt(i) == '<'){
                        a = 2;
                        hayArchivo = true;
                        int j;
                        for(j = 0; j < line.length(); j++){
                            if(line.charAt(j) == '>'){
                                break;
                            }
                        }
                        String cortar = "<<<" + line.substring
                                (i + 3, j) + ">>>";
                        String nuevoArchivo = line.
                                substring(i + 3, j);
                        char endline = line.charAt(line.length() - 1);
                        if(archivos.contains(nuevoArchivo
                                + ".txt") != -1){
                            b = -1;
                        }
                        else{
                            ListaEnlazada l = new ListaEnlazada
                                (nuevoArchivo + ".txt");
                            archivos.concatenate(l);
                            line = line.replace(cortar, "");
                            b = check(nuevoArchivo
                                + ".txt", archivos);
                        }
                    }
```

```
                }
                line = reader.readLine();
            }
            reader.close();
            readableFile.close();
        }
        catch(IOException e){
            e.printStackTrace();
        }
        return b;
    }
```

7. **Anexo 7 : Método lector**

```
public static void lector(String archivo){
        String line;
        try{
            FileReader readableFile = new FileReader(archivo);
            BufferedReader reader = new BufferedReader(readableFile);
            line = reader.readLine();
            while(line != null){
                int a = 0;
                boolean hayArchivo = false;
                for(int i = 0; i < line.length(); i++){
                    if(a > 0){
                        a--;
                        continue;
                    }
                    if(line.charAt(i) == '<'){
                        a = 2;
                        hayArchivo = true;
                        int j;
                        for(j = 0; j < line.length(); j++){
                            if(line.charAt(j) == '>'){
                                break;
                            }
                        }
                        String cortar = "<<<" + line.substring
                                (i + 3, j) + ">>>";
                        String nuevoArchivo = line.
                                substring(i + 3, j);
                        char endline = line.charAt(line.length() - 1);
                        line = line.replace(cortar, "");
                        if(endline == '>'){
                            System.out.print(line);
                            lector(nuevoArchivo + ".txt");
                        }
                        else {
                            lector(nuevoArchivo + ".txt");
                            System.out.println(line);
                        }
                    }
                }
                if(hayArchivo == false){
                    System.out.println(line);
                }
```

```
                line = reader.readLine();
            }
            reader.close();
            readableFile.close();
            if(line != null){
                System.out.println(line);
            }
        }
        catch(IOException e){
            e.printStackTrace();
        }
    }
```

8. **Anexo 8 : Clase ListaEnlazada**

```
public class ListaEnlazada {

  private String valor;
  private ListaEnlazada siguiente;

  public ListaEnlazada(String s) {
    this.valor = s;
    this.siguiente = null;
  }

  public void concatenate(ListaEnlazada l) {
    ListaEnlazada actual = this;

    while(actual.siguiente != null) {
      actual = actual.siguiente;
    }

    actual.siguiente = l;
  }

  public int length(){
    int i;

    ListaEnlazada actual = this.siguiente;
    for(i = 1; actual != null; i++) {
      actual = actual.siguiente;
    }

    return i;
  }

  public ListaEnlazada get(int i) {

    if(this.length() < i) {
      return null;
    }

    ListaEnlazada ans = this;
    for(int j = 0; j < i; j++) {
      ans = ans.siguiente;
    }
```

```
    return ans;

  }

  public int contains(String s) {
    int length = this.length();

    for(int i = 0; i < length; i++) {
      ListaEnlazada actual = this.get(i);
      if(actual.valor.equals(s)) {
        return i;
      }
    }

    return -1;
  }


}
```