Guion de prácticas

Entrega de prácticas:
Compilación separada con NetBeans

Febrero 2021

Metodología de la Programación

DGIM-GII-GADE

Curso 2020/2021

Índice

Contents

1 Práctica a entregar: Compilación separada con NetBeans (Palisimétrica) 5

2 Objetivos 6

3 Entrega 8

4 Apéndice 1: El programa original 9

1 Práctica a entregar: Compilación separada con NetBeans (Palisimétrica)

El programa a utilizar en esta entrega utiliza dos tipos de datos representados por las clases **SecuenciaCaracteres** y **VectorSecuencias** que van a representar una palabra vista como una secuenica de caracteres y un conjunto de palabras como un vector de palabras respectivamente.

La clase **SecuenciaCaracteres** es una estructura con tres campos y un método privado. El primer campo representa el tamaño máximo para la secuencia de caracteres (**TAMANIO**), el segundo es un vector de caracteres que respresentará a la secuencia de caracteres (**v**) y el tercero un contador de las posiciones usadas para representar la secuencia (**utilizados**). Por último, veáse el código del cuadro siguiente donde figura la definición de la clase además de un método privado (**contarOcurrencias**) que devuelve el número de ocurrencias en la secuencia del carácter pasado como parámetro:

```
class SecuenciaCaracteres{
private:
    static const int TAMANIO = 5000;
    char v[TAMANIO];
    int utilizados;
    int contarOcurrencias(char caracter){
        int contador=0;
        for(int i=0; i < utilizados; i++){
            if(v[i] == caracter){
                contador++;
            }
        }
        return contador;
    }
```

Además la clase **SecuenciaCaracteres** contiene una serie de métodos públicos disponibles:

```
SecuenciaCaracteres();
int obtenerUtilizados();
int capacidad();
void aniade(char nuevo);
char elemento(int indice);
int primeraOcurrencia(char aBuscar);
SecuenciaCaracteres subsecuencia(int posIni, int posFin);
bool esPalisimetrica();
```

en el que destacaremos el método **esPalisimetrica**[1] que devuelve un dato booleano que indica cuándo la secuencia de caracteres es palisimética, es decir, si cuando se parte por la mitad, da como resultado dos secuencias que tienen las mismas letras con las mismas frecuencias. Si la longitud de la secuencia es un número impar, la decisión se toma sin considerar la letra central. Por ejemplo la secuencia de caracteres `gaga` es palisimétrica. Las dos mitades `ga` y `ga` tienen los mismos caracteres con la misma frecuencia. Otros ejemplos de secuencias palisimétricas son `rotor`, `aabccaba`, `xyzxy`. La secuencia `abbaab` no es palisimétrica.

La segunda clase **VectorSecuencias** es una estructura con tres campos. El primer campo representa el tamaño máximo para el vector (**TAMA**),

[1] Este método es parte de un examen de Fundamentos de la Programación del curso 2020/2021.

el segundo es un vector de secuencias que respresentará al grupo de palabras (**vector**) y el tercero un contador de las posiciones usadas para representar la secuencia (**utilizados**).

```
class VectorSecuencias{
private:
    static const int  TAMA=20;
    SecuenciaCaracteres vector[TAMA];
    int utilizados;
```

Esta clase cuenta también con varios métodos públicos:

```
VectorSecuencias();
void aniade(SecuenciaCaracteres s);
int capacidad();
SecuenciaCaracteres secuencia(int indice);
int obtenerUtilizados();
```

Además en el fichero `secuencia.cpp` se encuentran varias funciones externas a las clases que son utilizadas en el programa principal main:

```
void pintaSecuencia(SecuenciaCaracteres s);
void pintaVector(VectorSecuencias v);
int cuantasPalisimetricas(VectorSecuencias v);
```

2 Objetivos

Para realizar esta tarea, tenga en cuenta que el objetivo es dividir el programa mediante la separación de ficheros de la forma vista en el guión de prácticas sobre Netbeans (Sección 4 - Un proyecto de compilación separada). Para ello, descargue el fichero **Secuencia.zip** (con la estructura de ficheros de la Figura 1) y descomprímalo en una carpeta independiente.

```
./
├── data
│   └── palisimetrica.dat
├── doc
│   └── secuencia.doxy
├── scripts
│   ├── runDocumentation.sh
│   └── doZipProject.sh
└── src
    └── secuencia.cpp
```

Figura 1: Contenido del fichero **Secuencia.zip**

- Compruebe mediante la creación de un proyecto de NetBeans que el programa dado funciona correctamente, redireccionando la entrada para usar el fichero **palisimetrica.dat** proporcionado en la carpeta **data**, y comprobando que la salida es la siguiente:

```
Secuencias Iniciales
gaga
rotor
aabcaba
abbaab

Hay 3 y 1 No Palisimetricas

Secs Palisimetricas
gaga
rotor
aabcaba

Secs No Palisimetricas
abbaab
```

- Separe el programa original en múltiples ficheros, de forma que cada uno de ellos soporte una parte de la implementación total. De esta forma, se separará la implementación de las clases y funciones en cada fichero **.cpp** mientras que las declaraciones de las clases y declaraciones de las funciones se incluirán en unos ficheros **.h** llamados ficheros de cabeceras. Además se creará un módulo **main.cpp** que contendrá el código que implementa el programa de comprobación de si una secuencia de caracteres es palisimétrica o no.

  No olvide incluir las directivas del preprocesador usando

```
#ifndef _FICHERO_H_
#define _FICHERO_H_
...
#endif
```

  en los ficheros `.h`, así como los `#include` necesarios en los ficheros `.h` y/o `.cpp` que lo necesiten. Para ello puede ser de utilidad dibujar un grafo de dependencias entre los distintos ficheros.

- Cree el nuevo proyecto de NetBeans con la modularización creada, configúrelo para que utilice la redirección de entrada con los datos proporcionados y obtenga el ejecutable binario.

- Una vez obtenido el ejecutable, compruebe el funcionamiento de los métodos con los datos proporcionados en **data**; estos son algunas secuencias de prueba, no obstante ¡¡pruebe también con otros ejemplos!!

- Si lo desea, puede ejecutar el script **runDocumentation.sh** para la obtención de la documentación del proyecto creado.

- Por útimo, ejecute el script **doZipProject.sh** para la obtención del paquete (entrega1.zip) a entregar en PRADO.

3 Entrega

Se debe de entregar a través de **PRADO** un fichero zip, **entrega1.zip** con la estructura de directorios ya expuesta en el guión sobre NetBeans: **data, doc, include, scripts, src, zip**.

El fichero **entrega1.zip** debe contener la estructura de la Figura 2.

```
./
├── build
├── data
│   └── palisimetrica.dat
├── dist
├── doc
│   └── secuencia.doxy
├── include
│   ├── secuencia.h
│   └── vectorSecuencias.h
├── nbproject
├── scripts
│   ├── runDocumentation.sh
│   └── doZipProject.sh
├── src
│   ├── secuencia.cpp
│   ├── vectorSecuencias.cpp
│   └── main.cpp
├── zip
└── Makefile
```

Figura 2: Estructura de la entrega entrega2.zip

Deben mantenerse estos nombres de carpetas para que el código pueda corregirse y compilarse de forma automática tras la entrega. También deben ajustarse los nombres de los archivos, funciones, clases,...., a las indicaciones dadas en los ejercicios. **NOTA: no se considerarán ejercicios que no cumplan estas normas. Tampoco aquellos en que se usen espacios en blanco en los nombres de los archivos o carpetas o caracteres especiales (como acentos)**.

En general, se proporciona el código básico para probar la funcionalidad pedida, con los casos de prueba indicados. Debe observarse la forma de uso de las funciones y/o métodos en el programa principal para especificar de forma correcta los argumentos de las funciones. **NOTA: puede modificarse el contenido del programa principal, pero compruebe que funciona con el main original**. Debe comprobarse que el programa entregado en **Prado** se ejecute y funcione correctamente.

4 Apéndice 1: El programa original

Todos estos ficheros están disponibles en Prado en el fichero **Secuencia.zip** para su descarga en la página de la asignatura.

```
#include <cstring>
#include <cstdlib>
#include <iostream>

using namespace std;

class SecuenciaCaracteres{
private:
    /**
     * Constante para el TAMANIO maximo de las secuencias
     */
    static const int TAMANIO = 5000;

    /**
     * secuencia de caracteres
     */
    char v[TAMANIO];

    /**
     * Contador de posiciones usadas
     */
    int utilizados;

    /**
     * Devuelve el contador de ocurrencias de un determinado
     * caracter
     * @param caracter caracter objetivo
     * @return contador de ocurrencias
     */
    int contarOcurrencias(char caracter){
        int contador=0;
        for(int i=0; i < utilizados; i++){
            if(v[i] == caracter){
                contador++;
            }
        }

        // devuelve contador
        return contador;
    }

public:

    /**
     * Constructor por defecto
     * Metodo ya implementado
     * @return
     */
    SecuenciaCaracteres(){
        // basta con hacer que utilizados sea 0
        utilizados=0;
    }

    /**
     * Devuelve el numero de posiciones usadas
     * Metodo ya implementado
     * @return
     */
    int obtenerUtilizados(){
        return utilizados;
    }

    /**
     * Devuelve el espacio total en la secuencia
     * Metodo ya implementado
     * @return
     */
    int capacidad(){
        return TAMANIO;
    }

    /**
     * Agrega un nuevo caracter a la secuencia
     * Metodo ya implementado
     * @param nuevo
     */
    void aniade(char nuevo){
        // se agrega si hay espacio
        if(utilizados < TAMANIO){
            v[utilizados]=nuevo;
            utilizados++;
        }
    }

    /**
     * Se accede al caracter de una posicion.
     * Metodo ya implementado
     * NOTA: no se comprueba la validez del indice
     * @param indice indice objetivo
     * @return caracter almacenado en indice
     */
```

```
    char elemento(int indice){
        return v[indice];
    }

    /**
     * Localiza la primera ocurrencia del caracter
     * pasado como argumento
     * Metodo ya implementado
     * @param aBuscar caracter a buscar
     * @return posicion de primera ocurrencia del caracter
     */
    int primeraOcurrencia(char aBuscar){
        int posicion=-1;
        for(int i=0; i < utilizados; i++){
            if(v[i] == aBuscar){
                posicion=i;
            }
        }

        // se devuelve la posicion
        return posicion;
    }

    /**
     * Construye la subsecuencia contenida entre dos posiciones
     * @param posIni
     * @param posFin
     * @return
     */
    SecuenciaCaracteres subsecuencia(int posIni, int posFin){
        SecuenciaCaracteres resultado;

        // solo se trabaja si las posiciones son validas
        if(posFin >= posIni && posIni >= 0 && posIni < utilizados
                && posFin < utilizados){
            // se agregan los caracteres de interes
            for(int i=posIni; i <= posFin; i++){
                resultado.aniade(v[i]);
            }
        }

        // se devuelve el resultado
        return resultado;
    }

    /**
     * Determina si una palabra cumple la condicion de palisimetrica
     * @return resultado de la comprobacion
     */
    bool esPalisimetrica(){
        bool resultado=true;

        // se generan dos secuencias: una para cada mitad
        int mitad = utilizados/2;

        // la primera secuencia va desde 0 hasta mitad
        SecuenciaCaracteres primera;
        for(int i=0; i < mitad; i++){
            primera.aniade(v[i]);
        }

        // se determina si la cadena tiene numero para o
        // impar de caracteres para determinar el comienzo
        // de la segunda cadena
        if(utilizados % 2 != 0){
            mitad = mitad+1;
        }

        SecuenciaCaracteres segunda;
        for(int i=mitad; i < utilizados; i++){
            segunda.aniade(v[i]);
        }

        // se genera una secuencia con los caracteres que
        // aparecen en la secuencia, sin repetidos
        SecuenciaCaracteres sinRepetidos;
        for(int i=0; i < utilizados; i++){
            if(sinRepetidos.primeraOcurrencia(v[i]) == -1){
                sinRepetidos.aniade(v[i]);
            }
        }

        // Se recorren ahora los caracteres de la secuencia
        // sin repetidos y se cuentan las apariciones en cada
        // una de las secuencias (para ello se usa un metodo
        // auxiliar, que consideramos privado)
        for(int i=0; i < sinRepetidos.utilizados && resultado == true; i++){
            int ocurrenciasPrimera = primera.contarOcurrencias(sinRepetidos.v[i]);
            int ocurrenciasSegunda = segunda.contarOcurrencias(sinRepetidos.v[i]);

            // se comprueba la igualdad
            if(ocurrenciasPrimera != ocurrenciasSegunda){
                resultado=false;
            }
        }

        // se deveulve el valor de resultado
        return resultado;
    }
};
```

```
//////////////////////////////////////////////////////////////////////////////////////////

class VectorSecuencias{
private:
     /**
      * Constante para el TAMANIO maximo de las secuencias
      */
     static const int  TAMA=20;
     SecuenciaCaracteres vector[TAMA];
     int utilizados;
public:
     /* Constructor por defecto
      * crea vector vacio
      */
     VectorSecuencias(){
         utilizados=0;
     }
     /**
      * Anade una nueva secuencia al vector
      * @param s secuencia a insertar
      */
     void aniade(SecuenciaCaracteres s){
            if(utilizados < TAMA){
            vector[utilizados++]=s;
            }
     }

     /**
      * Numero mÃ¡ximo de secuencias que se pueden almacenar en el vector
      * @return capacidad
      */
     int capacidad(){
         return TAMA;
     }
     /**
      * Devuelva la secuencia en la posicion dada por indice
      * @param indice posicion a consultar
      * @return secuencia en dicha posicion
      * @pre 0 \<= indice \< obtenerUtilizados()
      */
     SecuenciaCaracteres secuencia(int indice){
         return vector[indice];
     }
     /**
      * Numero de secuencias insertadas en el vector
      * @return  numero de secuencias en el vector
      */
     int obtenerUtilizados(){
         return utilizados;
     }


};

// Funciones genericas

/**
 * muestra una secuencia de caracteres en la salidad estandar
 * @param s secuencia a mostrar
 */
void pintaSecuencia(SecuenciaCaracteres s){
     for (int i=0;i<s.obtenerUtilizados();i++)
         cout << s.elemento(i);

}

/**
 * muestra un vector de secuencias en la salidad estandar
 * @param v vector a mostrar
 */
void pintaVector(VectorSecuencias v){
for (int i=0;i<v.obtenerUtilizados();i++){
     pintaSecuencia(v.secuencia(i));
     cout << endl;
}
}

/**
 * Cuenta el numero de secuencias palisimetricas en el vector
 * @param v el vector
 * @return numero de secs palisimetricas en v
 */
int cuantasPalisimetricas(VectorSecuencias v){
     int contador=0;
     for (int i=0; i<v.obtenerUtilizados(); i++){
         if (v.secuencia(i).esPalisimetrica())
              contador++;
     }
     return contador;
}


// Main

int main() {
     // se lee del flujo de entrada
     VectorSecuencias v;
     VectorSecuencias vpal, vnopal;
     bool continuar = true;
     while(continuar){
```

```
        string cadena;
        cin >> cadena;
        if(cadena == ""){
            continuar = false;
        }
        else{

            // se crea la secuencia
            SecuenciaCaracteres s;
            for(int i=0; i < cadena.length(); i++){
                s.aniade(cadena[i]);
            }
            v.aniade(s);

        }

    }
    cout << "Secuencias Iniciales"<< endl;
    pintaVector(v);
    cout << endl;

    for (int i=0; i<v.obtenerUtilizados();i++){
        // se hace la comprobacion
        bool cond = v.secuencia(i).esPalisimetrica();
        if (cond) {
            vpal.aniade(v.secuencia(i));
        } else {
            vnopal.aniade(v.secuencia(i));
        }
    }
    int np = cuantasPalisimetricas(v);
    cout << "Hay "<<  np << " y " << v.obtenerUtilizados()-np << " No Palisimetricas"<<endl;

    cout << endl << "Secs Palisimetricas"<<endl;
    pintaVector(vpal);
    cout << endl;
    cout << "Secs No Palisimetricas"<< endl;
    pintaVector(vnopal);
    cout<< endl;

    return 0;
}
```

secuencia.cpp

```
gaga
rotor
aabcaba
abbaab
```

palisimetrica.dat

```
PROJECT_NAME = "Palisimetrica"
PROJECT_NUMBER = 0
OUTPUT_LANGUAGE = SPANISH
OUTPUT_DIRECTORY = ./doc
INPUT = ./
RECURSIVE = YES
FILE_PATTERNS = *.cpp *.h
SOURCE_BROWSER = YES
INLINE_SOURCES = YES
HAVE_DOT = YES
CALL_GRAPH = YES
UML_LOOK = YES
```

secuencia.doxy