

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-1310 Sistemas Operativos
Grupo 01
I Semestre

**I Tarea programada: Embellecedor de código
fuentes**

Profesor:
Francisco Arroyo

Estudiantes:
Fabián Álvarez Chévez | B60369

13 de Abril del 2018

Índice

1. Introducción	3
2. Objetivos	3
3. Descripción	4
4. Diseño	5
5. Desarrollo	6
6. Manual de usuario	7
Requerimientos de Software	7
Compilación	7
Especificación de las funciones del programa	7
7. Casos de Prueba	8

1. Introducción

El tema que trata esta tarea programada es el embellecimiento del código fuente de un programa, pues normalmente muchos programadores desarrollan malas prácticas a la hora de programar causando una dificultad a la hora de reutilizar el código, o en el caso donde se quiera volver más eficiente o se le quieran realizar cambios, tomaría mucho tiempo, pues entender el código del programa toma su tiempo. Entonces el propósito de este programa es facilitar la vida de los programadores, transformando códigos escritos de forma desordenada y transformarlos a códigos ordenados y legibles.

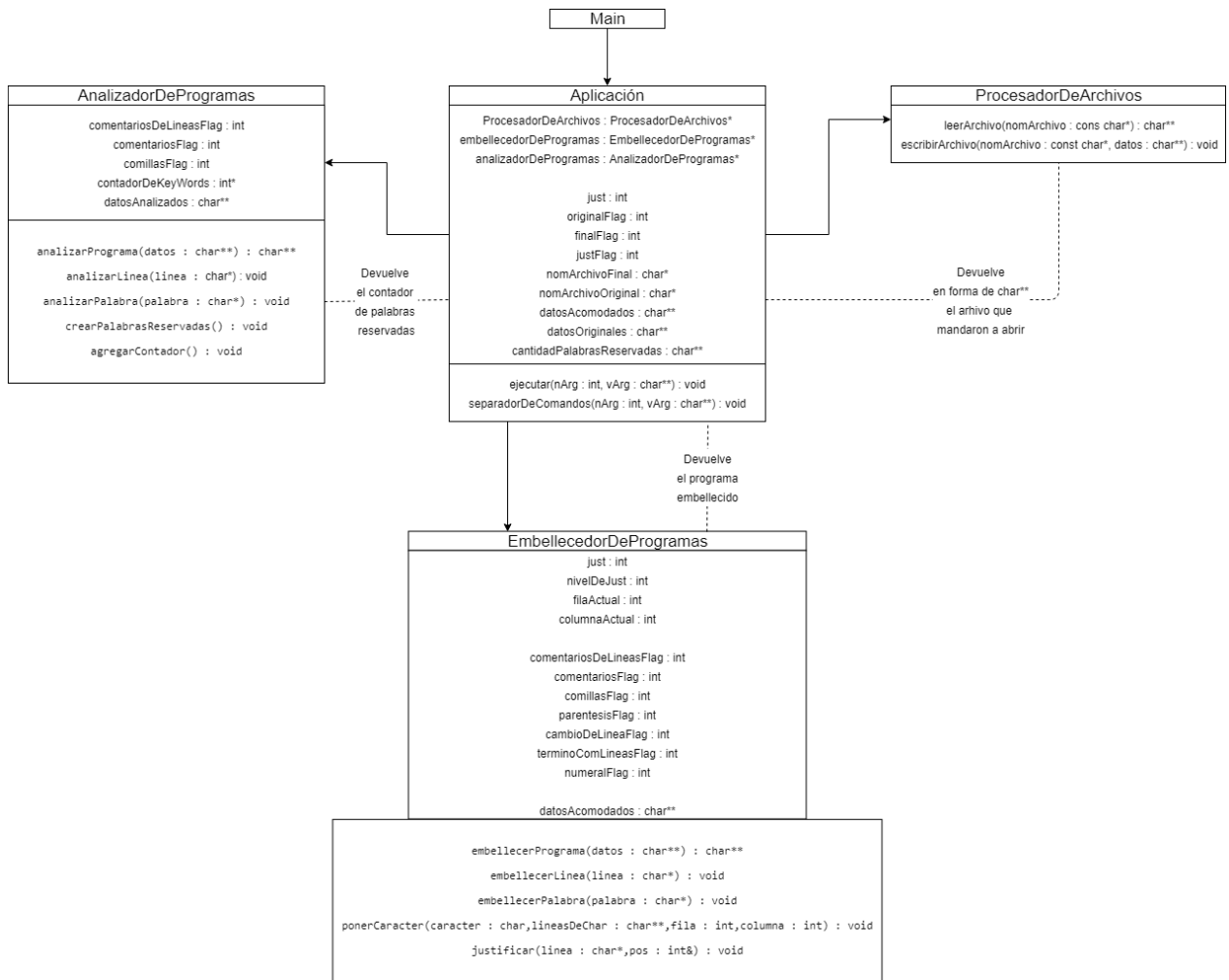
2. Objetivos

- Familiarizar al estudiante con los ambientes y herramientas de programación disponibles en el sistema operativo UNIX, al menos con algunos de sus sabores: Linux.

3. Descripción

- El estudiante debe desarrollar un programa en lenguaje de programación C o C++ en cualquier ambiente (Linux, Mac, UNIX, DOS, WINxx, etc.) pero debe correr adecuadamente en los equipos Linux de la ECCI. Los programas a justificar son programas fuentes de lenguaje C, Java o C++
- Para esta tarea programada el estudiante debe desarrollar de manera individual un embellecedor de programas fuentes escritos en el lenguaje de programación C, C++ o Java. Para ello se tendrá que:
 - Generar un nuevo archivo con los resultados
 - Agregar o eliminar espacios
 - Justificar de manera adecuada las estructuras sintácticas de los programas (class, try, if, do, while, for, etc.) de tal manera que se aprecie en el archivo resultante cómo están anidadas
 - No debe tomar en cuenta lo que se encuentre entre comillas
 - Separar las instrucciones se encuentran en una misma línea, en una por línea y justificarlas correctamente
 - Dejar los comentarios tal y como se encuentran
 - Contar las palabras reservadas del lenguaje y hacer un listado ordenado alfabéticamente de las palabras encontradas y la cantidad de veces que apareció en el archivo, por ejemplo <if, x >. El resultado de este conteo se deberá guardar en otro archivo
- Parámetros
 - El nombre del archivo fuente a justificar, puede especificarse en la línea de comandos, verificar que el archivo exista. Si no se especifica entonces el programa debe utilizar la entrada estándar
 - De la misma manera, el nombre del programa de salida puede indicarse en la línea de comandos con la opción -o nombre, si no se hace el programa debe utilizar su salida estándar
 - Se puede indicar como parámetro la cantidad de espacios a utilizar para justificar las instrucciones (-e n), donde n representa la cantidad de espacios, si no se especifica se debe utilizar un valor predeterminado, debe manejar adecuadamente los negativos
- En resumen la sintaxis del comando sería la siguiente:
 - **just [-e5] [archivo_original] [-o archivo_final]**
 - donde los paréntesis cuadrados indican partes opcionales y el parámetro 5 indica los espacios a dejar para anidar instrucciones
- También el programa se podrá llamar de la siguiente manera:
 - **just [-e3] <archivo_original >archivo_final**

4. Diseño



5. Desarrollo

La solución de este problema requiere una clase que pueda leer y escribir documentos, se necesita una clase que embellezca el documento, para esto vamos a utilizar banderas de control con las que se sabrá si se está leyendo un comentario de una sola línea o de varias, si está dentro de o si se debe realizar un cambio de líneas. Todo esto para manejar mejor que es lo que debería hacer el embellecedor para acomodar correctamente el archivo, esta clase va ir descomponiendo el programa en líneas y luego en términos, para simplificar a la hora de escoger como embellecer el programa. Hay otra clase por aparte que se encarga de analizar y contar las palabras reservadas del programa, este al igual que el embellecedor necesita de banderas para saber en que secciones si debe tomar en cuenta la palabra reservada, por otra parte de ser capas de crear un char** que contenga toda esta información. Al final la unión de estas clases en una aplicación que las controla y dirige permitirá resolver el problema que se plantea.

6. Manual de usuario

Requerimientos de Software

- Sistema Operativo: Linux
- Arquitectura: 64 bits
- Ambiente: Terminal

Compilación

Para compilar el programa, se utiliza `gcc` en la siguiente sentencia:

```
$ g++ -o just main.cpp Aplicacion.cpp AnalizadorDeProgramas.cpp EmbellecedorDeProgramas.cpp  
ProcesadorDeArchivos.cpp
```

Especificación de las funciones del programa

El programa solo puede embellecer código de tipo `c++`, haciendo que el único tipo de archivos que pueda abrir es con terminación `.cpp`

7. Casos de Prueba

Prueba 1:

Esta prueba verifica el funcionamiento del programa cuando se le indica correctamente un archivo donde se debe guardar la forma embellecida del código.

```
C:\Users\fakeb\Desktop\Universidad\IV Semestre\Sistemas Operativos\EmbellecedorDeProgramas>just Aplicacion.cpp -o Hola.cpp
Porfvor escriba el valor para justificar: 8
```

Figura 1: Salida en consola del Embellecedor de programas.

```
/*
 * param: int con el número de argumentos escritos desde la consola
 * param: char** con la lista de argumentos escritos desde la consola
 *
 * Funcio: es el encargado de ejecutar el programa
 */
void Aplicacion::ejecutar(int nArg, char** vArg){
    separadorDeComandos(nArg,vArg);
    datosOriginales = procesadorDeArchivos->leerArchivo(nomArchivoOriginal);
    if(datosOriginales){
        datosAcomodados = embellecedorDeProgramas->embellecerPrograma(datosOriginales);
        if(finalFlag){
            procesadorDeArchivos->escribirArchivo(nomArchivoFinal,datosAcomodados);
        }else{
            for(int i = 0; datosAcomodados[i] != 0; ++i){
                cout << datosAcomodados[i] << endl;
            }
        }
        cantidadPalabrasReservadas = analizadorDeProgramas->analizarPrograma(datosAcomodados);
        procesadorDeArchivos->escribirArchivo("Contador.txt",cantidadPalabrasReservadas);
    }
}
```

Figura 2: Forma en la que queda el código después de ser embellecido.

```
Contador.txt x AnalizadorDeProgramas.cpp x Aplicacion.cpp x foo.cpp x
30 break: 0
31 case: 0
32 catch: 0
33 char: 5
34 class: 0
35 const: 0
36 const_cast: 0
37 continue: 0
38 default: 0
39 delete: 4
40 do: 0
41 double: 0
42 dynamic_cast: 0
43 else: 5
44 enum: 0
45 explicit: 0
46 export: 0
47 extern: 0
48 false: 0
49 float: 0
50 for: 2
51 friend: 0
52 goto: 0
53 if: 8
54 inline: 0
55 int: 5
56 long: 0
57 mutable: 0
58 namespace: 1
59 new: 5
60 operator: 0
61 private: 0
62 protected: 0
63 public: 0
64
```

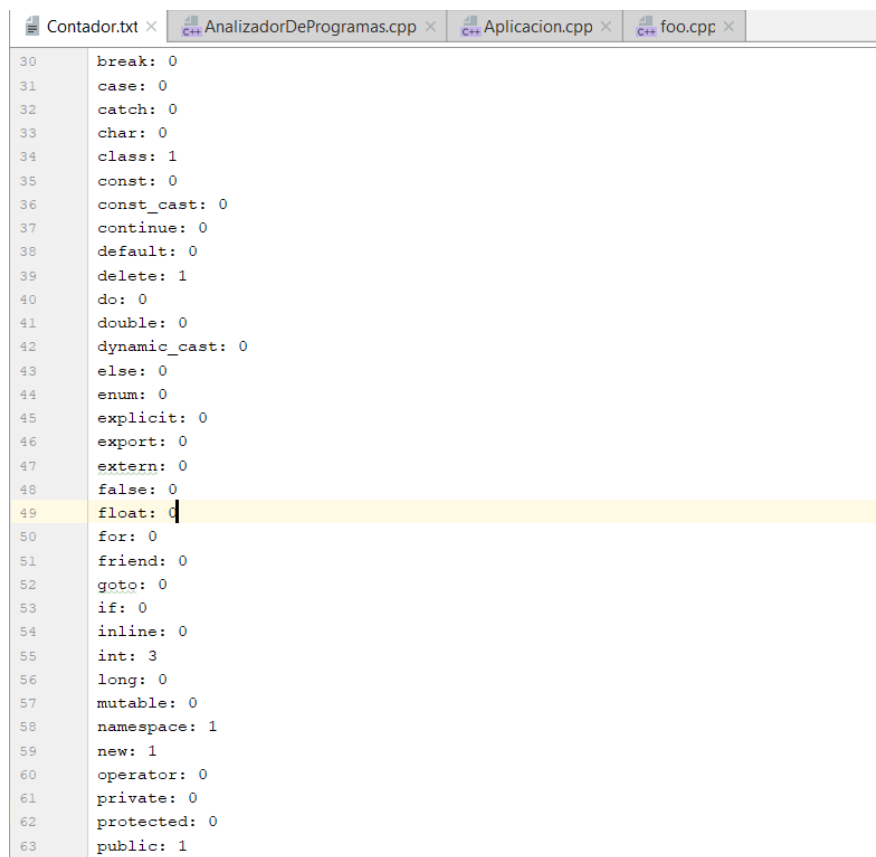
Figura 3: Contador de as palabraas reservadas

Prueba 2:

Esta prueba verifica el funcionamiento del programa cuando no se le indica correctamente un archivo donde se debe guardar la forma embellecida del código.

```
C:\Users\fakeb\Desktop\Universidad\V Semestre\Sistemas Operativos\EmbelledorDeProgramas>just -e 5 -o Jaja.txt foo.cpp
El tipo de archivo no es .cpp
#include <iostream>
using namespace std;
class foo {
    int x;
public:
    foo(int _x):x({})
    ~foo() {
        cout << "I am destroyed: " << x << endl;
    }
};
foo z(1);
int main() {
    // Los comentarios deben quedar tal y como estan, ;}, ;}, ;}
    // No hacer caso de comillas " u otras cosas ' { (
    foo * x = new foo(2);
    delete x;
    foo y(3);
    return 0;
}
```

Figura 4: Salida en consola del Embellecedor de programas.



```
30 break: 0
31 case: 0
32 catch: 0
33 char: 0
34 class: 1
35 const: 0
36 const_cast: 0
37 continue: 0
38 default: 0
39 delete: 1
40 do: 0
41 double: 0
42 dynamic_cast: 0
43 else: 0
44 enum: 0
45 explicit: 0
46 export: 0
47 extern: 0
48 false: 0
49 float: 0
50 for: 0
51 friend: 0
52 goto: 0
53 if: 0
54 inline: 0
55 int: 3
56 long: 0
57 mutable: 0
58 namespace: 1
59 new: 1
60 operator: 0
61 private: 0
62 protected: 0
63 public: 1
```

Figura 5: Forma en la que queda el código después de ser embellecido.

Referencias

- [1] BOS, V., AND MAUW, S. *A \LaTeX macro package for Message Sequence Charts—Maintenance document—Describing version*, June 2002. Included in MSC macro package distribution.
- [2] BOS, V., AND MAUW, S. *A \LaTeX macro package for Message Sequence Charts—Reference Manual—Describing version*, June 2002. Included in MSC macro package distribution.
- [3] BOS, V., AND MAUW, S. *A \LaTeX macro package for Message Sequence Charts—User Manual—Describing version*, June 2002. Included in MSC macro package distribution.
- [4] GOOSSENS, M., RAHTZ, S., AND MITTELBACH, F. *The \LaTeX Graphics Companion*. Addison-Wesley, 1997.
- [5] ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 1997.
- [6] LAMPORT, L. *\LaTeX —A Document Preparation System—User's Guide and Reference Manual*, 2nd ed. Addison-Wesley, 1994. Updated for $\LaTeX 2_{\epsilon}$.
- [7] RUDOLPH, E., P. GRAUBMANN, AND GRABOWSKI, J. Tutorial on message sequence charts (MSC'96). In *FORTE* (1996).