

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

CI-1310 Sistemas Operativos
Grupo 02
I Semestre

Tarea programada 0

Profesor:
Francisco Arroyo

Estudiantes:
Luis Porras Ledezma | B65477

11 de abril del 2018

Índice

1. Introducción	3
2. Objetivo	3
3. Descripción	4
4. Diseño	5
5. Desarrollo	6
6. Manual de usuario	8
Requerimientos de Software	8
Compilación	8
Especificación de las funciones del programa	8
7. Casos de Prueba	10

1. Introducción

En esta tarea se pretende crear un programa en `C++` funcional en Linux. Lo que hace el programa es justificar(embellecer) un código en `C++` dado como parámetro (nombre del archivo) por la terminal, el resultado de la ejecución del programa, es decir el código justificado con *e* espacios, se guarda en otro archivo (tanto *e* como el nombre del archivo de output se pueden pasar por la terminal). Además se quiere que el programa cuente el número de palabras reservadas del lenguaje en el código que recibe e indique en un documento aparte estos datos (palabra reservado - número de veces que apareció). Como se puede apreciar para desarrollar esta tarea se debe de tener un poco de conocimiento sobre el manejo de archivos de texto en Linux y en como Linux pasa los parámetros de la terminal al programa (`int argc, char* argv[]`).

2. Objetivo

- 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 ECCL. 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

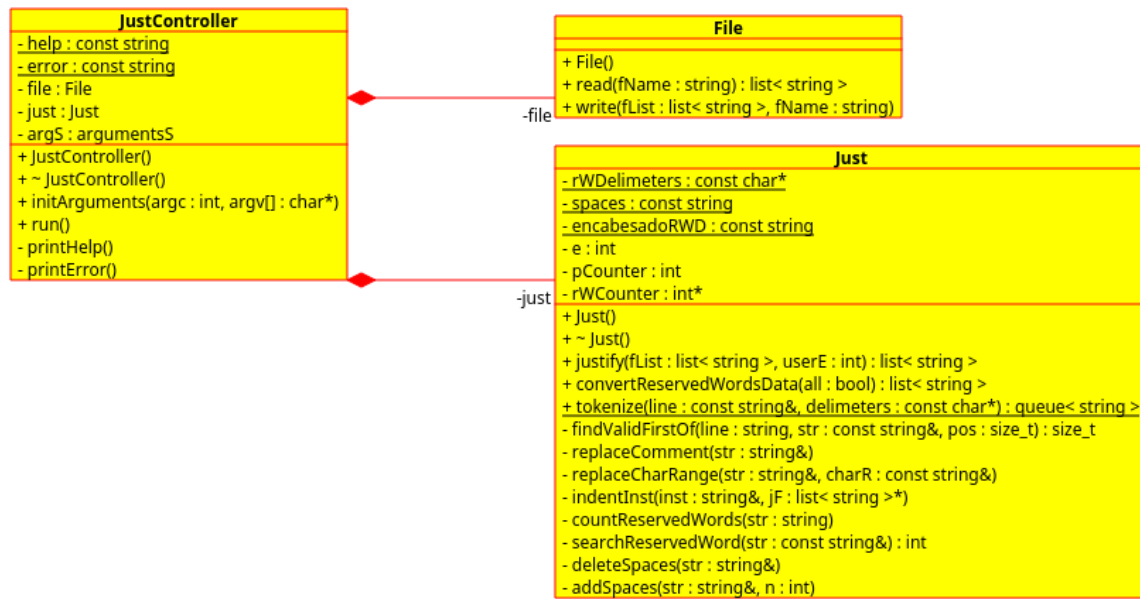


Figura 1: Diagrama UML

5. Desarrollo

La tarea se resolvió utilizando tres clases, tal como se ve en el diagrama (Figura 1). Donde *JustController* es la clase controlador que instancia a un objeto de la clase *File* para que escriba o lea de un archivo y a un objeto de la clase *Just* para realizar todo el proceso de embellecimiento de código. A continuación se explica los detalles más importantes de como funciona cada clase:

■ *justController*

Es la clase controlador del programa y utiliza los siguientes campos:

- *file* de tipo *File* es utilizado para leer y escribir en un archivo.
- *just* de tipo *Just* es utilizado para justificar el archivo leído y para obtener el registro de palabras reservadas encontradas.
- *argS* de tipo *argumentsS* una estructura definida en la misma clase, se utiliza para guardar la información de los argumentos pasados por la terminal.

Los métodos principales de esta clase son:

- *initArguments*, recibe por parámetros: *intarc* y *char * argv[]*. Se encarga de analizar los argumento con el que se llamo al programa y guardarlos en *argS*. No devuelve nada.
- *run*, no recibe ni devuelve nada. Se encarga de analizar la estructura *argS* para saber que argumentos pasó o no el usuario y realiza ciertas acciones dependiendo de eso, también revisa si hay algún error en el paso de parámetros o si el usuario pidió ayuda. Luego de eso llama el método *read* de *file*, pasa el archivo leído a *just* (*justify*) para que lo justifique y luego llama al método *write* de *file* para que escriba el archivo ya justificado que devuelve *just*, finalmente le pide a *just* las estadísticas sobre las palabras reservadas (*convetReservedWordData*) y llama nuevamente por ultima vez al metodo *write* de *file*.

■ *file*

Esta clase se encarga de leer y escribir de un archivo. No cuenta con ningún campo. Solo posee dos métodos:

- *read*, recibe el nombre del archivo a leer. Revisa si el nombre del archivo es "std::cin", si lo se lee de la entrada estandar, si no se lee del archivo, si el archivo no existe se reporta un error. Devuelve un *std :: list < std :: string >* con el contenido del archivo.
- *write*, recibe el nombre del archivo a escribir y una *std :: list < std :: string >* con el contenido a escribir. Al igual que su contra parte, se revisa si el nombre del archivo es "std::cout", si lo es se escribe en la salida estándar, si no en el archivo pasado por parámetro, si el archivo existe se le escribe en sima, sino se crea. No devuelve nada.

■ *Just*

Esta clase se encarga de lo fuerte del programa, es la que se encarga de embellecer el código y llevar el conteo de las palabras reservadas. Sus principales campos son:

- *e* de tipo *int* contiene el número de espacios con los que se desea indentar el codigo.
- *pCounter* de tipo *int* cuenta el numero de paréntesis abiertos en el código, se utiliza para indentar correctamente el código.
- *rWCounters* de tipo *int** es un array en memoria dinámica que lleva el conteo de cada palabra reservada en el código. Tiene una relación uno a uno con *rWords* un array de string global y estático que contiene todas la palabras reservadas de C ++.

Sus principales métodos son:

- *justify*, recibe: *std :: list < std :: string > fList* una lista de strings con el contenido del archivo leído, y *int userE* el numero de espacios con el que se quiere indentar el código.
justify en general lo que hace es, procesar cada string(linea) de *fList*, separar la linea en instrucciones individuales y mandar cada instrucción individual a indentar. Las instrucciones se separan buscando "{ } ;" cualquier de estos caracteres, por lo tanto se toman en cuenta casos especiales (comentarios, for, etc).
 Se devuelve una *std :: list < std :: string >* con el contenido del archivo ya justificado.
- *findValidFirstOf*, recibe: *line* de tipo *std :: string* es la linea en la que quiero encontrar algún carácter. *str* de tipo *std :: string &* los caracteres que quiero encontrar en *line*. Y *pos* de tipo *size_t* es la posición desde donde quiero empezar a buscar en *line*.
 Encuentra el primer carácter de los que indica *str* en *line* a partir de la posición *pos*, se asegura que ese caracter encontrado sea válido es decir que no este en un comentario o en medio de " o '. Para ello llama a los métodos: *replaceComment* renplasa todo un comentario por NULL_CHAR. Y *replaceCharRange* reemplaza lo que esta en medio de un carácter, especificado por parámetros, por NULL_CHAR
 Devuelve la posición del primer carácter encontrado y valido, en caso contrario devuelve *std :: string :: npos*.
- *indentInst*, recibe: *inst* de tipo *std :: string &* la instrucción a indentar. *jF* de tipo *std :: list < std :: string > &* donde guardar la instrucción ya indentada.
 Revisa si *inst* es un paréntesis que abre o cierra o y aumenta o disminuye el *pCounter* respectivamente. Y llama a los métodos: *deleteSpaces* (elimina los espacios en blanco al comienzo de un *string*), *addSpaces* (agrega *n* espacios al comienzo de un *string*) para indentar correctamente *inst*. También llama al método *countReservedWords*.
 No devuelve nada.
- *countReservedWords* recibe: *str* de tipo *std :: string* donde se quiere buscar una palabra reservada y aumentar su contador.
 Separa *str* en tokens con *just :: tokanize* utilizando *rWDelimeters* como delimitadores para el tokenizer. Luego busca por cada palabra si esta es una palabra reservada y si lo es, aumenta su contador, la búsqueda se hace llamando a *searchReservedWords*.
 No devuelve nada.
- *convertReservedWordsData* recibe: *all* un booleano que indica si se desea listar todas las palabras reservadas y no solo aquellas que aparecieron en el programa archivo.
 Convierte el contenido de *rWords* y *rWCounters* que tienen una relación uno a uno en un *std :: list < std :: string >* donde se listan las palabras reservadas con su respectivo contador.
 Devuelve un *std :: list < std :: string >* con los datos sobre las palabras reservadas y su respectivo contador.

6. Manual de usuario

Requerimientos de Software

- **Sistema Operativo:** [Linux]
- **Arquitectura:** [32 bits, 64 bits]
- **Ambiente:** [Consola (Shell)]

Compilación

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

```
g++ -o just main.cpp File.cpp JustController.cpp Just.cpp -Wall -g -O0 -lm -I. -std=c++11
```

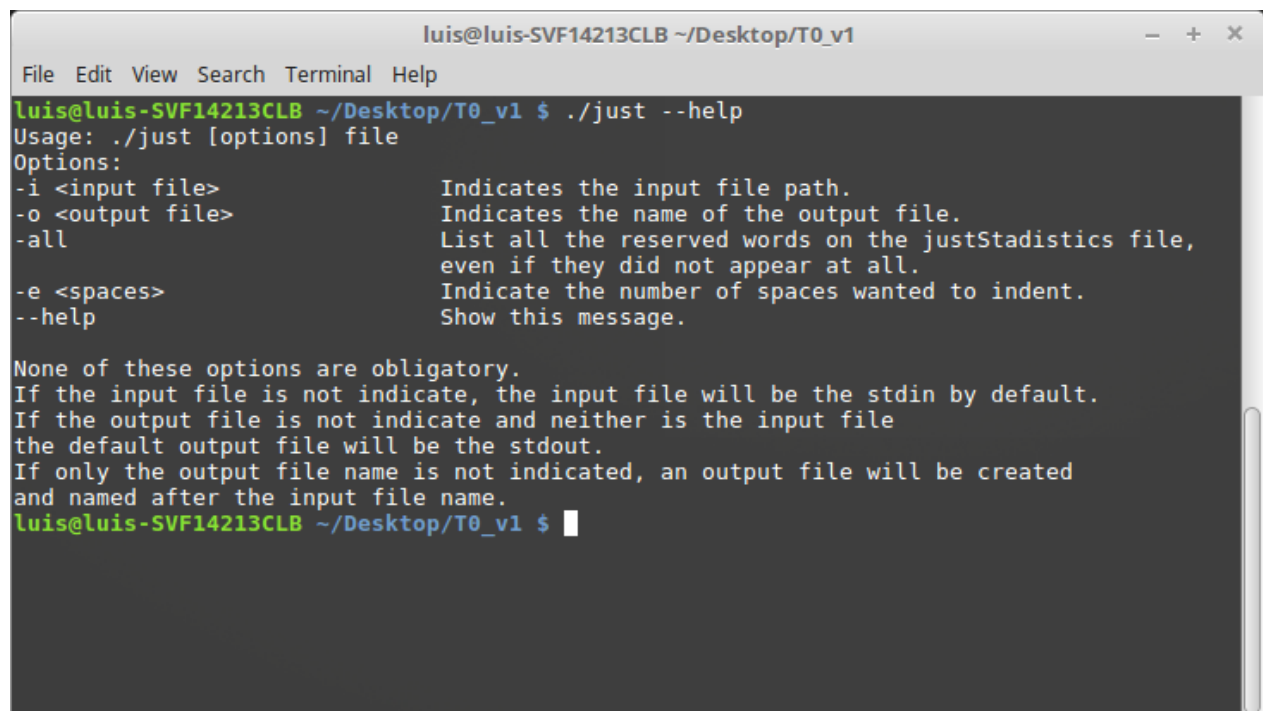
Para facilitar la compilación de la tarea esta viene con un archivo *Makefile* para ejecutarlo solo debe escribir:

```
make
```

en la consola, eso sí estando en el mismo directorio que el *Makefile* (obviamente).

Especificación de las funciones del programa

Las especificaciones de como llamar al programa se muestran en la ayuda del programa, de esta forma:



```
luis@luis-SVF14213CLB ~/Desktop/T0_v1
File Edit View Search Terminal Help
luis@luis-SVF14213CLB ~/Desktop/T0_v1 $ ./just --help
Usage: ./just [options] file
Options:
-i <input file>           Indicates the input file path.
-o <output file>          Indicates the name of the output file.
-all                     List all the reserved words on the justStadistics file,
                           even if they did not appear at all.
-e <spaces>               Indicate the number of spaces wanted to indent.
--help                   Show this message.

None of these options are obligatory.
If the input file is not indicate, the input file will be the stdin by default.
If the output file is not indicate and neither is the input file
the default output file will be the stdout.
If only the output file name is not indicated, an output file will be created
and named after the input file name.
luis@luis-SVF14213CLB ~/Desktop/T0_v1 $
```

Figura 2: Ayuda del programa

Dentro de las restricciones que tiene el programa cabe destacar:

- Toda instrucción *if*, *for*, *while*, etc, debe tener corchetes aún cuando esta solo tenga una instrucción en su interior. Es decir el programa NO indenta una instrucción de esta forma:

```
if(a == b)
    c = d;
```

- El programa solo separa instrucciones, por lo que cualquier cosa que NO termine en "; { }", NO sera separada, esto incluye a los *#include*, *#define*, *public :*, *private :*, *case :*, etc.
- El programa respeta linea en blanco y espacios en blanco dentro de una linea de código.

7. Casos de Prueba

Caso de prueba 1

Esta prueba verifica la indentación del código *TestCase1.cpp*, que es el caso de prueba que venia en el enunciado de la tarea.

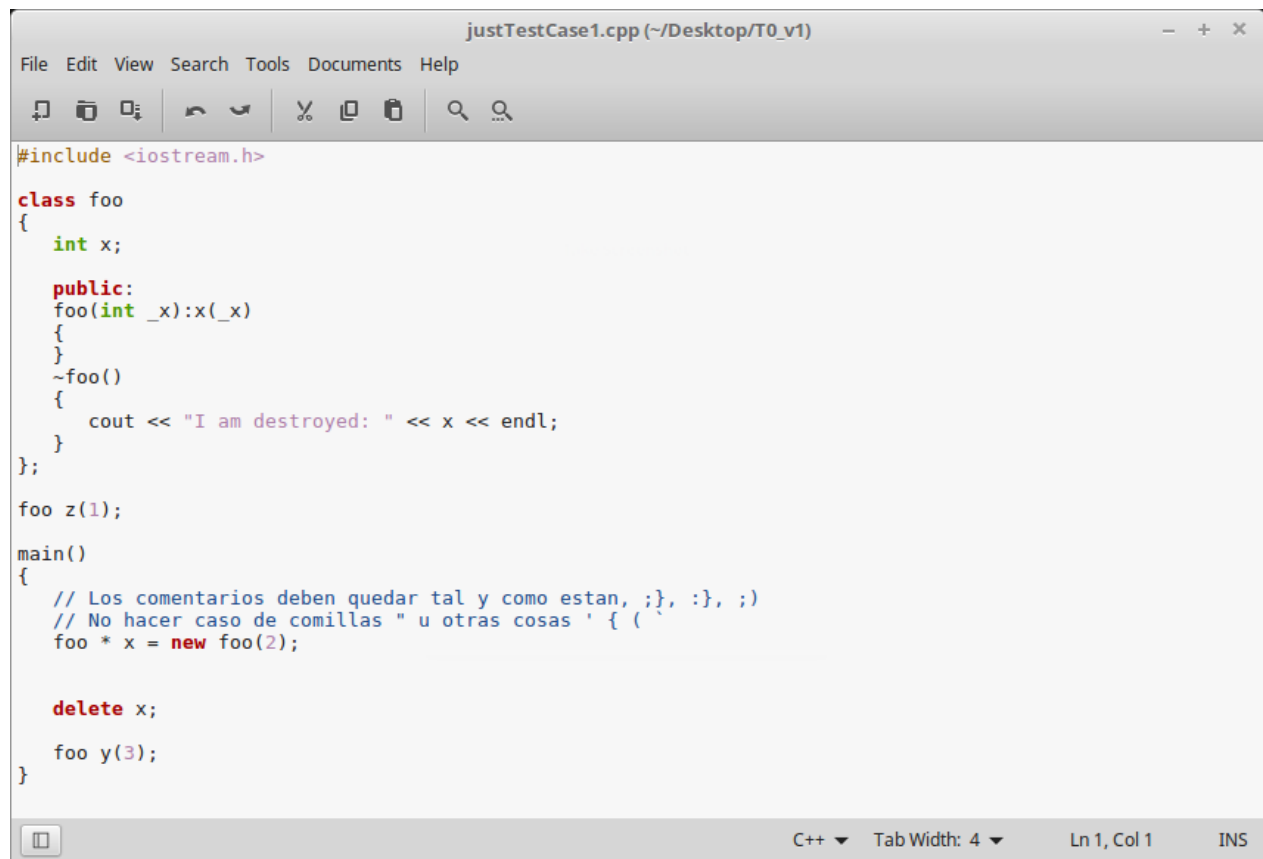
El código utilizado corresponde a:

```
1  #include <iostream.h>
2
3      class foo
4  {
5      int x;
6
7  public:
8      foo(int _x):x(_x) {}
9      ~foo() { cout << "I am destroyed: " << x << endl; }
10 };
11
12 foo z(1);
13
14 main()
15 {
16     // Los comentarios deben quedar tal y como estan, ;}, :}, ;)
17     // No hacer caso de comillas " u otras cosas ' { ( `
18     foo * x = new foo(2);
19
20
21
22
23     foo y(3);
24 }
```

El programa fue llamado de esta manera:

```
./just -i TestCases/TestCase1.cpp -e 3
```

Y el resultado obtenido se puede verificar en la figura 3:



```
justTestCase1.cpp (~/Desktop/T0_v1)
File Edit View Search Tools Documents Help
[Icons]
#include <iostream.h>

class foo
{
    int x;

public:
    foo(int _x):x(_x)
    {
    }
    ~foo()
    {
        cout << "I am destroyed: " << x << endl;
    }
};

foo z(1);

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);
}
```

C++ Tab Width: 4 Ln 1, Col 1 INS

Figura 3: Salida del caso de prueba 1

Como se puede apreciar el código se indento correctamente con 3 espacios, además como no se indicó el nombre del archivo de salida, este se creo y nombro en base del archivo de entrada (input: TestCase1.cpp, output: justTestCase1.cpp). Eso sí el archivo de salida siempre se crea en el folder donde se llamo al programa.

Caso de prueba 2

Esta prueba verifica la separación de instrucciones e indentación del código *TestCase2.cpp*, que es el caso de prueba que venia en el enunciado de la tarea, pero modificado.

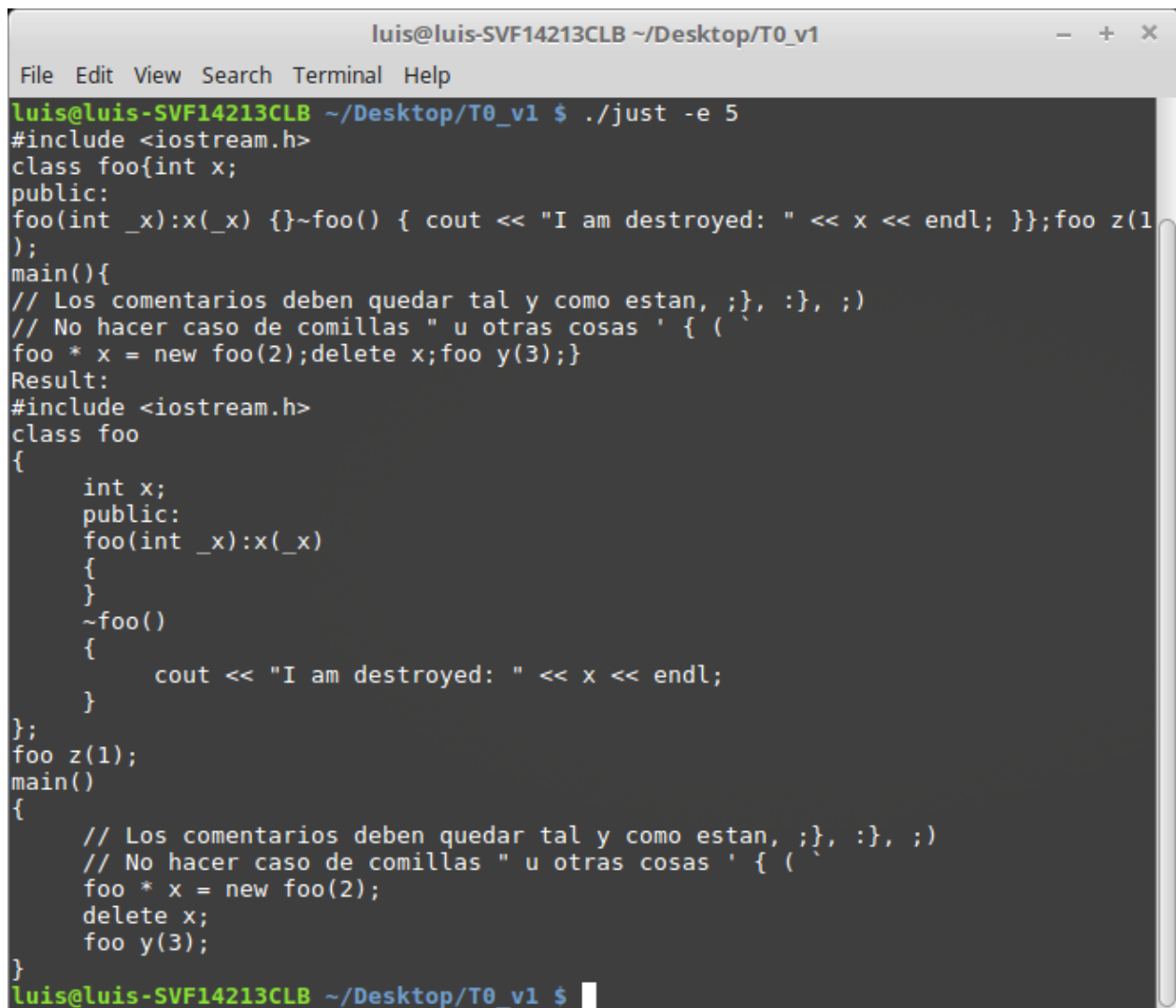
El código utilizado corresponde a:

```
1  #include <iostream.h>
2  class foo{int x;
3  public:
4  foo(int _x):x(_x) {}~foo() { cout << "I am destroyed: " << x << endl; }};foo z(1);
5  main(){
6  // Los comentarios deben quedar tal y como estan, ;}, :}, ;)
7  // No hacer caso de comillas " u otras cosas ' { ( `
8  foo * x = new foo(2);delete x;foo y(3);}
```

El programa fue llamado de esta manera:

```
./just -e 5
```

Y el resultado obtenido se puede verificar en la figura 4:



```
luis@luis-SVF14213CLB ~/Desktop/T0_v1
File Edit View Search Terminal Help
luis@luis-SVF14213CLB ~/Desktop/T0_v1 $ ./just -e 5
#include <iostream.h>
class foo{int x;
public:
foo(int _x):x(_x) {}~foo() { cout << "I am destroyed: " << x << endl; }};foo z(1);
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);}
Result:
#include <iostream.h>
class foo
{
    int x;
public:
    foo(int _x):x(_x)
    {
    }
    ~foo()
    {
        cout << "I am destroyed: " << x << endl;
    }
};
foo z(1);
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);
}
luis@luis-SVF14213CLB ~/Desktop/T0_v1 $
```

Figura 4: Salida del caso de prueba 2

Como se puede apreciar el código se indento correctamente con 5 espacios, además como no se indicó ni el path para el archivo de entrada, ni el nombre del archivo de salida, el programa asumió como archivo de entrada la stdin y como archivo de salida stdout.

Caso de prueba 3

Esta prueba verifica la indentación diferentes instrucciones del código *TestCase3.cpp*, que es el mismo código de la clase *justContoller.cpp*.

El código utilizado corresponde a:

```
1  #include "JustController.h"
2
3  const string JustController::help = "Usage: ./just [options] file\n"
4                                     "Options:\n"
5                                     "-i <input file>           Indicates the input\n"
6                                     "    file path.\n"
7                                     "-o <output file>       Indicates the name\n"
8                                     "    of the output file.\n"
9                                     "-all                  List all the\n"
10                                    "    reserved words on the justStadistics file,\n"
11                                    "    even if they did\n"
12                                    "    not appear at all.\n"
13                                    "-e <spaces>           Indicate the number\n"
14                                    "    of spaces wanted to indent.\n"
15                                    "--help               Show this message.\n"
16                                    "    n"\n"
17                                    "None of these options are obligatory.\n"
18                                    "If the input file is not indicate, the input file\n"
19                                    "    will be the stdin by default.\n"
20                                    "If the output file is not indicate and neither is\n"
21                                    "    the input file\n"
22                                    "the default output file will be the stdout.\n"
23                                    "If only the output file name is not indicated, an\n"
24                                    "    output file will be created \n"
25                                    "and named after the input file name.";
26
27 const string JustController::error = "Error: Invalid arguments";
28
29 JustController::JustController(){}
30
31 JustController::~~JustController(){}
32
33 void JustController::initArguments(int argc, char* argv[]){
34
35     for(int ind = 1; ind < argc; ++ind){
36         if(0 == strcmp(argv[ind], "--help")){
37             argS.isHelpAsked = true;
38             return;
39         }
40
41         if(0 == strcmp(argv[ind], "-i")){
42             if(ind + 1 < argc){
43                 if(argS.inFileName.empty()){
44                     argS.inFileName = argv[++ind];
45                 }
46             }
47             else{
48                 argS.errorCode = -1;
49                 return;
50             }
51         }
52     }
53 }
```

```

43
44     else if(0 == strcmp(argv[ind], "-o")){
45         if(ind + 1 < argc){
46             if(argS.outFileName.empty()){
47                 argS.outFileName = argv[++ind];
48             }
49         }
50         else{
51             argS.errorCode = -1;
52             return;
53         }
54     }
55
56     else if(0 == strcmp(argv[ind], "-e")){
57         if(ind + 1 < argc){
58             argS.e = atoi(argv[++ind]);
59         }
60         else{
61             argS.errorCode = -1;
62             return;
63         }
64     }
65
66     else if(0 == strcmp(argv[ind], "-all")){
67         argS.all = true;
68     }
69
70     else{
71         if(argS.inFileName.empty()){
72             argS.inFileName = argv[ind];
73         }
74     }
75 }
76
77
78 void JustController::run(){
79     if(argS.isHelpAsked){
80         printHelp();
81         return;
82     }
83
84     if(-1 == argS.errorCode){
85         printError();
86         return;
87     }
88
89     if(argS.e <= 0){
90         argS.e = DEFAULT_E;
91     }
92
93     if(!argS.inFileName.empty() && argS.outFileName.empty()){
94
95         string inFileName = argS.inFileName;
96         size_t found = inFileName.find_last_of("/\\");
97         if(found != string::npos){
98             inFileName = inFileName.substr(found + 1);

```

```

99     }
100     queue<string> inFileNameTokens = Just::tokenize(inFileName, ".");
101     string outFileName = "just";
102     outFileName.append(inFileNameTokens.front()+".");
103     outFileName.append(inFileNameTokens.back());
104
105     argS.outFileName = outFileName;
106 }
107
108 else if(argS.inFileName.empty() && !argS.outFileName.empty()){
109     argS.inFileName = "std::cin";
110 }
111
112 else if(argS.inFileName.empty() && argS.outFileName.empty()){
113     argS.inFileName = "std::cin";
114     argS.outFileName = "std::cout";
115 }
116
117 list<string> fL = file.read(argS.inFileName);
118 list<string> jF = just.justify(fL, argS.e);
119 list<string> rWDL = just.convertReservedWordsData(argS.all);
120 file.write(jF, argS.outFileName);
121 file.write(rWDL, "justStatistics.txt");
122 }
123
124 void JustController::printHelp(){
125     cout << help << '\n';
126 }
127
128 void JustController::printError(){
129     cerr << error << '\n';
130 }

```

El programa fue llamado de esta manera:

```
./just TestCases/TestCase3.cpp -o Result.cpp -e 7 -all
```


Y el resultado obtenido fue el siguiente, en un archivo llamado Result.cpp:

```
1  #include "JustController.h"
2
3  const string JustController::help = "Usage: ./just [options] file\n"
4  "Options:\n"
5  "-i <input file>           Indicates the input file path.\n"
6  "-o <output file>          Indicates the name of the output file.\n"
7  "-all                      List all the reserved words on the justStatistics file
8  ,\n"
9  "                          even if they did not appear at all.\n"
10 "-e <spaces>               Indicate the number of spaces wanted to indent.\n"
11 "--help                    Show this message.\n"
12 "\n"
13 "None of these options are obligatory.\n"
14 "If the input file is not indicate, the input file will be the stdin by default.\n"
15 "If the output file is not indicate and neither is the input file\n"
16 "the default output file will be the stdout.\n"
17 "If only the output file name is not indicated, an output file will be created \n"
18 "and named after the input file name.";
19
20 const string JustController::error = "Error: Invalid arguments";
21
22 JustController::JustController()
23 {
24 }
25
26 JustController::~~JustController()
27 {
28 }
29
30 void JustController::initArguments(int argc, char* argv[])
31 {
32     for(int ind = 1; ind < argc; ++ind)
33     {
34         if(0 == strcmp(argv[ind], "--help"))
35         {
36             argS.isHelpAsked = true;
37             return;
38         }
39
40         if(0 == strcmp(argv[ind], "-i"))
41         {
42             if(ind + 1 < argc)
43             {
44                 if(argS.inFileName.empty())
45                 {
46                     argS.inFileName = argv[++ind];
47                 }
48             }
49             else
50             {
51                 argS.errorCode = -1;
52                 return;
53             }
54         }
55     }
56 }
```

```

54
55     else if(0 == strcmp(argv[ind], "-o"))
56     {
57         if(ind + 1 < argc)
58         {
59             if(argS.outFileName.empty())
60             {
61                 argS.outFileName = argv[++ind];
62             }
63         }
64         else
65         {
66             argS.errorCode = -1;
67             return;
68         }
69     }
70
71     else if(0 == strcmp(argv[ind], "-e"))
72     {
73         if(ind + 1 < argc)
74         {
75             argS.e = atoi(argv[++ind]);
76         }
77         else
78         {
79             argS.errorCode = -1;
80             return;
81         }
82     }
83
84     else if(0 == strcmp(argv[ind], "-all"))
85     {
86         argS.all = true;
87     }
88
89     else
90     {
91         if(argS.inFileName.empty())
92         {
93             argS.inFileName = argv[ind];
94         }
95     }
96 }
97
98
99 void JustController::run()
100 {
101     if(argS.isHelpAsked)
102     {
103         printHelp();
104         return;
105     }
106
107     if(-1 == argS.errorCode)
108     {
109         printError();

```

```

110         return;
111     }
112
113     if(argS.e <= 0)
114     {
115         argS.e = DEFAULT_E;
116     }
117
118     if(!argS.inFileName.empty() && argS.outFileName.empty())
119     {
120
121         string inFileName = argS.inFileName;
122         size_t found = inFileName.find_last_of("/\\");
123         if(found != string::npos)
124         {
125             inFileName = inFileName.substr(found + 1);
126         }
127         queue<string> inFileNameTokens = Just::tokenize(inFileName, ".");
128         string outFileName = "just";
129         outFileName.append(inFileNameTokens.front()+".");
130         outFileName.append(inFileNameTokens.back());
131
132         argS.outFileName = outFileName;
133     }
134
135     else if(argS.inFileName.empty() && !argS.outFileName.empty())
136     {
137         argS.inFileName = "std::cin";
138     }
139
140     else if(argS.inFileName.empty() && argS.outFileName.empty())
141     {
142         argS.inFileName = "std::cin";
143         argS.outFileName = "std::cout";
144     }
145
146     list<string> fL = file.read(argS.inFileName);
147     list<string> jF = just.justify(fL, argS.e);
148     list<string> rWDL = just.convertReservedWordsData(argS.all);
149     file.write(jF, argS.outFileName);
150     file.write(rWDL, "justStatistics.txt");
151 }
152
153 void JustController::printHelp()
154 {
155     cout << help << '\n';
156 }
157
158 void JustController::printError()
159 {
160     cerr << error << '\n';
161 }

```

En esta ocasión no puse un screenshot por el tamaño del documento result. Además para este ejemplo el resultado del archivo justStatistics.txt es:

Number of reserved words found:

alignas	:	0
alignof	:	0
and	:	0
and_eq	:	0
asm	:	0
auto	:	0
bitand	:	0
bitor	:	0
bool	:	0
break	:	0
case	:	0
catch	:	0
char	:	1
char16_t	:	0
char32_t	:	0
class	:	0
compl	:	0
const	:	2
constexpr	:	0
const_cast	:	0
continue	:	0
decltype	:	0
default	:	0
delete	:	0
do	:	0
double	:	0
dynamic_cast	:	0
else	:	9
enum	:	0
explicit	:	0
export	:	0
extern	:	0
false	:	0
float	:	0
for	:	1
friend	:	0
goto	:	0
if	:	18
inline	:	0
int	:	2
long	:	0
mutable	:	0
namespace	:	0
new	:	0
noexcept	:	0
not	:	0
not_eq	:	0

nullptr	:	0
operator	:	0
or	:	0
or_equ	:	0
private	:	0
protected	:	0
public	:	0
register	:	0
reinterpret_cast	:	0
return	:	6
short	:	0
signed	:	0
sizeof	:	0
static	:	0
static_assert	:	0
static_cast	:	0
struct	:	0
switch	:	0
template	:	0
this	:	0
thread_local	:	0
throw	:	0
true	:	2
try	:	0
typedef	:	0
typeid	:	0
typename	:	0
union	:	0
unsigned	:	0
using	:	0
virtual	:	0
void	:	4
volatile	:	0
wchar_t	:	0
while	:	0
xor	:	0
xor_eq	:	0