

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

CI-1310 Sistemas Operativos
Grupo 02
I Semestre

I Tarea programada

Profesor:
Francisco Arroyo

Estudiantes:
Luis Porras Ledezma | B65477

11 de Mayo del 2018

Índice

1. Introducción	3
2. Objetivos	3
3. Descripción	4
4. Diseño	5
5. Desarrollo	6
6. Manual de usuario	9
Compilación	9
Especificación de las funciones del programa	9
7. Casos de Prueba	11

1. Introducción

En esta tarea se va a tomar el trabajo anterior (Tarea 0) y se le va a agregar concurrencia, es decir ahora el programa just va a recibir varios archivos (por la terminal) como parámetros, just va a crear un proceso por cada archivo y este proceso va a hacer lo que hacia la Tarea 0 a ese archivo, es decir, va a leer el código en C++ del archivo y producir un nuevo archivo "inputFileName.sgr" con el código embellecido. Sin embargo en esta versión cada subproceso (hijo) no va a crear un archivo con las estadísticas, sino que va a enviar mensajes al padre pasando los datos sobre las palabras reservadas encontradas en su archivo correspondiente. El padre luego recibe y procesa los mensajes de sus hijos, suma todos los datos, es decir, todos los contadores de cada palabra reservada, luego él escribe estos datos en un área de memoria compartida y crea otro hijo(subproceso) para que lea e imprima el contenido de la memoria compartida en la pantalla (este hijo imprime cada vez que el padre le indica que lo haga, es decir, cuando el padre ya haya recibido y sumado todas la palabras reservadas que empiezan con una letra específica). Para realizar esta tarea se van a utilizar herramientas para la comunicación entre procesos, estos últimos creados con la llamada al sistema "fork()", los IPC que se utilizaran son semáforos, memoria compartida y paso de mensajes (buzón).

2. Objetivos

- Lograr que el estudiante aplique los conceptos adquiridos sobre los tópicos de comunicación entre procesos, particularmente aquellos concernientes a comunicación mediante semáforos, memoria compartida y paso de mensajes.
- Desarrollar un programa en el lenguaje de programación C++, orientado al sistema operativo UNIX que utilice todas las facilidades para la comunicación entre procesos (IPC) en ese ambiente.

3. Descripción

- Debe construirse un programa que creará un número determinado de procesos (fork) que le ayudarán a completar su tarea.
- La función de este utilitario es la de contar la totalidad de palabras claves (if, for, switch, while, do) que aparecen en los archivos indicados por el usuario, semejante a la primera tarea programada, pero esta vez con varios archivos.
- La sintaxis del programa es la siguiente:
 - sangria [-e 5] Nombre_Archivo1 Nombre_Archivo2...

4. Diseño



Figura 1: Diagrama UML

5. Desarrollo

A continuación se explica la manera en la que se implemento la solución al problema que plantea la tarea, mediante la implementación de 4 clases tal y como se ven en el diagrama (Figura 1).

Explicación sobre cada clase (¿Qué hace? y ¿Como lo hace? (campos y métodos)):

Clase: *Just*

- Función:

Esta clase es la encargada de recibir y analizar los argumentos pasados por el usuario en la terminal. También es la encargada de crear un subprocesso por cada archivo de entrada recibido y hacer que este indente su archivo correspondiente. Además es el encargado de crear al subprocesso que imprime en pantalla y coordinares con este para que vaya imprimiendo en la salida es tardar las estadísticas de las palabras reservadas.

- Campos:

- *pCounter*: tipo *int*, cuenta el número de llaves abiertas.
- *file*: tipo *File*, instancia de la clase *File* utilizada para leer y escribir en un archivo.
- *arS*: tipo *ArgStruct*, almacena los posibles argumentos pasados por el usuario en la terminal al llamar al programa.
- *rWStructure*: de tipo *map < string, int >*, diccionario con las palabras reservadas y su respectivo contador.

- Métodos:

- *void justify(int argc, char * argv[])*
Indenta múltiples archivos pasados por el usuario (*argc*, *argv*) e imprime la suma de los datos sobre el numero de veces que aparece cada palabra reservada en cada archivo. Este método no devuelve nada.
Recibe como parámetros:
 - *argc*: tipo *int*, numero de argumentos pasados por el usuario.
 - *argv*: tipo *char * **, arreglo de parámetros pasados por el usuario.
- *static queue < string > tokenize(const string &str, const char * delimiters)*
Separa un *string* en tokens, el *string* es separado por los caracteres pasados como delimitadores. Este método regresa los tokens como un *queue < string >*.
Recibe como parámetros:
 - *const string &str*, el *string* a separar en tokens.
 - *const char * delimiters*, los caracteres por los que se quiere separar el *string*.
- *string getOutputFileName(const string& iFName)*
Genera un nombre para el archivo de salida (output filename) basado en nombre o path de archivo de entrada (input filename). Este método devuelve un *string* con el nombre generado para el archivo de salida.
Recibe como parámetros:
 - *const string& iFName*, nombre o path del archivo de entrada.
- *void indent(const string& iFName, const string &oFName, Message &m, int sonID)*
Indenta el código del archivo con path *iFName* y genera un archivo de salida con el resultado. Este método no devuelve nada.
Recibe como parámetros:
 - *const string& iFName*, nombre o path del archivo a indentar.

- *const string &oFName*, nombre para el archivo de salida con el código ya indentado.
- *Message &m*, buzón al cual enviar los datos sobre las palabras reservadas y cuantas veces se encontraron en el archivo.
- *int sonID*, identificador del subproceso.
- *list < string > splitInstructions(string &line)*
Separa una línea de código en sus respectivas instrucciones individuales. Este método devuelve una *list < string >* con las instrucciones individuales.
Recibe como parámetros:
 - *string &line*, línea de código a ser separada en instrucciones.
- *void indentInstruction(list < string > *justList, string instruction)*
Indenta una instrucción de código C++. Este método no devuelve nada.
Recibe como parámetros:
 - *list < string > *justList*, lista de *string* donde colocar la instrucción ya indentada.
 - *string instruction*, copia de la instrucción a indentar.
- *void deleteSpaces(string& str)*
Elimina los espacios en blanco al comienzo de un *string*. Este método no devuelve nada.
Recibe como parámetros:
 - *string& str*, *string* al cual eliminar los espacios en blanco.
- *void addSpaces(string& str, int n)*
Agrega *n* espacios en blanco al comienzo de un *string*. Este método no devuelve nada.
Recibe como parámetros:
 - *string& str*, *string* al cual agregar los espacios en blanco.
 - *int n*, número de espacios en blanco a agregar.
- *void countReservedWords(const string &instruction)*
Revisa si ahí una o más palabras reservadas en la instrucción y si las ahí aumenta sus contadores. Este método no devuelve nada.
Recibe como parámetros:
 - *const string &instruction*, instrucción a revisar.
- *size_t findFirstValidOf(const string& line, const string& str, size_t pos = 0)*
Encuentra el primer carácter de *str* en *line* que no está en un comentario o entre ' o ". Este método devuelve la posición del carácter encontrado, si ningún carácter de *str* es encontrado entonces se devuelve *std::string::npos*.
Recibe como parámetros:
 - *const string& line*, *string* en el que se desea buscar.
 - *const string& str*, *string* con los caracteres a buscar.
 - *size_t pos*, posición desde donde se desea empezar a buscar, es 0 por defecto.
- *size_t findValid(const string& line, const string& str, size_t pos = 0)*
Idéntico al método *size_t findFirstValidOf(const string& line, const string& str, size_t pos = 0)* solo que se busca la secuencia de caracteres exacta de *str* y no cualquiera de ellos.
- *void replaceComments(string& str)*
Reemplaza un comentario de la forma *"// ..."* por *NULL_C*. Este método no devuelve nada.
Recibe como parámetros:
 - *string& str*, *string* en la cual se desea reemplazar el comentario.
- *void replaceBetweenChar(string& str, const string& c)*
Reemplaza lo que sea que esté en medio de *c* por *NULL_C*. Este método no devuelve nada.
Recibe como parámetros:

- *string& str, string* en la que se desea reemplazar lo que esta en medio de *c*.
- *const string& c, string* que determina que reemplazar.
- *void initializeArguments(int argc, char *argv[])*
"Inicializa" la estructura *argS* con los argumentos pasados por el usuario. Este método no devuelve nada.
Recibe como parámetros:
 - *argc*: tipo *int*, numero de argumentos pasados por el usuario.
 - *argv*: tipo *char ***, arreglo de parámetros pasados por el usuario.
- *void printHelp()*
Imprime un mensaje de ayuda para el usuario.
- *printError()*
Imprime un mensaje de error para el usuario.
- *void sendReservedWordData(Message &m, int mtype)*
Envia los datos sobre las palabras reservadas (palabra reservada y su contador) al buzón *m* con el tipo de mensaje *mtype*. Este método no devuelve nada.
Recibe como parámetros:
 - *Message &m*, buzón al cual enviar los mensajes con los datos de las palabras reservadas.
 - *int mtype*, tipo de mensaje para los mensajes enviados por ese subproceso.
- *int countReservedWordsFirstLetters()*
Me indica el número de diferentes letras con la que empiezan las palabras reservadas. Devuelve ese número de letras distintas.

Clase: *File*

■ Función:

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 *list < string >* con el contenido del archivo.
- *write*, recibe el nombre del archivo a escribir y una *list < 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.

Las clases *Sem* y *Message* son las mismas clases que se hicieron en los laboratorios anteriores del curso. Y se utilizaron para la coordinación y comunicación entre procesos respectivamente.

6. Manual de usuario

- **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 Just.cpp Sem.cpp Message.cpp -Wall -g -O0 -lm -I. -std=c++11
```

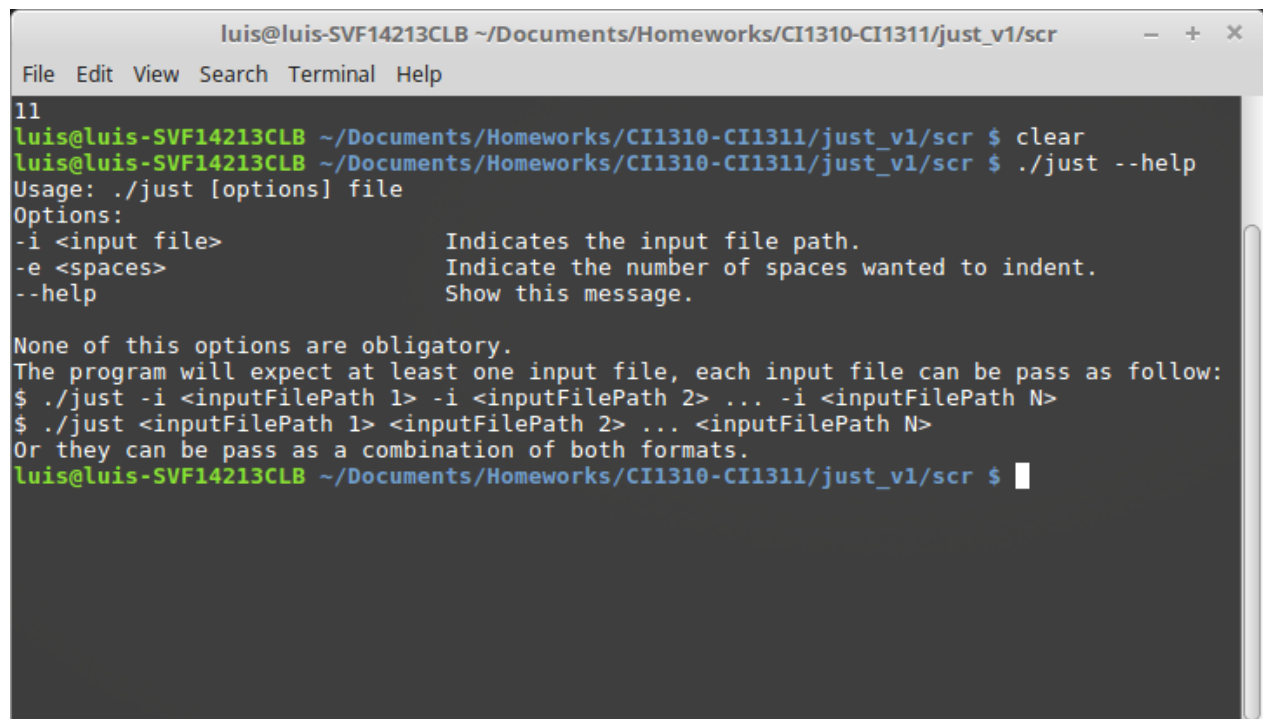
Para facilitar la compilación de la tarea esta viene con un archivo *Makefile* para ejecutarlo solo debe escribir:

```
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 ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr
File Edit View Search Terminal Help
11
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr $ clear
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr $ ./just --help
Usage: ./just [options] file
Options:
-i <input file>           Indicates the input file path.
-e <spaces>              Indicate the number of spaces wanted to indent.
--help                   Show this message.

None of this options are obligatory.
The program will expect at least one input file, each input file can be pass as follow:
$ ./just -i <inputFilePath 1> -i <inputFilePath 2> ... -i <inputFilePath N>
$ ./just <inputFilePath 1> <inputFilePath 2> ... <inputFilePath N>
Or they can be pass as a combination of both formats.
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr $
```

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.
- Se debe tener cuidado con el tamaño de los archivos que se pasan simultáneamente como parámetros, ya que el buzón se puede llenar y causar que el programa se caiga.

7. Casos de Prueba

Prueba 1:

En esta prueba se verifica la funcionalidad completa de la tarea.

El código pasado por parámetro corresponde a:

■ TestCase1.cpp:

```
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                                     delete x;
22
23     foo y(3);
24 }
```

■ TestCase2.cpp:

```
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);}
```

■ TestCase3.cpp

```
1  #include "Just.h"
2
3  //Static const variables initialization
4  const string Just::spaces = " \t";
5  const string Just::encabesadoRWD = "Number of reserved words found: ";
6  const char* Just::rWDelimiters = " (){}:<>*&";
7
8  Just::Just(): e(0), pCounter(0){
```

```

9     rWCounter = new int[M];
10 }
11
12 Just::~Just(){
13     delete[] rWCounter;
14 }
15
16 list<string>Just::justify(list<string> fList, int userE){
17
18     e = userE;
19     list<string>justFile;
20     string line;
21
22     //For each line of code
23     for(list<string>::iterator ind = fList.begin(); ind != fList.end(); ++ind){
24         line = *ind;
25
26         //First I need to check if there are multiple instructions per line. If
27         //there are I need to separate them into single instructions.
28         size_t actualP = 0;
29         size_t instP = findValidFirstOf(line, "{};", actualP);
30         string instruction;
31
32         while(instP != string::npos){
33
34             instruction = line.substr(actualP, instP-actualP+1);
35
36             //Special cases
37             //for loop
38             if(instruction.find("for ") != string::npos || instruction.find("for("
39                 ) != string::npos){
40                 if(instruction.back() == ';'){
41                     size_t pPos = findValidFirstOf(line, ")", instP + 1);
42                     if(pPos != string::npos){
43                         actualP = instP + 1;
44                         instP = pPos;
45                         instruction.append(line.substr(actualP, instP-actualP+1));
46                     }
47                 }
48             }
49             //};
50             if(instruction.back() == '}'){
51                 size_t sCP = line.find_first_not_of(spaces, instP+1);
52                 if(sCP != string::npos && line[sCP] == ';'){
53                     actualP = instP + 1;
54                     instP = sCP;
55                     instruction.append(line.substr(actualP, instP-actualP+1));
56                 }
57             }
58
59             actualP = instP+1;
60             instP = findValidFirstOf(line, "{};", actualP);
61
62             if(instP == string::npos && actualP < line.size()){
63                 string instComment = line.substr(actualP);
64                 if(instComment.find_first_not_of(spaces) != string::npos){

```

```

63         if(instComment.find("//") != string::npos){ //If
64             there if a comment at the end of the instruction
65             instruction.append(instComment);
66         }
67         else{ //If
68             there is another instruction
69             line = instComment;
70             actualP = 0;
71         }
72     }
73     //Seperate {} parenthesis from each instruction if needed
74     size_t pPos = findValidFirstOf(instruction, "{}");
75     if(pPos != string::npos){
76         string subInstruction = instruction.substr(pPos);
77         instruction.erase(pPos);
78         if(instruction.find_first_not_of(spaces) != string::npos){
79             indentInst(instruction, &justFile);
80         }
81         indentInst(subInstruction, &justFile);
82     }
83     else{
84         indentInst(instruction, &justFile);
85     }
86 }
87 //If the line is only a comment.
88 if(actualP == 0){
89     instruction = line;
90     indentInst(instruction, &justFile);
91 }
92 }
93
94 return justFile;
95 }
96
97 queue<string> Just::tokenize(const string &line, const char* delimiters){
98     queue<string> lineTokens;
99
100     char* cLine = new char [line.length()+1];
101     strcpy(cLine, line.c_str());
102     char* p = strtok(cLine, delimiters);
103     while(p != 0){
104         lineTokens.push(p);
105         p = strtok(NULL, delimiters);
106     }
107     delete[] cLine;
108
109     return lineTokens;
110 }
111
112 size_t Just::findValidFirstOf(string line, const string& str, size_t pos){
113     replaceComment(line);
114     replaceCharRange(line, "\"");
115     replaceCharRange(line, "'");
116

```

```

117     size_t strPos = line.find_first_of(str, pos);
118     return strPos;
119 }
120
121 void Just::replaceComment(string &str){
122     size_t pPos = str.find("//");
123     if(pPos != string::npos){
124         str.replace(pPos, str.size()-pPos, str.size()-pPos, NULL_C);
125     }
126 }
127
128 void Just::replaceCharRange(string &str, const string &charR){
129     size_t firstP = str.find_first_of(charR);
130     size_t secondP = 0;
131     while(firstP != string::npos){
132         secondP = str.find_first_of(charR, firstP + 1);
133         str.replace(firstP, secondP + 1 - firstP, secondP + 1 - firstP, NULL_C);
134         firstP = str.find_first_of(charR, secondP + 1);
135     }
136 }
137
138 void Just::indentInst(string& inst, list<string>* jF){
139     deleteSpaces(inst);
140
141     if(findValidFirstOf(inst, "{") != string::npos){
142         if(pCounter > 0){
143             --pCounter;
144         }
145     }
146
147     addSpaces(inst, e*pCounter);
148     jF->push_back(inst);
149
150     //I need to update the reserved words counters
151     countReservedWords(inst);
152
153     if(findValidFirstOf(inst, "{") != string::npos){
154         ++pCounter;
155     }
156 }
157
158 void Just::countReservedWords(string str){
159     if(!str.empty()){
160         //First I need to make sure I don't count reserved word in comments or
161         //between " or '.
162         replaceComment(str);
163         replaceCharRange(str, "\\");
164         replaceCharRange(str, "'");
165
166         //Now I need to separate str into token
167         queue<string> strTokens = tokenize(str, rWDelimiters);
168
169         //Finally I need to check if a tokens is a reserved word, and if it is,
170         //update it's counter.
171         int found = 0;
172         while(!strTokens.empty()){

```

```

171         found = searchReservedWord(strTokens.front());
172         if(found >= 0){
173             ++rWCounter[found];
174         }
175         strTokens.pop();
176     }
177 }
178 }
179
180 int Just::searchReservedWord(const string& str){
181     if(str.size() > 1){
182         for(int ind = 0; ind < M; ++ind){
183             if(rWords[ind].compare(str) == 0){
184                 return ind;
185             }
186         }
187     }
188     return -1;
189 }
190
191 void Just::deleteSpaces(string &str){
192     size_t found = str.find_first_not_of(spaces);
193     if(found!=string::npos){
194         str = str.substr(found);
195     }
196     else{
197         str.clear();
198     }
199 }
200
201 void Just::addSpaces(string &str, int n){
202     str.insert(str.begin(), n, ' ');
203 }
204
205 list<string> Just::convertReservedWordsData(bool all){
206     list<string> rWDataList;
207     rWDataList.push_back(encabesadoRWD + '\n');
208     char rWData[25];
209     for(int ind = 0; ind < M; ++ind){
210         if(all){
211             sprintf(rWData, "%-16s :%4d", rWords[ind].c_str(), rWCounter[ind]);
212             rWDataList.push_back(rWData);
213         }
214         else{
215             if(rWCounter[ind] > 0){
216                 sprintf(rWData, "%-16s :%4d", rWords[ind].c_str(), rWCounter[ind]);
217                 ;
218                 rWDataList.push_back(rWData);
219             }
220         }
221     }
222     return rWDataList;
223 }

```

El programa fue llamado de esta manera:

```
./just test/TestCase1.cpp test/TestCase2.cpp test/TestCase3.cpp -e 10
```


El resultado en la terminal al ejecutar el programa de esa forma fue:

```
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr $ ./just test/TestCase1.cpp test/TestCase2.cpp test/TestCase3.cpp -e 10
alignas      : 0
alignof      : 0
and          : 0
and_eq       : 0
asm          : 0
auto         : 0
bitand       : 0
bitor        : 0
bool         : 1
break        : 0
case         : 0
catch        : 0
char         : 6
char16_t     : 0
char32_t     : 0
class        : 2
compl        : 0
const        : 0
const_cast   : 0
constexpr    : 0
continue     : 0
decltype     : 0
default      : 0
delete       : 4
do           : 0
double       : 0
dynamic_cast : 0
else         : 4
enum         : 0
explicit     : 0
export       : 0
extern       : 0
false        : 0
float        : 0
for          : 3
friend       : 0
goto         : 0
```

Figura 3: Salida del caso de prueba en la terminal.

El resultado en la carpeta donde se ejecuto el programa fue:

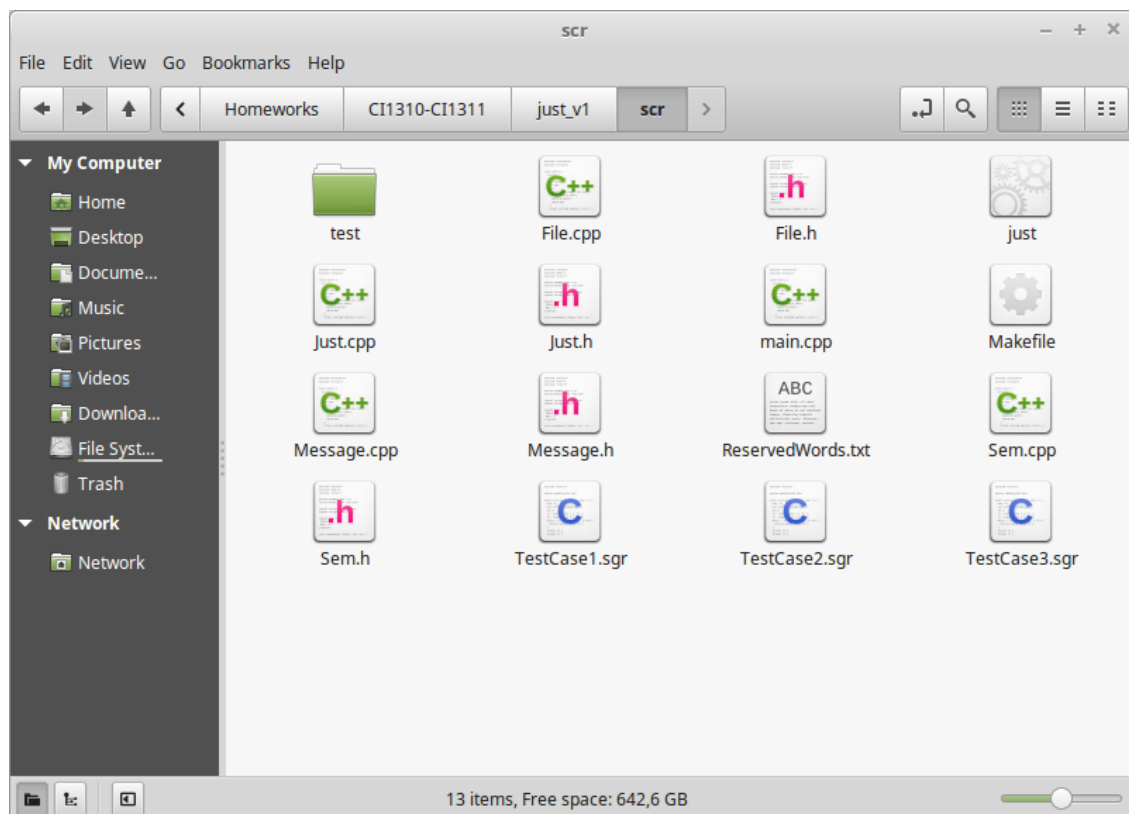


Figura 4: Archivos de salida del caso de prueba.

El contenido de los archivos de salida de este caso de prueba corresponde a:

■ TestCase1.sgr:

```
1  #include <iostream.h>
2
3  class foo
4  {
5      int x;
6
7      public:
8      foo(int _x):x(_x)
9      {
10     }
11     ~foo()
12     {
13         cout << "I am destroyed: " << x << endl;
14     }
15 };
16
17 foo z(1);
18
19 main()
20 {
21     // Los comentarios deben quedar tal y como estan, ;}, :}, ;)
22     // No hacer caso de comillas " u otras cosas ' { ( `
23     foo * x = new foo(2);
24
25
26     delete x;
27
28     foo y(3);
29 }
```

■ TestCase2.cpp:

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

```

20         delete x;
21         foo y(3);
22     }

```

■ TestCase3.cpp

```

1  #include "Just.h"
2
3  //Static const variables initialization
4  const string Just::spaces = " \t";
5  const string Just::encabesadoRWD = "Number of reserved words found: ";
6  const char* Just::rWDelimiters = " (){}:<>*&";
7
8  Just::Just(): e(0), pCounter(0)
9  {
10     rWCounter = new int[M];
11 }
12
13 Just::~Just()
14 {
15     delete[] rWCounter;
16 }
17
18 list<string>Just::justify(list<string> fList, int userE)
19 {
20
21     e = userE;
22     list<string>justFile;
23     string line;
24
25     //For each line of code
26     for(list<string>::iterator ind = fList.begin(); ind != fList.end(); ++
        ind)
27     {
28         line = *ind;
29
30         //First I need to check if there are multiple instructions per
            line. If there are I need to separate them into sigle
            instructions.
31         size_t actualP = 0;
32         size_t instP = findValidFirstOf(line,"{};", actualP);
33         string instruction;
34
35         while(instP != string::npos)
36         {
37
38             instruction = line.substr(actualP, instP-actualP+1);
39
40             //Special cases
41             //for loop
42             if(instruction.find("for ") != string::npos ||
                instruction.find("for(") != string::npos)
43             {
44                 if(instruction.back() == ';'')
45                     {

```

```

46         size_t pPos = findValidFirstOf(
47             line, ")", instP + 1);
48         if(pPos != string::npos)
49         {
50             actualP = instP + 1;
51             instP = pPos;
52             instruction.append(
53                 line.substr(actualP
54                     , instP-actualP+1))
55                 ;
56         }
57     }
58     //};
59     if(instruction.back() == '{}')
60     {
61         size_t sCP = line.find_first_not_of(spaces
62             , instP+1);
63         if(sCP != string::npos && line[sCP] == ';'
64             )
65         {
66             actualP = instP + 1;
67             instP = sCP;
68             instruction.append(line.substr(
69                 actualP, instP-actualP+1));
70         }
71     }
72     actualP = instP+1;
73     instP = findValidFirstOf(line,"{};", actualP);
74
75     if(instP == string::npos && actualP < line.size())
76     {
77         string instComment = line.substr(actualP);
78         if(instComment.find_first_not_of(spaces)
79             != string::npos)
80         {
81             if(instComment.find("//") !=
82                 string::npos){ //
83                 If there if a comment at the
84                 end of the instruction
85                 instruction.append(instComment);
86             }
87             else
88             {
89                 //If there is another instruction
90                 line = instComment;
91                 actualP = 0;
92             }
93         }
94     }
95
96     //Separete {} parenthesis from each instruction if needed
97     size_t pPos = findValidFirstOf(instruction, "{}");
98     if(pPos != string::npos)

```

```

89         {
90             string subInstruction = instruction.substr(pPos);
91             instruction.erase(pPos);
92             if(instruction.find_first_not_of(spaces) != string::npos)
93             {
94                 indentInst(instruction, &justFile);
95             }
96             indentInst(subInstruction, &justFile);
97         }
98     else
99     {
100         indentInst(instruction, &justFile);
101     }
102 }
103 //If the line is only a comment.
104 if(actualP == 0)
105 {
106     instruction = line;
107     indentInst(instruction, &justFile);
108 }
109 }
110
111 return justFile;
112 }
113
114 queue<string> Just::tokenize(const string &line, const char* delimiters)
115 {
116     queue<string> lineTokens;
117
118     char* cLine = new char [line.length()+1];
119     strcpy(cLine, line.c_str());
120     char* p = strtok(cLine, delimiters);
121     while(p != 0)
122     {
123         lineTokens.push(p);
124         p = strtok(NULL, delimiters);
125     }
126     delete[] cLine;
127
128     return lineTokens;
129 }
130
131 size_t Just::findValidFirstOf(string line, const string& str, size_t pos)
132 {
133     replaceComment(line);
134     replaceCharRange(line, "\\");
135     replaceCharRange(line, "'");
136
137     size_t strPos = line.find_first_of(str, pos);
138     return strPos;
139 }
140
141 void Just::replaceComment(string &str)
142 {
143     size_t pPos = str.find("//");

```

```

144         if(pPos != string::npos)
145         {
146             str.replace(pPos, str.size()-pPos, str.size()-pPos, NULL_C);
147         }
148     }
149
150 void Just::replaceCharRange(string &str, const string &charR)
151 {
152     size_t firstP = str.find_first_of(charR);
153     size_t secondP = 0;
154     while(firstP != string::npos)
155     {
156         secondP = str.find_first_of(charR, firstP + 1);
157         str.replace(firstP, secondP + 1 - firstP, secondP + 1 - firstP
158             , NULL_C);
159         firstP = str.find_first_of(charR, secondP + 1);
160     }
161
162 void Just::indentInst(string& inst, list<string>* jF)
163 {
164     deleteSpaces(inst);
165
166     if(findValidFirstOf(inst, "{") != string::npos)
167     {
168         if(pCounter > 0)
169         {
170             --pCounter;
171         }
172     }
173
174     addSpaces(inst, e*pCounter);
175     jF->push_back(inst);
176
177     //I need to update the reserved words counters
178     countReservedWords(inst);
179
180     if(findValidFirstOf(inst, "(") != string::npos)
181     {
182         ++pCounter;
183     }
184 }
185
186 void Just::countReservedWords(string str)
187 {
188     if(!str.empty())
189     {
190         //First I need to make sure I don't count reserved word in
191         //comments or between " or '.
192         replaceComment(str);
193         replaceCharRange(str, "\"");
194         replaceCharRange(str, "'");
195
196         //Now I need to separate str into token
197         queue<string> strTokens = tokenize(str, rWDDelimiters);

```

```

198         //Finally I need to check if a tokens is a reserved word, and
        if it is, update it's counter.
199         int found = 0;
200         while(!strTokens.empty())
201         {
202             found = searchReservedWord(strTokens.front());
203             if(found >= 0)
204             {
205                 ++rWCounter[found];
206             }
207             strTokens.pop();
208         }
209     }
210 }
211
212 int Just::searchReservedWord(const string& str)
213 {
214     if(str.size() > 1)
215     {
216         for(int ind = 0; ind < M; ++ind)
217         {
218             if(rWords[ind].compare(str) == 0)
219             {
220                 return ind;
221             }
222         }
223     }
224     return -1;
225 }
226
227 void Just::deleteSpaces(string &str)
228 {
229     size_t found = str.find_first_not_of(spaces);
230     if(found!=string::npos)
231     {
232         str = str.substr(found);
233     }
234     else
235     {
236         str.clear();
237     }
238 }
239
240 void Just::addSpaces(string &str, int n)
241 {
242     str.insert(str.begin(), n, ' ');
243 }
244
245 list<string> Just::convertReservedWordsData(bool all)
246 {
247     list<string> rWDataList;
248     rWDataList.push_back(encabesadoRWD + '\n');
249     char rWData[25];
250     for(int ind = 0; ind < M; ++ind)
251     {
252         if(all)

```

```

253         {
254             sprintf(rWData, "%-16s :%4d", rWords[ind].c_str(),
255                     rWCounter[ind]);
256             rWDataList.push_back(rWData);
257         }
258         else
259         {
260             if(rWCounter[ind] > 0)
261             {
262                 sprintf(rWData, "%-16s :%4d", rWords[ind].
263                     c_str(), rWCounter[ind]);
264                 rWDataList.push_back(rWData);
265             }
266         }
267     }
268     return rWDataList;

```

Prueba 2: En esta prueba se verifica el manejo de errores del programa pasándole como parámetro archivos inválidos (no existentes).

El programa fue llamado de esta manera:

```
./just invalidFileName1 invalidFileName2 invalidFileName3
```

La salida en pantalla de correr el programa de esa manera fue:

```

luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr
luis@luis-SVF14213CLB ~/Documents/Homeworks/CI1310-CI1311/just_v1/scr $ ./just invalidFileName1 invalidFileName2 invalidFileName3
Unable to open file
Unable to open file
Unable to open file

```

Figura 5: Salida de pasar archivos inexistentes como parámetros

Si alguno de los hijos no puede abrir el archivo envía al padre los mensajes de las palabras reservadas y sus contadores siempre en 0.