**Diseño**

El diseño básico involucra dos clases: un simple\_input\_parser y un simple\_loc\_counter; la función main () simplemente invoca un simple\_loc\_counter para analizar la entrada estándar.

La clase simple\_input\_parser es realmente simple: una vez que se le asigna un flujo de entrada, una llamada a simple\_input\_parser :: parse\_until\_eof lee las líneas, una a la vez, las transforma de acuerdo con transform\_line y almacena el resultado en m\_last\_line. Después de leer cada línea, parse\_until\_eof llama a parse\_last\_line para actuar en la última línea leída y transformada.

La clase simple\_loc\_counter subclases simple\_input\_parser; la transformación aplicada a cada línea es simple: se omiten los espacios en blanco iniciales y finales. Cada línea se clasifica según una serie de funciones como last\_line\_is\_comment, last\_line\_is\_compiler\_directive, etc. Un contador de enteros se encarga de los comentarios de bloque "anidados".

La clase simple\_loc\_counter agrega cada LOC a un búfer de líneas contables y simplemente devuelve el tamaño de ese búfer (en LOC) cuando se le pregunta. También puede imprimir cada línea contable, útil durante la prueba.

**Code**

**simple\_input\_parser.h**

|  |
| --- |
| /\*  \*/  #ifndef SIMPLE\_INPUT\_PARSER\_H  #define SIMPLE\_INPUT\_PARSER\_H  #include <string>  #include <iostream>  //a standard "framework" for parsing a set of input lines  class simple\_input\_parser  {  public:  //sets input stream  void set\_input\_stream (istream \* new\_input);  //parse input until EOF  void parse\_until\_eof (void);  //reads single line from input, transforms it, stores it in last\_line  void read\_line (void);  //sets the last line read  void set\_last\_line (const std::string & new\_line);  //last line read from input  const std::string & last\_line (void) const;  //parses the last line read  virtual void parse\_last\_line (void);  //returns a "transformed" copy of the given line  virtual std::string transformed\_line (const std::string & line) const;  //constructor  simple\_input\_parser (void);  //virtual destructor  virtual ~ simple\_input\_parser (void);  //resets the parser  virtual void reset (void);  private:  //the input stream  istream \* m\_input\_stream;  //the last line of input  std::string m\_last\_line;  };  #endif  /\*  \*/ |

**simple\_input\_parser.cpp**

|  |
| --- |
| /\*  \*/  #include "simple\_input\_parser.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  void  simple\_input\_parser::set\_input\_stream (istream \* new\_input)  {  m\_input\_stream = new\_input;  }  void  simple\_input\_parser::parse\_until\_eof (void)  {  REQUIRE (m\_input\_stream != NULL);  while (!(m\_input\_stream->eof ()))  {  read\_line ();  parse\_last\_line ();  }  }  void  simple\_input\_parser::read\_line (void)  {  REQUIRE (!(m\_input\_stream->eof ()));  const int  input\_buffer\_size = 255;  char  input\_buffer[input\_buffer\_size];  m\_input\_stream->getline (input\_buffer, input\_buffer\_size);  //for some reason, the G++ standard library needs me to do this or it  //doesn't register the EOF condition properly. This makes no sense  //to me...  char c = m\_input\_stream->get();  m\_input\_stream->putback( c );  std::string input\_line (input\_buffer);  //no source code line should be longer than 255!  CHECK (input\_line.size () < 255);  set\_last\_line (transformed\_line (input\_line));  }  void  simple\_input\_parser::set\_last\_line (const std::string & new\_line)  {  m\_last\_line = new\_line;  }  const  std::string & simple\_input\_parser::last\_line (void) const  {  return m\_last\_line;  }  void  simple\_input\_parser::parse\_last\_line (void)  {  //basic version does nothing  }  std::string simple\_input\_parser::transformed\_line (const std::string & line) const  {  return line;  }  simple\_input\_parser::simple\_input\_parser (void)  {  reset ();  }  simple\_input\_parser::~simple\_input\_parser (void)  {  }  void  simple\_input\_parser::reset (void)  {  m\_input\_stream = NULL;  m\_last\_line = "";  }  /\*  \*/ |

**simple\_loc\_counter.h**

|  |
| --- |
| /\*  \*/  #ifndef SIMPLE\_LOC\_COUNTER\_H  #define SIMPLE\_LOC\_COUNTER\_H  #ifndef SIMPLE\_INPUT\_PARSER\_H  #include "simple\_input\_parser.h"  #endif  #include <string>  #include <vector>  //subclass of simple\_input\_parser that stores countable lines of code in a buffer  //and can return their count.  class simple\_loc\_counter:public simple\_input\_parser  {  public:  //adds last line to the buffered lines if it is countable  void parse\_last\_line (void);  //the count of LOC  int loc\_count (void) const;  //whether last line was comment  bool last\_line\_is\_comment (void) const;  //whether last line was compiler directive  bool last\_line\_is\_compiler\_directive (void) const;  //whether we are in a block comment  bool is\_in\_block\_comment (void) const;  //whether last line was part of a begin/end pair  bool last\_line\_is\_begin\_or\_end (void) const;  //whether the last line was countable  bool last\_line\_is\_countable (void) const;  //whether the last line was empty  bool last\_line\_is\_empty (void) const;  //updates the block comment count  void update\_block\_comment\_count (void);  //whether the last line starts with the given string  bool last\_line\_starts\_with (const std::string & search\_string) const;  //whether a given string starts with a search string  static bool string\_starts\_with (const std::string & given\_string,  const std::string & search\_string);  //returns the input string stripped of leading/trailing whitespace    std::string string\_stripped\_of\_whitespace (const std::string & input\_string) const;  //returns the transformed line (here, the line stripped of whitespace)  virtual std::string transformed\_line (const std::  string & input\_string) const;  //constructor  simple\_loc\_counter (void);  //destructor  virtual ~ simple\_loc\_counter (void);  //resets the object  void reset (void);  //writes the countable lines to the given output stream  void write\_countable\_lines (ostream & ostr) const;  protected:  //the buffered countable lines  std::vector < std::string > m\_countable\_lines;  //the "block comment" nesting level  int m\_block\_comment\_nesting\_level;  //the beginning of a block comment  static const std::string & block\_comment\_begin;  //the end of a block comment  static const std::string & block\_comment\_end;  //the beginning of an inline comment  static const std::string & inline\_comment\_begin;  //the beginning of a compiler directive  static const std::string & compiler\_directive\_begin;  //the "begin block" string  static const std::string & block\_begin;  //the "end block" string  static const std::string & block\_end;  //whitespace characters  static const std::string & whitespace\_characters;  };  #endif  /\*  \*/ |

**simple\_loc\_counter.cpp**

|  |
| --- |
| /\*  \*/  #include "simple\_loc\_counter.h"  #ifndef YAK\_MIN\_MAX\_H  #include "yak\_min\_max.h"  #endif  void  simple\_loc\_counter::parse\_last\_line (void)  {  if (last\_line\_is\_countable ())  {  m\_countable\_lines.push\_back (last\_line ());  }  }  int  simple\_loc\_counter::loc\_count (void) const  {  return m\_countable\_lines.size ();  }  bool  simple\_loc\_counter::last\_line\_is\_comment (void) const  {  bool Result = false;  if (last\_line\_starts\_with (block\_comment\_begin)  || last\_line\_starts\_with (inline\_comment\_begin)  || is\_in\_block\_comment ())  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_compiler\_directive (void) const  {  bool Result = false;  if (last\_line\_starts\_with (compiler\_directive\_begin))  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::is\_in\_block\_comment (void) const  {  bool Result = false;  if (m\_block\_comment\_nesting\_level > 0)  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_begin\_or\_end (void) const  {  bool Result = false;  if (last\_line\_starts\_with (block\_begin)  || last\_line\_starts\_with (block\_end))  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_empty (void) const  {  return (last\_line ().length () == 0);  }  bool  simple\_loc\_counter::last\_line\_is\_countable (void) const  {  bool Result = true;  if ((last\_line\_is\_comment ())  || (last\_line\_is\_begin\_or\_end ())  || (last\_line\_is\_compiler\_directive ()) || (last\_line\_is\_empty ()))  {  Result = false;  }  return Result;  }  void  simple\_loc\_counter::update\_block\_comment\_count (void)  {  //count through the string; add 1 to the block comment count if the begin  //string is encountered, subtract one if the end string is encountered.  for (unsigned int i = 0; i < last\_line ().length (); ++i)  {  std::string line\_remaining =  last\_line ().substr (i, last\_line ().length ());  if (string\_starts\_with (line\_remaining, block\_comment\_begin))  {  ++m\_block\_comment\_nesting\_level;  }  else if (string\_starts\_with (line\_remaining, block\_comment\_end))  {  --m\_block\_comment\_nesting\_level;  }  }  }  bool  simple\_loc\_counter::  last\_line\_starts\_with (const std::string & search\_string) const  {  return string\_starts\_with (last\_line (), search\_string);  }  bool  simple\_loc\_counter::string\_starts\_with (const std::string & given\_string,  const std::string & search\_string)  {  int  substring\_size =  yak\_min (given\_string.length (), search\_string.length ());  std::string substring = given\_string.substr (0, substring\_size);  bool Result = (substring == search\_string);  return Result;  }  std::string  simple\_loc\_counter::  string\_stripped\_of\_whitespace (const std::string & input\_string) const  {  std::string::size\_type start =  input\_string.find\_first\_not\_of (whitespace\_characters);  if (start == input\_string.npos)  {  start = 0;  }  std::string::size\_type end =  input\_string.find\_last\_not\_of (whitespace\_characters);  if (end == input\_string.npos)  {  end = 0;  }  std::string Result = input\_string.substr (start, (end == 0) ? 0 : end + 1);  return Result;  }  std::string  simple\_loc\_counter::transformed\_line (const std::string & input\_string) const  {  return string\_stripped\_of\_whitespace (input\_string);  }  simple\_loc\_counter::simple\_loc\_counter (void)  {  reset ();  }  simple\_loc\_counter::~simple\_loc\_counter (void)  {  }  void  simple\_loc\_counter::reset (void)  {  m\_countable\_lines.clear ();  m\_block\_comment\_nesting\_level = 0;  }  void  simple\_loc\_counter::write\_countable\_lines (ostream & ostr) const  {  for (std::vector < std::string >::const\_iterator iter =  m\_countable\_lines.begin (); iter != m\_countable\_lines.end (); ++iter)  {  ostr << \*iter << "\n";  }  }  const  std::string & simple\_loc\_counter::block\_comment\_begin = "/\*";  const  std::string & simple\_loc\_counter::block\_comment\_end = "\*/";  const  std::string & simple\_loc\_counter::inline\_comment\_begin = "//";  const  std::string & simple\_loc\_counter::compiler\_directive\_begin = "#";  const  std::string & simple\_loc\_counter::block\_begin = "{";  const  std::string & simple\_loc\_counter::block\_end = "}";  const  std::string & simple\_loc\_counter::whitespace\_characters = " \t\n\0x32";  /\*  \*/ |

**main.cpp**

|  |
| --- |
| /\*  \*/  #ifndef SIMPLE\_LOC\_COUNTER\_H  #include "simple\_loc\_counter.h"  #endif  istream \*  input\_stream\_from\_args (int arg\_count, const char \*\*arg\_vector)  {  istream \*Result = NULL;  if (arg\_count == 1)  {  Result = &cin;  }  else  {  const char \*help\_text = "PSP exercise 2A: Count the physical LOC from standard input\n according to the style and counting guidelines in reports \n 1A and 2A. \n \n Usage:\n \tpsp\_2a \n \n";  cout << help\_text;  }  return Result;  }  int  main (int arg\_count, const char \*\*arg\_vector)  {  //get the input stream, or print the help text as appropriate  istream \*input\_stream = input\_stream\_from\_args (arg\_count, arg\_vector);  if (input\_stream != NULL)  {  simple\_loc\_counter counter;  counter.set\_input\_stream (input\_stream);  counter.parse\_until\_eof ();  //output the loc  cout << "LOC: " << counter.loc\_count () << "\n";  }  }  /\*  \*/ |

**simple\_input\_parser.e**

|  |
| --- |
| deferred class SIMPLE\_INPUT\_PARSER  feature {ANY}    parse\_until\_eof is  --parses all input until an EOF is reached  require  input\_stream /= Void;  do  from  until  input\_stream.end\_of\_input  loop  read\_line;  if not input\_stream.end\_of\_input then  parse\_last\_line;  end;  end;  end -- parse\_until\_eof    set\_input(new\_input\_stream: INPUT\_STREAM) is  --sets the input stream  do  input\_stream := new\_input\_stream;  end -- set\_input    read\_line is  --reads a line from standard input  do  input\_stream.read\_line;  last\_line := transformed\_line(input\_stream.last\_string);  end -- read\_line    last\_line: STRING;    input\_stream: INPUT\_STREAM;    parse\_last\_line is    deferred  end -- parse\_last\_line    transformed\_line(to\_transform: STRING): STRING is  --transforms the line according to rules defined in subclasses  do  Result := to\_transform;  end -- transformed\_line  end -- class SIMPLE\_INPUT\_PARSER |

**simple\_loc\_counter.e**

|  |
| --- |
| class SIMPLE\_LOC\_COUNTER  -- counts one form of LOC in eiffel files  inherit  SIMPLE\_INPUT\_PARSER  redefine parse\_last\_line, transformed\_line  end;    creation {ANY}  make  feature {ANY}    make is  do  !!counted\_lines.make(1,0);  end -- make    parse\_last\_line is  -- store countable lines in an array  do  if last\_line\_is\_countable then  counted\_lines.add\_last(last\_line);  end;  end -- parse\_last\_line    counted\_lines: ARRAY[STRING];  --array containing countable lines    loc\_count: INTEGER is  -- number of lines counted as LOC  do  Result := counted\_lines.count;  end -- loc\_count    last\_line\_is\_comment: BOOLEAN is  do  if last\_line\_starts\_with(comment\_begin) then  Result := true;  else  Result := false;  end;  end -- last\_line\_is\_comment    comment\_begin: STRING is "--";    last\_line\_is\_compiler\_directive: BOOLEAN is false;    in\_block\_comment: BOOLEAN is false;    last\_line\_starts\_with(test\_string: STRING): BOOLEAN is  do  if last\_line.has\_prefix(test\_string) then  Result := true;  else  Result := false;  end;  end -- last\_line\_starts\_with    last\_line\_is\_countable: BOOLEAN is  do  if last\_line\_is\_comment or last\_line\_is\_begin\_or\_end or last\_line\_is\_empty then  Result := false;  else  Result := true;  end;  end -- last\_line\_is\_countable    last\_line\_is\_begin\_or\_end: BOOLEAN is  do  if last\_line\_starts\_with("do") or last\_line\_starts\_with("end") then  Result := true;  else  Result := false;  end;  end -- last\_line\_is\_begin\_or\_end    last\_line\_is\_empty: BOOLEAN is  do  if last\_line.empty then  Result := true;  else  Result := false;  end;  end -- last\_line\_is\_empty    transformed\_line(string: STRING): STRING is  do  Result := string\_stripped\_of\_whitespace(string);  end -- transformed\_line    string\_stripped\_of\_whitespace(string: STRING): STRING is  do  Result := string.twin;  Result.replace\_all('%T',' ');  Result.left\_adjust;  Result.right\_adjust;  end -- string\_stripped\_of\_whitespace    print\_counted\_lines(output: OUTPUT\_STREAM) is  local  index: INTEGER;  do  from  index := counted\_lines.lower;  until  not counted\_lines.valid\_index(index)  loop  output.put\_string(counted\_lines.item(index));  output.put\_string("%N");  index := index + 1;  end;  end -- print\_counted\_lines  end -- class SIMPLE\_LOC\_COUNTER |

**main.e**

|  |
| --- |
| class MAIN  creation {ANY}  make  feature {ANY}    make is  local  simple\_loc\_counter: SIMPLE\_LOC\_COUNTER;  do  !!simple\_loc\_counter.make;  simple\_loc\_counter.set\_input(io);  simple\_loc\_counter.parse\_until\_eof;  std\_output.put\_string("LOC: ");  std\_output.put\_integer(simple\_loc\_counter.loc\_count);  end -- make  end -- class MAIN |

**Prueba**

Se descubrieron muchos problemas aquí, y aunque algunos eran fáciles de solucionar (evidentemente la clase de cadena estándar de C ++, o al menos esta implementación, da resultados diferentes entre longitud y tamaño ...), algunos eran francamente complicados (si el estándar C ++ La clase de cadena no encuentra lo que busca en find\_first\_not\_of o find\_last\_not\_of, devuelve un valor misterioso, npos-- Confieso que esperaba la longitud de la cadena o algo así. Esto ocupó aproximadamente 21 minutos de la fase de prueba).

Mi decisión de permitir que la clase simple\_loc\_counter imprima líneas contadas hizo que las pruebas fueran mucho más agradables, ya que pude comparar más fácilmente lo que el programa estaba contando realmente con lo que había en los archivos enviados.

|  |  |
| --- | --- |
| Program Number | LOC |
| 1A | 94 |
| 2A | 233 |

**PSP0.1 Project Plan Summary**

**Table 2-10. Project Plan Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Adrian Jesus Lora Molina | Date: | 170621 |
| Program: | LOC counter | Program# | B1 |
| Instructor: | Juan Manuel González Calleros | Language: | C++ |

|  |  |  |  |
| --- | --- | --- | --- |
| Program Size | Plan | Actual | To date |
| Base |  | 0 |  |
| Deleted |  | 0 |  |
| Modified |  | 0 |  |
| Added |  | 233 |  |
| Reused |  | 0 | 0 |
| Total New and Changed | (no planned size) | 233 | 233 |
| Total LOC |  | 233 | 233 |
| Total new/reused |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time in Phase (min): | Plan | Actual | To Date | To Date% |
| Planning | 10 | 10 | 24 | 7 |
| Design | 30 | 27 | 44 | 13 |
| Code | 60 | 75 | 99 | 30 |
| Compile | 20 | 26 | 31 | 9 |
| Test | 45 | 59 | 110 | 34 |
| Postmortem | 10 | 17 | 20 | 6 |
| Total | 175 | 214 | 328 | 100 |
| *Defects Injected* |  | Actual | To Date | To Date % |
| Plan |  | 0 | 0 | 0 |
| Design |  | 11 | 11 | 26 |
| Code |  | 19 | 27 | 63 |
| Compile |  | 1 | 2 | 5 |
| Test |  | 0 | 3 | 7 |
| Total development |  | 31 | 43 | 100 |
| *Defects Removed* |  | Actual | To Date | To Date % |
| Planning |  | 0 | 0 | 0 |
| Design |  | 0 | 0 | 0 |
| Code |  | 11 | 11 | 26 |
| Compile |  | 11 | 18 | 42 |
| Test |  | 9 | 14 | 33 |
| Total development |  | 31 | 43 | 100 |
| After Development |  | 0 | 0 |  |

|  |
| --- |
| Eiffel code/compile/test |

|  |  |  |  |
| --- | --- | --- | --- |
| Time in Phase (min) | Actual | To Date | To Date % |
| Code | 31 | 46 | 53 |
| Compile | 18 | 28 | 32 |
| Test | 5 | 13 | 15 |
| Total | 54 | 87 | 100 |
| Defects Injected | Actual | To Date | To Date % |
| Design | 3 | 3 | 13 |
| Code | 14 | 20 | 87 |
| Compile | 0 | 0 | 0 |
| Test | 0 | 0 | 0 |
| Total | 17 | 23 | 100 |
| Defects Removed | Actual | To Date | To Date % |
| Code | 1 | 1 | 4 |
| Compile | 11 | 14 | 61 |
| Test | 5 | 8 | 35 |
| Total | 17 | 23 | 100 |

**Time Recording Log**

**Table 2-11. Time Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Adrian Jesus Lora Molina | Date: | 170621 |
| Instructor: | Juan Manuel González Calleros | Program# | B1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start | Stop | Interruption Time | Delta time | Phase | Comments |
| 170621 10:56:15 | 170621 11:25:13 | 1 | 27 | design |  |
| 170621 11:28:59 | 170621 12:48:48 | 4 | 75 | code |  |
| 170621 12:49:09 | 170621 13:15:23 | 0 | 26 | compile |  |
| 170621 13:15:45 | 170621 14:15:25 | 0 | 59 | test |  |
| 170621 14:58:59 | 170621 15:19:37 | 3 | 17 | postmortem |  |
|  |  |  |  |  |  |

**Defect Reporting Log**

**Table 2-13. Defect Recording Log - C++**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Adrian Jesus Lora Molina | Date: | 170621 |
| Instructor: | Juan Manuel González Calleros | Program# | B1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect found | Type | Reason | Phase Injected | Phase Removed | Fix time | Comments |
| 170621 11:54:19 | ic | om | design | code | 1 | Olvidé agregar el método setter para last\_line |
| 170621 11:58:06 | ic | om | design | code | 1 | Olvidé agregar el método de reinicio |
| 170621 12:08:01 | ic | om | design | code | 2 | Se agregó el método last\_line\_starts\_with |
| 170621 12:12:39 | ic | om | design | code | 1 | Se agregó la cadena compiler\_directive\_begin |
| 170621 12:15:30 | ic | om | design | code | 1 | Se agregaron cadenas de inicio / fin de bloque |
| 170621 12:18:56 | id | ty | design | code | 1 | renombrado métodos "is\_last\_line ..." a "last\_line\_is ..." |
| 170621 12:24:17 | ic | om | design | code | 2 | Tuve que agregar yak\_min\_max para obtener la funcionalidad mínima y máxima |
| 170621 12:26:54 | ic | om | code | code | 2 | Refactorizado last\_line\_starts\_with en un método string\_starts\_with más genérico |
| 170621 12:35:45 | ic | om | design | code | 0 | Se agregó una cadena de caracteres de espacio en blanco |
| 170621 12:38:17 | ic | om | design | code | 1 | Constructor agregado y método de reinicio |
| 170621 12:42:35 | ic | om | design | code | 1 | Se agregó el método write\_countable\_lines |
| 170621 12:49:50 | wn | ty | code | compile | 0 | Llamado "set\_input" en lugar de "set\_input\_stream" |
| 170621 12:52:26 | id | om | code | compile | 1 | Olvidé agregar yak\_defs, yak\_exception para incluir la ruta |
| 170621 12:56:19 | iu | ty | code | compile | 0 |  |
| 170621 12:58:06 | wn | ty | code | compile | 0 | Nombre mal escrito de last\_line\_is\_countable |
| 170621 12:58:55 | wn | ty | code | compile | 0 | No califico completamente std :: string :: size\_type |
| 170621 13:00:17 | sy | om | code | compile | 0 | Olvidé limpiar "virtual" de la declaración del método en el archivo .cpp |
| 170621 13:01:15 | sy | om | code | compile | 0 | Olvidé agregar "simple\_loc\_counter ::" a la declaración del método |
| 170621 13:02:52 | iu | om | code | compile | 0 | Calificador const mal utilizado en la declaración del método para write\_countable\_lines |
| 170621 13:04:02 | iu | om | code | compile | 0 | Necesario agregar calificador const en write\_countable\_lines |
| 170621 13:04:44 | sy | om | code | compile | 0 | Se olvidó de eliminar "static" de la declaración en cpp de las variables estáticas de la clase |
| 170621 13:06:30 | iu | ex | compile | compile | 7 | Tuve que manipular la configuración del archivo MAKE para reutilizar un código externo que no estaba destinado a esta reutilización |
| 170621 13:20:28 | wn | kn | code | test | 4 | Se usó "tamaño" en lugar de "longitud" para la longitud de la cadena |
| 170621 13:25:22 | is | om | code | test | 1 | Olvidé agregar destructores virtuales (advertido por el compilador) |
| 170621 13:27:43 | iu | om | code | test | 0 | Se usó "int" en lugar de "unsigned int" para comparar con size () |
| 170621 13:28:25 | wn | om | code | test | 0 | Múltiples usos de "tamaño" en lugar de "longitud" para la longitud de la cadena |
| 170621 13:30:41 | iu | kn | code | test | 21 | find \_ \* \_ not\_of devuelve un misterioso "npos" cuando la cadena solo contiene los caracteres de búsqueda. |
| 170621 13:54:32 | ic | om | design | test | 2 | ¡Olvidé buscar líneas vacías! |
| 170621 13:58:59 | iu | kn | code | test | 0 | Uno a uno; no incluía el carácter final en string\_stripped\_of\_whitespace |
| 170621 14:01:20 | iu | kn | code | test | 10 | Off-by-the-other-one; incluía líneas en blanco que constan de un espacio |
| 170621 14:12:07 | wa | kn | code | test | 1 | Solo estaba buscando "}" para cerrar bloques, descontando posibles "};" ... |
|  |  |  |  |  |  |  |