PSP0.1 Ejercicio 2A

1. Resumen del plan del proyecto nivel PSP0.1

PSP0.1 Project Plan Summary - Program 2A

Student	1-Rubén Ignacio Couoh Ku		Date	19/01/2017
Program	LOC Counter		Program#	2A
Instructor	Carlos Mojica		Language	Node v6.9.4 LTS

	Plan	Actual	To Date	To Date%
Program Size (LOC)				
Base(B)		0		
Deleted(D)		0		
Modified(M)		0		
Added(A)		76		
Reused(R)		0	0	
Total N&C (N)	80	76	76	
Total LOC(T)		76	76	
Total New Reused		0	0	
Time in Phase (min.)				
Planning	10	9	19	8.0
Design	10	4	5	2.0
Code	80	57	154	66.1
Compile	15	4	18	7.5
Test	10	14	23	10.0
Postmortem	10	11	15	6.3
Total	125	98	233	100.0
Defects Injected				
Planning		0	0	0.0
Design		0	0	0.0
Code		4	11	100.0
Compile		0	0	0.0
Test		0	0	0.0
Total Development		4	11	100.0
Defects Removed				
Planning		0	0	0.0
Design		0	0	0.0
Code		0	0	0.0
Compile		3	9	81.8
Test		1	2	18.2
Total Development		4	11	100.0
After Development		0	0	

2. Resumen del plan del proyecto nivel PSP0

PSP0 Project Plan Summary - Program 1A

Student	1-Rubén Ignacio Couoh Ku	Date	10/01/2017
Program	Correlation	Program#	1A
Instructor	Carlos Mojica	Language	Node v6.9.4 LTS

	Plan	Actual	To Date	To Date%
Time in Phase (min.)				
Planning		10	10	7.4
Design		1	1	0.7
Code		97	97	71.9
Compile		14	14	10.4
Test		9	9	6.7
Postmortem		4	4	3.0
Total	180.0	135	135	100.0
Defects Injected				
Planning		0	0	0.0
Design		0	0	0.0
Code		7	7	100.0
Compile		0	0	0.0
Test		0	0	0.0
Total Development		7	7	100.0
Defects Removed				
Planning		0	0	0.0
Design		0	0	0.0
Code		0	0	0.0
Compile		6	6	85.7
Test		1	1	14.3
Total Development		7	7	100.0
After Development		0	0	

3. Forma de registros PIPs.

Notas y Comentarios
Agregar una pequeña descripción y fórmulas para calcular para calcular las líneas de código en el script PSP0.1 postmortem.

4. Forma de registro de tiempos.

Project	Phase	Date	Start	Int.	Stop	Delta	Comments
2	PLAN	01/19/17	14:32:21		14:37:26	5.1	Revisión de los requerimientos del programa para el conteo de líneas de código y aclaración de dudas.
2	PLAN	01/19/17	14:38:53		14:42:29	3.6	Estimación y registro de tiempos en las formas de registro.
2	DLD	01/19/17	14:48:52		14:52:35	3.7	Identificación de las tareas: leer archivos del códifo fuente del programa, procesar el contenido de los archivos paracontar las LOC y mostrar en la pantalla el analisis de los resultados
2	CODE	01/19/17	15:11:01		15:32:55	21.9	Codificación de funcionalidad para leer los archivos del código fuente del programa.
2	CODE	01/19/17	15:57:43	1.0	16:10:20	11.6	Codificación de la funcionalidad para el procesamiento y conteo de las líneas de codigo. Interrupcion para resolver dudas de SVS--configuración de la interfaz de red
2	CODE	01/19/17	16:13:22		16:18:28	5.1	Continuación de la codificación de la funcionalidad para el procesamiento y conteno de las líneas de codigo.
2	CODE	01/19/17	16:27:28		16:35:10	7.7	Mostrar la solución del análisis en pantalla.
2	CODE	01/19/17	16:39:23		16:50:07	10.7	Integración de las funcionalidades leer archivos del código fuente del programa, procesar el contenido de los archivos del código fuente para contar las LOC y mostrar en la pantalla el analisis de los resultados.
2	COMPILE	01/19/17	16:58:57		17:00:31	1.6	Se corrigio la expresion regular que filtra los comentarios y líneas en blanco, falto escapar el carácter "/" que filtra los comentarios cortos "//".
2	COMPILE	01/19/17	17:04:06		17:04:36	0.5	Se corrigio un error en la creación de la interface readline, se introdujo un ";" dentro de las opciones de configuración
2	COMPILE	01/19/17	17:13:18		17:14:46	1.5	Se corrigio el error en el paso de los argumentos en la funcion printSummary(folder, summaries) no se paso el argumento folder y mala escritura de la palabra reservada cosole.
2	TEST	01/19/17	17:36:43		17:43:59	7.3	Se corrigio la expresion regular que filtra comentarios y lineas en blanco, no se indico en el patronel inicio "^" de línea y fin "$"de línea.
2	TEST	01/19/17	17:54:08		18:01:11	7.0	Ejecucion del programa alimentado con los dos programas PSP0-A1 y PSP0.1-A2, se adapto el código fuente del programa A1 a la plantilla de codificación.
2	PM	01/19/17	18:26:51		18:37:29	10.6	validación de tiempos y defectos en las formas de registros.

5. Forma de registro de defectos.

Project	Date	Num	Type	Injected	Removed	FixTime	Fix Ref.	Description
2	19/01/2017	8	40	CODE	COMPILE	1.6		Se corrigio la expresion regular que filtra los comentarios y líneas en blanco, falto escapar el carácter "/" que filtra los comentarios cortos "//".
2	19/01/2017	9	20	CODE	COMPILE	0.5		Se corrigio un error en la creación de la interface readline, se introdujo un ";" dentro de las opciones de configuración
2	19/01/2017	10	50	CODE	COMPILE	1.5		Se corrigio el error en el paso de los argumentos en la funcion printSummary(folder, summaries) no se paso el argumento folder y mala escritura de la palabra reservada cosole.
2	19/01/2017	11	40	CODE	TEST	7.3		Se corrigio la expresion regular que filtra comentarios y lineas en blanco, no se indico en la búsqueda el inicio "^" de línea y fin "$"de línea

6. Código fuente del programa.

```
/*************************************************************************/
/*  Name:        Rubén Couoh.                                            */
/*  Date:        19/01/2017                                              */
/*  Description: Programa utilizado para contar las lineas de código    */
/*************************************************************************/

const GLOB     = require('glob');
const FS       = require('fs');
const READLINE = require('readline');
/*************************************************/
/*  pattern encuentra todos los mentarios del tipo  */
/*      Comentario corto: //                        */
/*      Comentario largo: \/*... *\/                */
/*************************************************/

let pattern = /^\s*\/{2,}|^\s*\/\*.*\*\/\s*$|^\s*.{0}\s*$/;
/*****************************************************************************************/
/*  Reuse Instructions                                                                   */
/*      getFiles(folder, extensions, cb);                                                */
/*      Purpose:     Obtiene las lista de nombres de diferentes                          */
/*                   tipos archivos que se encuentran en una carpeta.                    */
/*      Limitations: NA                                                                  */
/*      Return:      Regresa un arreglo con los nombres de los archivos de la carpeta.  */
/*****************************************************************************************/
function getFiles(folder, extensions, cb)
{
    let files = '*.+({extensions})'.replace('{extensions}', extensions.join('|'));
    let pattern = '{folder}/{files}'
                    .replace('{folder}', folder)
                    .replace('{files}', files);

    GLOB(pattern, cb);
}
```

```
34
35  /*****************************************************************************************************************/
36  /*  Reuse Instructions                                                                                           */
37  /*      countLinesOfCode(file, cb)                                                                               */
38  /*      Purpose:     Cuenta las líneas de código del ${file}                                                     */
39  /*      Limitations: NA                                                                                          */
40  /*      Return:      Regresa un JSON {name: file, linesOfCode: linesOfCode} en la funcion callback.              */
41  /*****************************************************************************************************************/
42  function countLinesOfCode(file, cb)
43  {
44      let linesOfCode = 0;
45      let summary = {};
46      let rl = READLINE.createInterface({
47          input: FS.createReadStream(file, 'utf-8')
48      });
49
50      rl.on('line', (line) => {
51          //  Si no es comentario o línea en blanco la cuenta como línea de código.
52          if (!pattern.test(line)) {
53              linesOfCode++;
54          }
55
56      });
57
58      rl.on('close', () => {
59          summary = {name: file, linesOfCode: linesOfCode}
60          cb(null, summary);
61      });
62  }
63
```

```
64  /*****************************************************************************************/
65  /*  Reuse Instructions                                                                   */
66  /*      printSummary(folder, summaries);                                                 */
67  /*      Purpose:     Imprime en la consola un resúmen del total de líneas de código.     */
68  /*      Limitations: NA                                                                  */
69  /*      Return:      Regresa undefined.                                                  */
70  /*****************************************************************************************/
71  function printSummary(folder, summaries)
72  {
73      let linesOfCode = 0;
74      console.log();
75      console.log('*****************************************************');
76      console.log(`*\tPrograma: ${folder}`);
77      summaries.forEach((summary) => {
78          linesOfCode += summary.linesOfCode;
79          console.log(`*\t${summary.name}:\tLOC ${summary.linesOfCode}`);
80      });
81      console.log('*****************************************************');
82      console.log(`*\t\tTotal LOC:\t${linesOfCode}`);
83      console.log('*****************************************************');
84  }
85
```

```
85
86
87  (function main()
88  {
89      //  Carpeta donde se encuentra el programa
90      //let folder = './../../PSP0/PROG1';
91      let folder = './../../PSP0.1/PROG2';
92
93      // Tipos de archivos en los cuales se desean contar las líneas de código.
94      // Pueden existir archivos e configuración que no se desean contar.
95      let extensions = ['js'];
96
97      getFiles(folder, extensions, (err, files) => {
98          if (err) {
99              console.error(err);
100             return;
101         }
102         countLinesOfCodeByFile(files, processSumaries);
103
104     });
105
106     function countLinesOfCodeByFile(files, cb)
107     {
108         let remaining = files.length;
109         let summaries = [];
110
111         files.forEach(function (file)
112         {
113             countLinesOfCode(file, (err, summary) => {
114                 if (err) {
115                     console.log(err);
116                 }
117                 summaries.push(summary);
118                 if (--remaining === 0) {
119                     cb(summaries);
120                 }
121             });
122         });
123     }
124
125     function processSumaries(summaries)
126     {
127         printSummary(folder, summaries);
128     }
129 })();
```

7. Reporte R1

Plantilla de Estándar de Conteo de LOC

Nombre: Estándar de codificación para curso de PSP Lenguaje: NodeJS v6.9
Autor: Rubén Ignacio Couoh Ku Fecha: 18/01/2017

Tipo de conteo Físico/Lógico	**Tipo** Físico	**Comentarios**
Tipo de Sentencia	**Incluir**	**Comentarios**
Ejecutables	**Sí**	**Una por cada línea.**
No ejecutables:		
Declaraciones	**Sí**	**Una por cada declaración.**
Directivas del compilador	**Sí**	**Una por cada directiva.**
Comentarios		**Los comentarios y las líneas en blanco serán ignorados durante el conteo.**
En su propia línea	**No**	
Con código	**No**	
Líneas en blanco	**No**	
Aclaraciones		
Estructuras control de flujo (if, else, else if, switch, try, catch)	**Sí**	
Estructuras de iteración (for, while, do while)	**Sí**	
Saltos (continue, break, throw, return)	**Sí**	
Expresiones que finalizan con ";"	**Si**	
Delimitadores de bloques "{" y "}"	**Si**	
Declaraciones de datos (clases, métodos, variables)	**Sí**	
Directivas de compilación	**Si**	

8. Resumen R2.

Plantilla de Estándar de Codificación

Propósito	Guiar en el desarrollo de programas de software
Encabezado de programas	Comenzar todos los programas con un encabezado descriptivo.
Formato de encabezado	/********** /* Name: nombre de programador. */ /* Date: la fecha en la que se inició el desarrollo del programa. */ /* Description: una corta descripción del programa y lo que hace. */ **********
Instrucciones de reutilización	• Describir cómo es usado el programa. Proveer el formato de declaración, valores y tipos así como los límites de los parámetros. • Proveer advertencias de valores ilegales, condiciones de sobre flujo o cualquier otra condición que pudiera resultar en una operación impropia.
Ejemplo de reutilización	/********** /* Reuse instructions */ /* printLine(lineOfCharacter) */ /* Purpose: to print string, 'lineOfCharacter', on one print line */ /* Limitations: the line length must not exceed LINE_LENGTH */ /* Return 0 if printer not ready to print, else 1 */ **********
Identificadores	Usar nombres descriptivos para todas las variables, funciones, constantes y cualquier otro identificador. Evitar abreviaciones o el uso de una única letra.
Variables	• Se deberá usar el estilo de escritura "*lowerCamelCase*" para nombrar las variables. Ejemplos: • var lineOfCharacter = 'Hola mundo'; • var isEmpty = true; • var counter = 0;
Constantes	• Se deberá usar puras mayúsculas y separadas por "_" en caso de dos o más palabras para nombrar las constantes. Ejemplos: • const SIZE = 100; • const LINE_LENGTH = 1000;
Métodos /Funciones	• Se deberá usar verbos en infinitivo para el nombre de los métodos. • Se deberá usar el estilo de escritura "*lowerCamelCase*" Ejemplos: • run () {} • draw () {}

Clases	• Se deberá usar sustantivos para nombrar las clases. • Se deberá usar el estilo de escritura "UpperCamelCase" para nombrar las clases. Ejemplos: • class Math {} • class WoodenChair {}
Comentarios	• Documentar el código de tal manera que el lector pueda entender su operación. • Los comentarios deben explicar tanto el propósito y el comportamiento del código. • Comentar la declaración de variables para indicar su propósito. • Se deberá usar dos tipos de comentario corto y largo. Ejemplos: • Largo: /* Comentario largo … */ • Corto: // Comentario corto.
Ejemplos de buenos comentarios	// have all records been processed? if (recordCount > limit) {}
Ejemplo de malos comentarios	// check if record count exceeds limit if (recordCount > limit) {}
Secciones principales	Preceder las secciones por un bloque de comentarios que describa el tipo de procesamiento que hacen.
Ejemplo	/********** /* The program section examines the contents of the array 'grades' and calcu- */ /* lates the average class grade. */ **********
Espacios en blaco	• Escribir los programas con suficiente espacio en blanco para que sea legible. • Separa cada constructor de programa con al menos un espacio en blanco.
Identación	• Identar cada nivel o rama respecto a la anterior. • Cerrar y abrir los niveles o ramas en su propia línea alineándolas.

Ejemplo de identación	while (missDistance > threshold) {
	successCode = moveRobot (targetLocation);
	if (successCode === MOVE_FAILED) {
	Console.log('The robot move has failed.');
	}
	}

Uso de mayúsculas y minúsculas	• En los mensajes hacia los usuarios pueden usarse tanto minúsculas como mayúsculas para dejar en claro el contenido.

<table>
<tr><td>Declaraciones de variables</td><td>

Se deberá realizar una única declaración por línea e inicializarlas.
Se deberá declarar las variables al comienzo de los bloques.
Las variables locales se deberán declarar en el bloque donde se utilicen.

Ejemplos:
<pre>
let base = 100;
method()
{
  let count = 0;
  while(count < base) {
    console.log(base + count);
    count++;
  }
}
</pre>
</td></tr>
<tr><td>Funciones/Métodos</td><td>
<pre>
function nameFunction()
{
  ...
}
nameMethod ()
{
  ...
}
(param) => {
  ...
}
</pre>
</td></tr>
<tr><td>Declaraciones de clases</td><td>
<pre>
class List {
    constructor()
    {
        this.head = null;
        this.tail = null;
    }

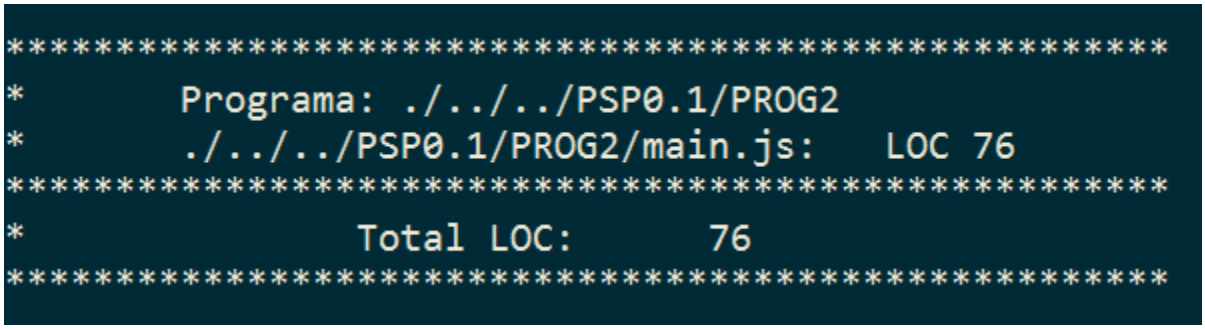
    get isEmpty()
    {
        return this.head === null;
    }
}

</pre>
</td></tr>
</table>

Declaraciones de estructuras de control de flujo.	let isEmpty = true;
	let count = 0;
	
	if (isEmpty) {
	…
	}
	if (isEmpty) {
	…
	} else {
	…
	}
	if (isEmpty) {
	…
	} else if (count > 10) {
	…
	} else {
	…
	}
	
	let char = 'r';
	switch (char) {
	case 'r': …
	break;
	default: …
	break;
	}
	
	try {
	…
	} catch (e) {
	…
	}
	
	try {
	…
	} catch (e) {
	…
	} finally {
	…
	}

Declaraciones de estructuras de iteración	for (let i=0; i<10; i++) {
	…
	}
	
	for (let object in objects) {
	…
	}
	
	for (let value of values) {
	…
	}
	
	while (value < 10) {
	…
	value++;
	}
	
	do {
	…
	} while(value < 10);

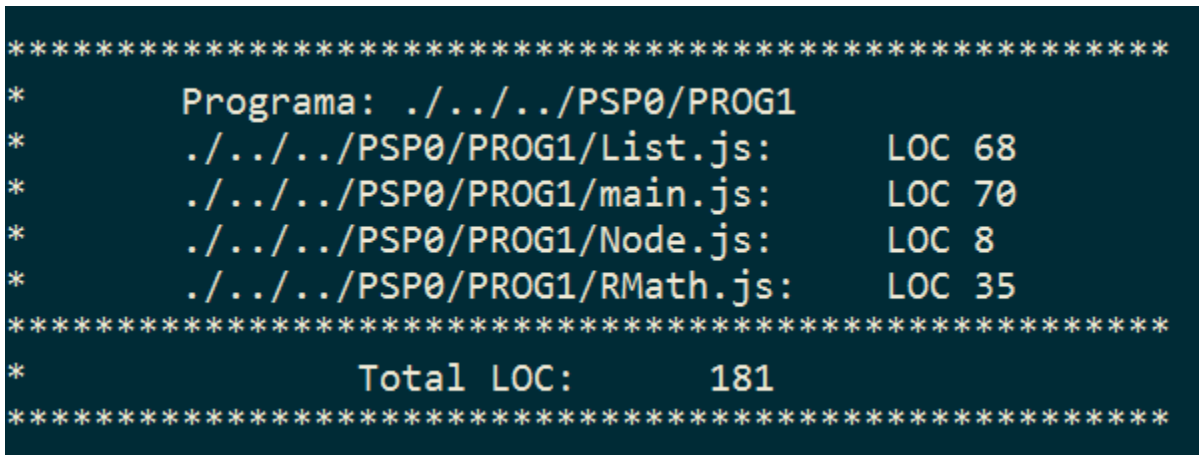
9. Interfaz de usuario.



10. Resultados.

Entrada: Programa 1A

Salida:



Entrada: Programa 2A

Salida:

```
*************************************************************
*       Programa: ./../../PSP0.1/PROG2
*       ./../../PSP0.1/PROG2/main.js:   LOC 76
*************************************************************
*               Total LOC:      76
*************************************************************
```

Resultados:

	Manual	Automático	Correcto
PROG1	181	181	SÍ
PROG2	76	76	SÍ

Tabla 10.1

Conclusiones:

Como se puede observar en la tabla 10.1 los resultados del programa de conteo son satisfactorios.