

# TPI - “Juego de la vida toroidal”

**Fecha de entrega:** 16 de noviembre de 2018  
**Devolución y coloquio:** 28 de noviembre de 2018  
**Recuperatorio:** 3 de diciembre de 2018

## 1. Antes de empezar

1. Bajar del campus de la materia Archivos TPI.
2. Descomprimir el ZIP.
3. Dentro del ZIP van a encontrar una carpeta llamada *juegoDeLaVida*, cargarla como proyecto en CLion.
4. El proyecto tiene varios *targets* que compilan cosas diferentes:
  - *juegoDeLaVidaTest*: para ejecutar los casos de test (ejercicios 1, 3 y 5).
  - *juegoDeLaVida*: interfaz gráfica (ejercicios 2, 3 y 4).

## 2. Ejercicios

1. Escribir tests para la función `soloBloques`. Ver sección **Testing**.
2. Implementar las funciones descritas en la sección **Entrada/Salida** en el archivo *es.cpp*.
3. Implementar las funciones especificadas en la sección **Especificación** en el archivo *solucion.cpp*. Utilizar los tests provistos por la cátedra para verificar sus soluciones (los cuales **No pueden modificar**).
4. (*Opcional*) Utilizar la interfaz gráfica provista para probar las funciones de Entrada/Salida y visualizar la evolución del toroide. Ver sección **Interfaz gráfica**.
5. Completar (agregando) los tests necesarios para cubrir todas las líneas del archivo *solucion.cpp*. Utilizar la herramienta `lcov` para dicha tarea. Ver sección **Análisis de cobertura**.

## 3. Testing

Para el siguiente ejercicio se pide escribir tests para la siguiente especificación. Los tests deben ser útiles para encontrar errores en la implementación del algoritmo que la cátedra posee. **Opcional:** Implementar el algoritmo.

```
proc soloBloques (in t: toroide, out res: Bool) {  
  Pre {esToroide(t)}  
  Post {res = true  $\leftrightarrow$  ( $\forall f : \mathbb{Z}$ ) ( $\forall c : \mathbb{Z}$ ) ( $viva(f, c, t) \rightarrow (esSupIzqBloque(f, c, t) \vee esInfIzqBloque(f, c, t) \vee$   
     $esSupDerBloque(f, c, t) \vee esInfDerBloque(f, c, t))$ )}}  
  pred esSupIzqBloque (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {esBloqueDesdeBordeSupIzq(filaToroide(f - 1, t),  
    columnaToroide(c - 1, t), t)}  
  pred esInfIzqBloque (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {esBloqueDesdeBordeSupIzq(filaToroide(f - 2, t),  
    columnaToroide(c - 1, t), t)}  
  pred esSupDerBloque (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {esBloqueDesdeBordeSupIzq(filaToroide(f - 1, t),  
    columnaToroide(c - 2, t), t)}  
  pred esInfDerBloque (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {esBloqueDesdeBordeSupIzq(filaToroide(f - 2, t),  
    columnaToroide(c - 2, t), t)}  
  pred esBloqueDesdeBordeSupIzq (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {( $\forall i : \mathbb{Z}$ ) ( $\forall j : \mathbb{Z}$ ) ( $0 \leq i \leq 3 \wedge 0 \leq j \leq 3 \rightarrow$   
    ( $viva(filaToroide(f + i, t), columnaToroide(c + j, t)) \leftrightarrow (0 < i < 3 \wedge 0 < j < 3)$ ))}}  
  pred esToroide (t: toroide) {length(t) > 0  $\wedge_L$  length(t[0]) > 0  $\wedge$  ( $\forall f : \mathbb{Z}$ ) ( $0 \leq f < length(t) \rightarrow_L length(t[f]) =$   
    length(t[0]))}  
  aux filaToroide (f:  $\mathbb{Z}$ , t: toroide) :  $\mathbb{Z} = f \bmod filas(t)$ ;  
  aux columnaToroide (c:  $\mathbb{Z}$ , t: toroide) :  $\mathbb{Z} = c \bmod columnas(t)$ ;
```

```

pred viva (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {enRangoToroide(f, c, t)  $\wedge_L$  t[f][c] = true}
pred enRangoToroide (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {enRango(f, t)  $\wedge_L$  enRango(c, t[f])}
pred enRango (i:  $\mathbb{Z}$ , s: seq(T)) { $0 \leq i < \text{length}(s)$ }
aux filas (t: toroide) :  $\mathbb{Z} = \text{length}(t)$ ;
aux columnas (t: toroide) :  $\mathbb{Z} = \text{length}(t[0])$ ;
}

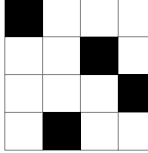
```

## 4. Entrada/Salida

Representaremos un toroide en un archivo con una secuencia de valores numéricos. Contando desde el principio del archivo, la interpretación de dichos valores es la que se detalla a continuación:

1. Cantidad de filas (**cantidadFilas**).
2. Cantidad de columnas (**cantidadColumnas**).
3. Lista de números que indican los valores de cada celda, ordenados primero por fila, y luego por columna. 1 indica que la celda está viva, 0 que está muerta. Debe haber exactamente **cantidadFilas**  $\times$  **cantidadColumnas** valores.
4. Entero positivo que se debe corresponder a la cantidad de celdas vivas que tiene el toroide.

Por ejemplo, el archivo de la izquierda representa un toroide de  $4 \times 4$  donde los únicos elementos vivos se corresponden a los indicados en el dibujo de la derecha:

4 4	
1 0 0 0	
0 0 1 0	
0 0 0 1	
0 1 0 0	
4	

Implementar las siguientes funciones:

1. `toroide cargarToroide(string nombreArchivo, bool &status)`

Que dado un nombre de archivo **nombreArchivo**, lea e interprete el contenido del mismo como un toroide. Debe modificar **status** para indicar si el procesamiento fue exitoso o no. Los motivos por los que la función debe devolver **status = false** (no exitoso) son:

- No se pudo abrir el archivo (inexistente o sin permisos).
- Los datos contenidos en el archivo no respetan el formato descrito anteriormente.

2. `bool escribirToroide(string nombreArchivo, toroide &t)`

Que dado un nombre de archivo **nombreArchivo** y un toroide **t**, almacene el toroide en el archivo indicado respetando el formato descrito anteriormente. El valor de retorno debe indicar si la operación se pudo realizar exitosamente o no. Los motivos por los que la función debe devolver **false** (no exitoso) son:

- No se pudo abrir el archivo para escritura (sin permisos, *path* inexistente, etc).
- No se pudo escribir todo el toroide (sin espacio en disco, etc).

## 5. Especificación

Implementar algoritmos que cumplan las siguientes especificaciones

```

proc esValido (in t: toroide, out result: Bool) {
    Pre {True}
    Post {result = true  $\leftrightarrow$  esToroide(t)}
}

```

```

proc posicionesVivas (in t: toroide, out vivas: seq( $\mathbb{Z} \times \mathbb{Z}$ )) {
    Pre {esToroide(t)}
}

```

```

Post {estanLasVivas( $t, vivas$ )  $\wedge$  noEstanLasMuertas( $t, vivas$ )  $\wedge$  noHayOtras( $t, vivas$ )  $\wedge$  sinRepetidos( $vivas$ )}
pred estanLasVivas ( $t$ : toroide,  $vivas$ :  $seq(\mathbb{Z} \times \mathbb{Z})$ ) { $(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(viva(f, c, t) \rightarrow (f, c) \in vivas)$ }
pred noEstanLasMuertas ( $t$ : toroide,  $vivas$ :  $seq(\mathbb{Z} \times \mathbb{Z})$ ) { $(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(muerta(f, c, t) \rightarrow (f, c) \notin vivas)$ }
pred noHayOtras ( $t$ : toroide,  $vivas$ :  $seq(\mathbb{Z} \times \mathbb{Z})$ ) { $(\forall f : \mathbb{Z})(\forall c : \mathbb{Z})(\neg enRangoToroide(f, c, t) \rightarrow (f, c) \notin vivas)$ }
}

proc densidadPoblacion (in  $t$ : toroide, out result :  $\mathbb{R}$ ) {
  Pre {esToroide( $t$ )}
  Post {result = cantidadVivas( $t$ )/superficieTotal( $t$ )}
}

proc evolucionDePosicion (in  $t$ : toroide, in posicion :  $\mathbb{Z} \times \mathbb{Z}$ , out result : Bool) {
  Pre {esToroide( $t$ )  $\wedge_L posValida(t, posicion)$ }
  Post {result = true  $\leftrightarrow$  debeVivir( $t, posicion_0, posicion_1$ )}
}

proc evolucionToroide (inout  $t$ : toroide) {
  Pre { $t = T_0 \wedge esToroide(T_0)$ }
  Post {esEvolucionToroide( $t, T_0$ )}
}

proc evolucionMultiple (in  $t$ : toroide, in  $k$ :  $\mathbb{Z}$ , out result: toroide) {
  Pre {esToroide( $t$ )  $\wedge k \geq 1$ }
  Post {esEvolucionNivelK(result,  $t, k$ )}
}

proc esPeriodico (in  $t$ : toroide, inout  $p$ :  $\mathbb{Z}$ , out result: Bool) {
  Pre {esToroide( $t$ )  $\wedge p = P_0$ }
  Post {(esSuPropiaEvolucion( $t$ )  $\rightarrow$  result = true  $\wedge$  tienePeriodoP( $t, p$ ))  $\wedge$ 
    ( $\neg esSuPropiaEvolucion(t) \rightarrow (result = false \wedge p = P_0)$ )}
  pred esSuPropiaEvolucion ( $t$ : toroide) {( $\exists k : \mathbb{Z})(k > 0 \wedge esEvolucionNivelK(t, t, k))$ }
  pred tienePeriodoP ( $t$ : toroide,  $p$ :  $\mathbb{Z}$ ) {esEvolucionNivelK( $t, t, p$ ) $\wedge(\forall q : \mathbb{Z})(1 \leq q < p \rightarrow \neg esEvolucionNivelK(t, t, q))$ }
}

proc primosLejanos (in  $t1$ : toroide, in  $t2$ : toroide, out primos: Bool) {
  Pre {esToroide( $t1$ )  $\wedge esToroide(t2)$ }
  Post {primos = true  $\leftrightarrow$  (mismaDimension( $t1, t2$ )  $\wedge (\exists k : \mathbb{Z})(esEvolucionNivelK(t2, t1, k))$ }
}

proc seleccionNatural (in  $ts$ :  $seq(toroide)$ , out res:  $\mathbb{Z}$ ) {
  Pre {todosToroides( $ts$ )  $\wedge length(ts) > 0$ }
  Post { $0 \leq res < length(ts) \wedge_L sobreviveATodos(res, ts)$ }
  pred todosToroides ( $ts$ :  $seq(toroide)$ ) {( $\forall t : toroide)(t \in ts \rightarrow esToroide(t))$ }
  pred sobreviveATodos ( $i$ :  $\mathbb{Z}$ ,  $ts$ :  $seq(toroide)$ ) { $\neg vaAMorir(ts[i]) \vee esElUltimoEnCaer(i, ts)$ }
  pred vaAMorir ( $t$ : toroide) {( $\exists tmuerto : toroide)(esEvolucionToroide(tmuerto, t) \wedge muerto(tmuerto))$ }
  pred muerto ( $t$ : toroide) {( $\forall i : \mathbb{Z})(\forall j : \mathbb{Z})(enRangoToroide(i, j, t) \rightarrow_L muerta(i, j, t))$ }
  pred esElUltimoEnCaer ( $i$ :  $\mathbb{Z}$ ,  $ts$ :  $seq(toroide)$ ) {( $\forall j : \mathbb{Z})(0 \leq j < length(t) \rightarrow_L muereAntes(t[j], t[i]))$ }
}

```

```

    pred muereAntes (t1: toroide, t2: toroide)  $\{(\exists x1 : \mathbb{Z})((\exists x2 : \mathbb{Z})(ticksHastaElFin(t1, x1) \wedge ticksHastaElFin(t2, x2) \wedge x1 \leq x2))\}$ 

    pred ticksHastaElFin (t: toroide, x:  $\mathbb{Z}$ )  $\{(\exists tomuer : toroide)(muerto(tomuer) \wedge esEvolucionNivelK(tomuer, t, x) \wedge (\forall k : \mathbb{Z})(0 \leq k < x \rightarrow \neg esEvolucionNivelK(tomuer, t, k)))\}$ 
}

proc fusionar (in t1: toroide, in t2: toroide, out res: toroide) {
    Pre  $\{esToroide(t1) \wedge esToroide(t2) \wedge mismaDimension(t1, t2)\}$ 
    Post  $\{esToroide(res) \wedge mismaDimension(res, t1) \wedge_L interseccionVivas(res, t1, t2)\}$ 
    pred interseccionVivas (t1: toroide, t2: toroide, t: toroide)  $\{(\forall f : \mathbb{Z})((\forall c : \mathbb{Z})(enRangoToroide(f, c, t) \rightarrow_L (viva(f, c, t) \leftrightarrow viva(f, c, t1) \wedge viva(f, c, t2))))\}$ 
}

proc vistaTrasladada (in t1: toroide, in t2: toroide, out res: Bool) {
    Pre  $\{esToroide(t1) \wedge esToroide(t2)\}$ 
    Post  $\{res = true \leftrightarrow mismaDimension(t1, t2) \wedge_L esTrasladada(t1, t2)\}$ 
}

proc enCrecimiento (in t: toroide, out res: Bool) {
    Pre  $\{esToroide(t)\}$ 
    Post  $\{res = true \leftrightarrow creceSuperficie(t)\}$ 
    pred creceSuperficie (t: toroide)  $\{(\exists tsig : toroide)(esEvolucionToroide(tsig, t) \wedge (\exists s1 : \mathbb{Z})((\exists s2 : \mathbb{Z})(esMenorSuperficieQueCubre(s1, t) \wedge esMenorSuperficieQueCubre(s2, tsig) \wedge s1 < s2)))\}$ 
    pred esMenorSuperficieQueCubre (sMin:  $\mathbb{Z}$ , t: toroide)  $\{(\forall sup : \mathbb{Z})(esSuperficieQueCubre(sup, t) \rightarrow sMin \leq sup)\}$ 
    pred esSuperficieQueCubre (sup:  $\mathbb{Z}$ , t: toroide)  $\{(\exists tcov : toroide)(esToroide(tcov) \wedge_L menorSuperficie(tcov, t) \wedge mismaCantidadVivas(tcov, t) \wedge esSubtoroideTrasladado(tcov, t) \wedge sup = superficieTotal(tcov))\}$ 
}

pred enRango (i:  $\mathbb{Z}$ , s:  $seq\langle T \rangle$ )  $\{0 \leq i < length(s)\}$ 
pred esToroide (t: toroide)  $\{length(t) > 0 \wedge_L length(t[0]) > 0 \wedge (\forall f : \mathbb{Z})(0 \leq f < length(t) \rightarrow_L length(t[f]) = length(t[0]))\}$ 
pred enRangoToroide (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide)  $\{enRango(f, t) \wedge_L enRango(c, t[f])\}$ 
pred viva (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide)  $\{enRangoToroide(f, c, t) \wedge_L t[f][c] = true\}$ 
pred muerta (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide)  $\{enRangoToroide(f, c, t) \wedge_L t[f][c] = false\}$ 
pred vivaToroide (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide)  $\{viva(filaToroide(f, t), columnaToroide(c, t), t)\}$ 
pred sinRepetidos (s:  $seq\langle T \rangle$ )  $\{(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})((enRango(i, s) \wedge enRango(j, s) \wedge i \neq j) \rightarrow_L s[i] \neq s[j])\}$ 
pred posValida (t: toroide, p:  $\mathbb{Z} \times \mathbb{Z}$ )  $\{enRangoToroide(p_0, p_1, t)\}$ 
pred debeVivir (t: toroide, f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ ) {
     $(viva(f, c, t) \rightarrow 2 \leq vecinosVivos(t, f, c) \leq 3) \wedge (muerta(f, c, t) \rightarrow vecinosVivos(t, f, c) = 3)$ 
}
aux vecinosVivos (t: toroide, f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ ) :  $\mathbb{Z} =$ 
 $\sum_{i=-1}^1 \sum_{j=-1}^1$  if  $(i \neq 0 \vee j \neq 0) \wedge vecinaViva(t, f, c, i, j)$  then 1 else 0 fi;
pred vecinaViva (t: toroide, f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , i:  $\mathbb{Z}$ , j:  $\mathbb{Z}$ )  $\{vivaToroide(f + i, c + j, t)\}$ 
pred esEvolucionToroide (tf: toroide, ti: toroide)  $\{mismaDimension(tf, ti) \wedge_L (\forall f : \mathbb{Z})((\forall c : \mathbb{Z})(enRangoToroide(f, c, ti) \rightarrow_L (viva(f, c, tf) \leftrightarrow debeVivir(ti, f, c))))\}$ 
pred esEvolucionNivelK (tf: toroide, ti: toroide, k:  $\mathbb{Z}$ )  $\{k \geq 0 \wedge_L (\exists s : seq\langle toroide \rangle)(length(s) = k + 1 \wedge_L s[0] = ti \wedge sonTicksConsecutivos(s) \wedge s[k] = tf)\}$ 
pred sonTicksConsecutivos (s:  $seq\langle toroide \rangle$ )  $\{(\forall i : \mathbb{Z})(0 \leq i < length(s) - 1 \rightarrow_L esEvolucionToroide(s[i + 1], s[i]))\}$ 

```

```

aux superficieTotal (t: toroide) :  $\mathbb{Z}$  = filas(t) * columnas(t);
aux filas (t: toroide) :  $\mathbb{Z}$  = length(t);
aux filaToroide (f:  $\mathbb{Z}$ , t: toroide) :  $\mathbb{Z}$  = f mod filas(t);
aux columnas (t: toroide) :  $\mathbb{Z}$  = length(t[0]);
aux columnaToroide (c:  $\mathbb{Z}$ , t: toroide) :  $\mathbb{Z}$  = c mod columnas(t);
pred mismaDimension (t1: toroide, t2: toroide) {length(t1) = length(t2)  $\wedge_L$  ( $\forall f : \mathbb{Z}$ )(enRango(f, t1)  $\rightarrow_L$  length(t1[f]) = length(t2[f]))}
aux cantVivas (t: toroide) :  $\mathbb{Z}$  =  $\sum_{f=0}^{filas(t)-1} \sum_{c=0}^{columnas(t)-1}$  if viva(f, c, t) then 1 else 0 fi;
pred menorSuperficie (st: toroide, t: toroide) {superficieTotal(st)  $\leq$  superficieTotal(t)}
pred mismaCantidadVivas (st: toroide, t: toroide) {cantVivas(st) = cantVivas(t)}
pred esSubtoroideTrasladado (st: toroide, t: toroide) {( $\exists tor : toroide$ )(esSubtoroide(st, tor)  $\wedge$  esTrasladada(t, tor))}
pred esTrasladada (t1: toroide, t2: toroide) {( $\exists i : \mathbb{Z}$ )( $\exists j : \mathbb{Z}$ )(traslacion(t1, t2, i, j))}
pred translacion (t1: toroide, t2: toroide, i:  $\mathbb{Z}$ , j:  $\mathbb{Z}$ ) {( $\forall f : \mathbb{Z}$ )( $\forall c : \mathbb{Z}$ )enRangoToroide(i, j, t1)  $\rightarrow_L$  (viva(f, c, t1)  $\leftrightarrow$  vivaToroide(f + i, c + j, t2))}
pred esSubtoroide (st: toroide, t: toroide) {( $\exists xi : \mathbb{Z}$ )( $\exists xf : \mathbb{Z}$ )( $\exists yi : \mathbb{Z}$ )( $\exists yf : \mathbb{Z}$ )(enRangoToroide(xi, yi, t)  $\wedge$  enRangoToroide(xf, yf, t)  $\wedge$  sonLasDimensionesDelToroide(xf - xi, yf - yi, st)  $\wedge_L$  mismosValores(t, st, xi, yi, xf, yf))}
pred sonLasDimensionesDelToroide (f:  $\mathbb{Z}$ , c:  $\mathbb{Z}$ , t: toroide) {filas(t) = f  $\wedge$  columnas(t) = c}
pred mismosValores (t: toroide, st: toroide, xi:  $\mathbb{Z}$ , yi:  $\mathbb{Z}$ , xf:  $\mathbb{Z}$ , yf:  $\mathbb{Z}$ ) {( $\forall i : \mathbb{Z}$ )( $\forall j : \mathbb{Z}$ )((xi  $\leq i \leq xf \wedge yi \leq j \leq yf$ )  $\rightarrow_L$  t[i][j] = st[i - xi][j - yi])}

```

## 6. Interfaz gráfica

Los archivos del TP incluyen el código de una interfaz gráfica que les va a permitir visualizar la evolución del toroide. El código de la misma se encuentra en el archivo *juegoDeLaVida.cpp*. Las máquinas de los laboratorios ya tienen todas las dependencias necesarias (en Linux), pero en caso de querer utilizar una máquina personal, se necesita tener instalada la librería SDL2 (<https://www.libsdl.org/download-2.0.php>), que cuenta con versiones para Linux, Windows y Mac. En caso de utilizar Debian o Ubuntu, instalar el paquete `libsdl2-dev` (`sudo apt-get install libsdl2-dev`).

La interfaz gráfica requiere para su correcto funcionamiento que se encuentren completas las siguientes funciones:

- cargarToroide (sección **Entrada/Salida**).
- escribirToroide (sección **Entrada/Salida**).
- densidadPoblacion (sección **Especificación**).
- evolucionToroide (sección **Especificación**).

Una vez compilado (target *juegoDeLaVida* en CLion), debe ejecutarse desde una terminal para poder pasarle los parámetros necesarios:

```
→ ./juegoDeLaVida toroide [--nuevo --size=N,M]
```

- toroide: nombre del archivo desde donde se cargará el toroide.
- --nuevo: indica que el archivo especificado por toroide será sobrescrito por uno nuevo. Para utilizar esta opción debe especificarse, además, el parámetro --size.
- --size=N,M: solo en caso de especificar --nuevo, indica que el nuevo toroide debe tener el tamaño N filas  $\times$  M columnas.

Al ejecutar la interfaz, se verá una ventana con la visualización del toroide en su estado inicial. Por medio del teclado se indican las acciones que harán cambiar el estado del toroide:

- E: evoluciona el toroide un paso.
- P: evoluciona el toroide pasos sucesivos (hasta que se lo detenga).

- S: detiene la evolución.
- Shift+S: guarda el estado actual del toroide en el archivo especificado como parámetro.
- L: vuelve a cargar el toroide del archivo. En caso tratarse de un toroide nuevo, se reseteará al toroide vacío.

Además del teclado, el click izquierdo permite modificar el valor de la celda apuntada por el mouse. El efecto logrado será que el nuevo valor sea el contrario del que tenía previamente. Esta operación puede realizarse incluso cuando el toroide se encuentra en evolución.

## 7. Análisis de cobertura

Para realizar el análisis de cobertura de código utilizaremos la herramienta Gcov, que es parte del compilador GCC. El target *juegoDeLaVidaTest* ya está configurado para generar información de cobertura de código en tiempo de compilación. Esta información estará en archivos con el mismo nombre que los códigos fuente del TP, pero con extensión \*.gcno. Al ejecutar los casos de test, se generarán en el mismo lugar que los \*.gcno otra serie de archivos con extensión \*.gcda. Una vez que tenemos ambos conjuntos de archivos, ejecutar el siguiente comando:

```
→ lcov --capture --directory carpetaDelTP --output-file salida/coverage.info
```

Se generará el archivo `coverage.info` dentro de la carpeta `salida`, que luego podremos convertir a HTML para su visualización con el siguiente comando:

```
→ genhtml salida/coverage.info --output-directory salida/cobertura
```

Finalmente, se generará un archivo `index.html` dentro de `salida/cobertura` con el reporte correspondiente. Utilizar cualquier navegador para verlo.

Para mayor información, visitar:

<https://medium.com/@naveen.maltesh/generating-code-coverage-report-using-gnu-gcov-lcov-ee54a4de3f11>.

## Términos y condiciones

El trabajo práctico se realiza de manera grupal con grupos de exactamente 2 personas. Para aprobar el trabajo se necesita:

- Que todos los ejercicios estén resueltos.
- Que las soluciones sean correctas.
- Que todos los tests provistos por la cátedra funcionen.
- Que las soluciones sean prolijas: evitar repetir implementaciones innecesariamente y usar adecuadamente funciones auxiliares.

## Pautas de Entrega

Se debe enviar un e-mail a la dirección `tpalgo1@gmail.com`. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[ALG01;TPI]` seguido inmediatamente del nombre del grupo.
- En el cuerpo del email deberán indicar: Nombre, apellido, libreta universitaria de cada integrante.
- Debe enviarse el archivo comprimido (.zip) que contenga los archivos *es.cpp* y *solucion.cpp* completos. Además, enviar la carpeta de tests.

**Importante:** se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.