El formato de los ficheros de entrada de los laberintos es el siguiente (se muestra `data_maze_1.txt`):

```
12 12
1 8 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 1 1 0 0 0 1
1 1 0 0 0 0 0 1 0 1 0 1
1 1 1 0 1 0 0 0 1 1 0 1
1 0 1 0 0 0 1 0 0 1 0 1
1 1 1 1 0 1 1 1 0 1 0 1
1 0 1 0 0 0 1 1 0 0 1 1
1 1 1 1 1 0 1 1 0 1 0 1
1 1 0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 1 1 0 0 0 1
1 1 1 0 1 1 0 0 1 1 0 9
1 1 1 1 1 1 1 1 1 1 1 1
```

Se distinguen diferentes números o cifras:

- Las dos primeras cifras "`12 12`" indican la dimensión del laberinto en filas y columnas, respectivamente.
- Los "**`1`**" (definido en el código como `WALL_ID`) indican un muro (*wall*) y los "`0`" (definido en el código como `PASS_ID`) indican zona de paso/pasillo (*pass*).
- El "**`8`**" (definido como `START_ID`) indica la entrada del laberinto.
- El "**`9`**" (definido `END_ID`) indica la salida del laberinto.

Por tanto, el **objetivo final** de la práctica es, dado un **fichero con un laberinto** en el formato descrito anteriormente donde se indica la **entrada (`START_ID`)** y la **salida (`END_ID`)**, encontrar el camino desde la entrada a la salida, y guardarlo de alguna forma que se pueda mostrar posteriormente (usando, por ejemplo, el código `PATH_ID` sobre la propia matriz del laberinto).

FASE I: COMPRENSIÓN DE LOS RECURSOS Y DESARROLLO DEL MÉTODO QUE COMPRUEBA SI UNA CELDA A USAR ES VÁLIDA

La tarea de esta primera fase consiste en leer y entender la clase `maze_t` (desglosada en los ficheros `.hpp` y `.cpp`) y el fichero principal `main_maze_t.cpp`. Los detalles de implementación de estos códigos se explicarán en el vídeo y el foro de la semana tutorizada.

Para compilar los códigos debemos teclear:

```
$ g++ main_maze_t.cpp maze_t.cpp -o main_maze_t
```

En esta fase se debe desarrollar el método `maze_t::is_ok_(const int i, const int j)` que comprueba si una celda que está situada en la fila `i` y la columna `j` es adecuada para pasar a ella.

```
bool
maze_t::is_ok_(const int i, const int j) const
{
  // retornar true si se cumplen TODAS estas condiciones:
  // - fila i y la columna j están dentro de los límites del laberinto,
  // - la celda en (i, j) no puede ser un muro,
  // - la celda (i, j) no puede haber sido visitada antes.
}
```

FASE II: DESARROLLAR EL MÉTODO RECURSIVO QUE RESUELVE EL LABERINTO

En esta fase, se pide desarrollar el caso base y el caso general del método `maze_t::solve_(const int, const int)` que aparecen indicados en el propio código:

```
bool
maze_t::solve_(const int i, const int j)
{
  // CASO BASE:
  // retornar 'true' si 'i' y 'j' han llegado a la salida

  // [poner código aquí]

  // marcamos la celda como visitada
  visited_(i, j) = true;

  // CASO GENERAL:
  // para cada una de las 4 posibles direcciones (N, E, S, W) ver si es
posible
  // el desplazamiento (is_ok_) y, en ese caso, intentar resolver el
laberinto
  // llamando recursivamente a 'solve'.
  // Si la llamada devuelve 'true', poner en la celda el valor PATH_ID, y
  // propagarla retornando también 'true'

  // [poner código aquí]

  // desmarcamos la celda como visitada (denominado "backtracking") y
  // retornamos 'false'
  visited_(i, j) = false;
  return false;
}
```

Esta fase se puede probar ejecutando los diferentes ejemplos de laberintos de la siguiente forma:

```
$ ./main_maze_t < data_maze_1.txt
$ ./main_maze_t < data_maze_2.txt
$ ./main_maze_t < data_maze_3.txt
$ ./main_maze_t < data_maze_fail.txt
```

FASE III: ALMACENAR LA SOLUCIÓN GENERADA EN UNA ESTRUCTURA DE DATOS

Para superar esta fase, se debe desarrollar una forma de guardar la solución encontrada para poder ser mostrada posteriormente sobre el laberinto. Se puede hacer uso de cualquier clase desarrollada en las prácticas anteriores, pero se valorará la elección de la **estructura de datos más eficiente** (en términos de espacio) para guardar el camino de salida.

Evaluación

La calificación de la práctica dependerá de la calidad y eficiencia de la misma.

- Si la práctica funciona correctamente, así como la modificación propuesta por el profesor, y ha concluido la Fase I: **hasta 5**
- Si la práctica funciona correctamente, así como la modificación propuesta por el profesor, y ha concluido la Fase II: **hasta 8**
- Si además de lo anterior, se ha efectuado la Fase III: **hasta 10**

Una vez presentada la modificación de la práctica (y no antes) deben ser enviados solo los ficheros cpp y hpp a través del enlace de esta tarea al campus virtual.

Resultado esperado

Si ejecutan la práctica con todas las fases desarrolladas, la salida esperada por pantalla sería la siguiente: