Tecnológico de Costa Rica
Escuela de Ingeniería en Computación

Sistemas Operativos
Profesora: Erika Marín Schumman

1ER PROYECTO: HILOS

Estudiantes:

Sophya Mc Lean Morales - 2021461577

Jose Pablo Burgos Retana - 2021152011

Oriental, Cartago

Fecha de entrega: Domingo 07 de abril 2024

Tabla de contenidos

A. Introducción

El presente proyecto consiste en la simulación de un laberinto en el lenguaje C. El programa está diseñado para emular la exploración de un laberinto mediante la creación de hilos que se desplazan de campo en campo. Este proyecto tiene como objetivo principal implementar un algoritmo que permita a los hilos navegar por el laberinto, generando nuevos hilos para explorar diferentes caminos cuando se encuentren con bifurcaciones.

La lectura del laberinto se realiza a partir de un archivo de texto que contiene las dimensiones del laberinto y la ubicación de las paredes. Cada hilo creado informará la cantidad de espacios recorridos en su ruta y si logró alcanzar la salida del laberinto. Además, se mostrará en pantalla, mediante caracteres, el movimiento de los hilos y los espacios visitados.

Este proyecto de programación en C para Linux combina la lógica de navegación en laberintos con la concurrencia de hilos, ofreciendo una experiencia interactiva y desafiante tanto para el desarrollo como para la visualización de la exploración del laberinto.

b. Estrategia de Solución

Para el desarrollo de este proyecto, se optó por desarrollar un programa en C para implementar la lógica del laberinto: Lectura, asignación, resolución y acceso.

Para poder programar el código de manera que fuera funcional para linux, como lo pide la especificación, se utilizó la máquina virtual de Ubuntu y en ella se descargó el programa Visual Studio Code, lo cual permitió ir programando el código y probándolo en Linux. Cabe destacar que en un inicio esta no fue la solución escogida, los desarrolladores estuvieron programando su código en Visual Studio Code en Windows, sin embargo se dieron cuenta de que parte de la sintaxis e importaciones funcionaban distinto en Windows y Linux.

En cuanto al laberinto como tal, se definió un array bidimensional como estructura de datos para guardar la información del Laberinto. Se creó un “type struct” llamado “Celda.h” en el cual se definieron las características o atributos que tiene cada celda del laberinto:

- Caracter: Determina al char que tiene ese espacio del laberinto (‘1’ si es una celda disponible, ‘*’ si es una pared del laberinto, ‘/’ si es una salida y ‘R’ si es una celda ya recorrida)
- Izquierda, derecha, arriba, abajo: Estos son 4 campos distintos que tiene la celda, los cuales permiten saber si la celda ha sido recorrida en alguna de esas direcciones. Inicialmente estos campos están en 0.

Se decidió manejar este tipo de estructura, pues a la hora de ir haciendo las validaciones con los hilos que recorren al laberinto, se puede saber si un campo ya fue recorrido en esa dirección o no, de manera que oriente al hilo y se evite la duplicidad de recorridos.

Para leer el archivo proporcionado por el usuario,el cual contiene las especificaciones del laberinto, se decidió utilizar una función propia de C. Lo primero que se hace es abrir el archivo y leer la primera fila, la cual debe contener las dimensiones que tendrá el laberinto, por ejemplo: 10x10, 5x5, 100x100. Una vez que las dimensiones fueron leídas, estas se asignan a las variables respectivas para guardar esta información y se crea el arreglo bidimensional correspondiente al laberinto.

Se decidió crear el laberinto hasta que se leen las dimensiones porque de lo contrario el laberinto tendría que utilizarse de manera dinámica, lo que puede complicar las cosas en temas de memoria.

Posterior a esto, se recorre cada línea del laberinto, por cada línea se crea una nueva lista (array) y por cada caracter de la línea, se crea una struct “Celda” (Cuando encuentra un char de espacio en blanco, lo guarda en la celda como un ‘1’, para representar que es un espacio disponible).

Llegado al último caracter de la línea, la lista se agrega al laberinto. De manera que al finalizar de leer todo el archivo, se tiene un arreglo bidimensional compuesto por listas (cada una representa una línea del archivo) y cada lista guarda una serie de “objetos” celda.

La creación de los hilos para recorrer el laberinto se hace a través de una serie de validaciones. El hilo se compone de dos atributos: Dirección y cantidad de celdas recorridas. Inicialmente el hilo comienza en la celda (0,0) y va moviéndose en la dirección que indique el hilo, cuando se encuentra con una pared(*) o se terminan las dimensiones del hilo, este crea otro con una dirección diferente para que continúe recorriendo.

c. Análisis de Resultados:

Deberá elaborar un listado de todas y cada una de las actividades y tareas que deben cubrirse a nivel funcional, para cada una de ellas debe aportar el porcentaje de realización y en caso de no ser el 100% debe justificarse.

Tarea	Porcentaje completado	Justificación
El hilo guarda la dirección y conteo de campos visitados		
Se utiliza un semáforo para controlar el acceso al laberinto	50%	Existe la lógica, sin embargo no se pudo implementar con el hilo.
El laberinto se lee desde un archivo .txt y se guarda en memoria	100%	
Se imprime en consola el laberinto actualizado	100%	
Se determina correctamente cuando un hilo ha finalizado	80%	Existe un pequeño error, pues hay casos en los que termina la ejecución sin razón, evitando generar más hilos.
Se generan correctamente nuevos hilos cuando se cumplen las condiciones	80%	Existe un pequeño error, pues hay casos en los que termina la ejecución sin razón, evitando generar más hilos.
Cada campo del laberinto guarda las direcciones en las que fue recorrido	100%	

d. Lecciones aprendidas:

Debe prepararse un listado de las lecciones aprendidas producto del desarrollo de la tarea programada. Las lecciones aprendidas pueden ser de carácter personal y/o técnico que involucre aspectos que han logrado un aprendizaje en temas de investigación, desarrollo de habilidades técnicas y habilidades blandas como trabajo en equipo, comunicación, forma de expresar ideas, entre otros.

1. Es importante el manejo y buena distribución del tiempo para lograr la tarea asignada sin apuros.
2. El uso de semáforos evita conflictos al acceder a recursos compartidos.
3. La comunicación entre los miembros del equipo es de gran valor, pues permite a los estudiantes una resolución de conflictos adecuada y oportuna.
4. La buena comunicación entre los integrantes del equipo permite que el desarrollo del proyecto fluya.
5. Es importante abarcar dudas con los compañeros de equipo para que todos estén en la misma página.

e. Casos de pruebas:

La importancia de los casos de prueba en el desarrollo de software radica en la mejora de la calidad del producto, la detección y corrección de errores a tiempo, la reducción de los costos de desarrollo y la mejora del rendimiento.

Las pruebas de software son esenciales para asegurar que el producto funcione como se espera, cumpla con los requisitos establecidos y ofrezca una buena experiencia al usuario.

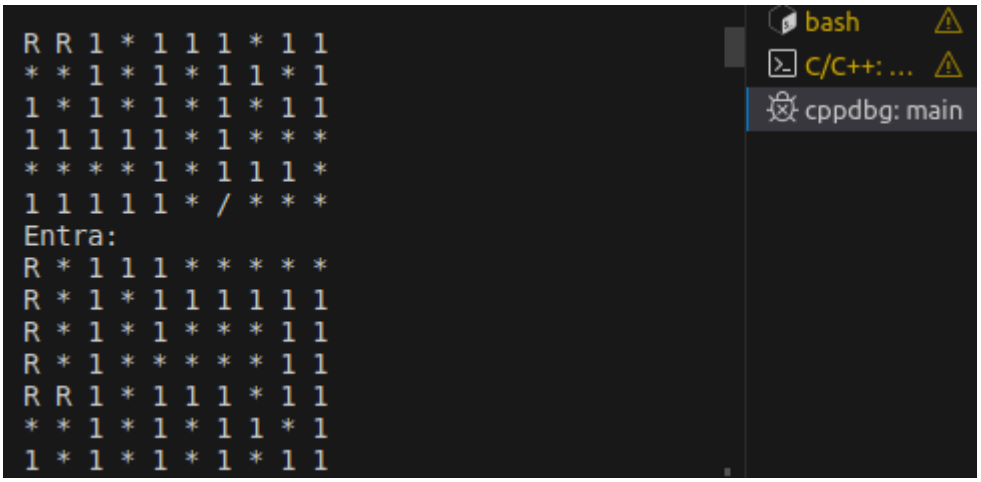
Los casos de prueba son una parte fundamental de las pruebas de software, ya que ayudan a definir los escenarios y situaciones que se deben evaluar.

En este caso se realizaron pruebas con dos laberintos diferentes, ambos proporcionados por la docente.

Laberinto 1 (lab1.txt):

```
10 10
 *   *****
 * *
 * * ***
 * *****
   *   *
** * *  *
 * * * *
     * ***
**** *   *
     */***
```

Resultado:



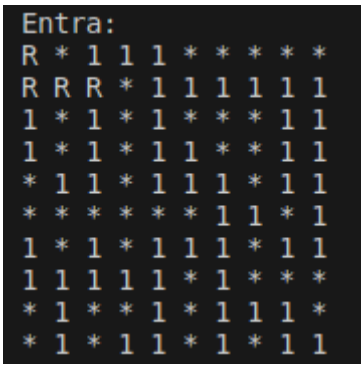
```
R * R R R * * * * *
R * R * R R R R R R
R * R * 1 * * * R R
R * R * * * * * R R
R R R * 1 1 1 * R R
* * 1 * 1 * 1 1 * R
1 * 1 * 1 * 1 * R R
1 1 1 1 1 * 1 * * *
* * * * 1 * 1 1 1 *
1 1 1 1 1 * / * * *
HILO TERMINADO
Cantidad de Celdas Recorridas: 27
```

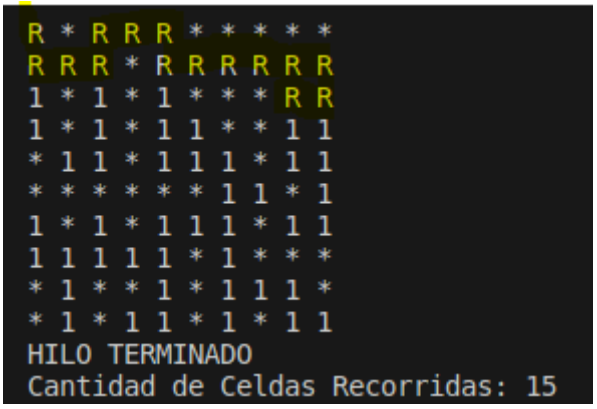
Laberinto 2 (lab2.txt):

```
11 11
 *   ******
   *      *
 * * ***   *
 * *  **   *
*  *   *   *
******  * * *
 * *   *   *
     * ****
* ** *   **
* *  * *  /
*/*********
```

Resultado:

```
Entra:
R * 1 1 1 * * * * *
1 1 1 * 1 1 1 1 1 1
1 * 1 * 1 * * * 1 1
1 * 1 * 1 1 * * 1 1
* 1 1 * 1 1 1 * 1 1
* * * * * * 1 1 * 1
1 * 1 * 1 1 1 * 1 1
1 1 1 1 1 * 1 * * *
* 1 * * 1 * 1 1 1 *
* 1 * 1 1 * 1 * 1 1
```





f. Comparación:

Los procesos y los hilos son conceptos fundamentales en la programación paralela. La principal diferencia entre ellos radica en cómo se crean, cómo comparten recursos y cómo interactúan entre sí.

Los procesos son entidades independientes creadas por el sistema operativo a petición de otros procesos. Cada proceso tiene su propio espacio de direccionamiento, lo que significa que no comparten memoria directamente. Cuando un proceso crea otro proceso, se denomina bifurcación (fork). La bifurcación es un mecanismo costoso en términos de recursos del sistema operativo, ya que implica la creación de un nuevo espacio de direccionamiento y la duplicación de todos los recursos asociados al proceso padre.

Los hilos, por otro lado, son entidades de ejecución ligeras que comparten el mismo espacio de direccionamiento con el proceso al que pertenecen. Esto significa que los hilos pueden acceder directamente a las mismas variables y estructuras de datos que el proceso que los creó. La creación y el cambio de contexto de los hilos son mucho más eficientes que los de los procesos, ya que no requieren la duplicación de recursos del sistema operativo.

La diferencia más significativa entre los procesos y los hilos, es que los primeros son típicamente independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema.

En el caso específico de este proyecto, si se utilizaran forks en vez de hilos para la resolución del laberinto, se habrían dado algunas condiciones:

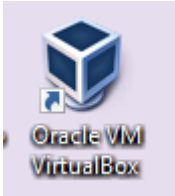
- ➔ Consumo de recursos: Los procesos son más pesados que los hilos, ya que cada proceso tiene su propio espacio de memoria y recursos del sistema operativo, mientras que los hilos comparten el mismo espacio de memoria y recursos del proceso padre. Por lo tanto, crear procesos hijos consume más recursos del sistema operativo y de la memoria que crear hilos.
- ➔ Comunicación entre procesos: La comunicación entre procesos es más compleja que la comunicación entre hilos. Los procesos utilizan mecanismos de comunicación interprocesos (IPC) como tuberías, sockets o memoria compartida, mientras que los hilos utilizan variables compartidas o sincronización de hilos.
- ➔ Sincronización: La sincronización entre procesos es más difícil que la sincronización entre hilos. Los procesos utilizan mecanismos de sincronización como semáforos o mutexes, mientras que los hilos utilizan mecanismos de sincronización de hilos como barreras o monitores.
- ➔ Contexto de ejecución: Cuando un proceso crea un proceso hijo, el proceso hijo se ejecuta en un contexto diferente al del proceso padre. Por lo tanto, si el proceso hijo modifica algún dato, el proceso padre no podrá ver los cambios hasta que se realice una operación de sincronización. En cambio, los hilos se ejecutan en el mismo contexto que el proceso padre, por lo que pueden compartir y modificar datos sin necesidad de sincronización.

Entonces podríamos decir que tal vez no sea tan buena idea implementar este proyecto con forks, si bien es posible hacerlo, se tendría un aumento en el uso de recursos y la comunicación sería más compleja. Lo que podría afectar el rendimiento del sistema operativo.

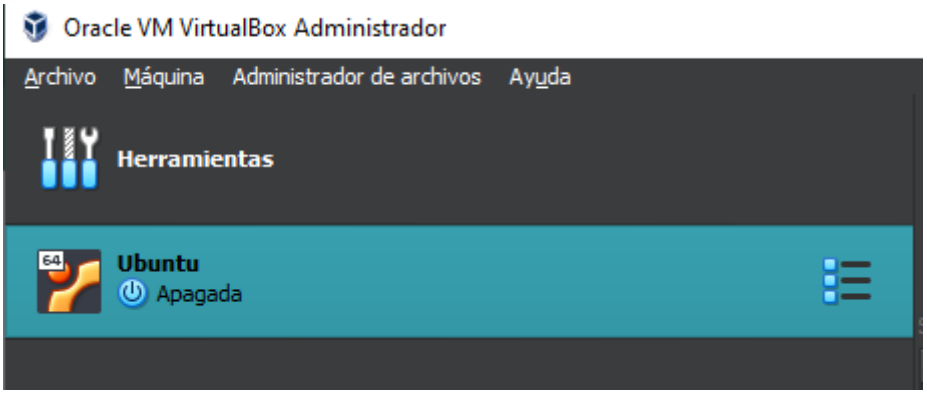
g. Manual de usuario:

Especificar como compilar y correr su tarea.

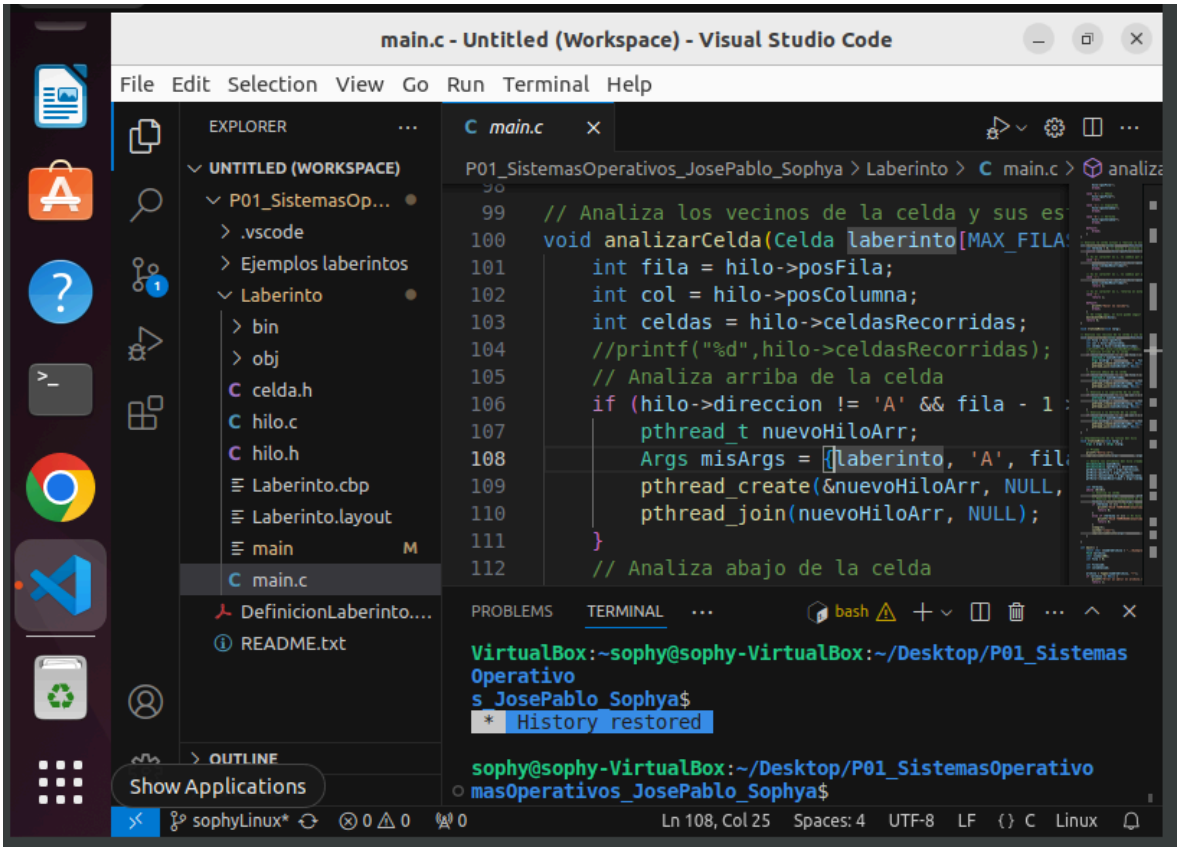
El proyecto se realizó en el lenguaje de programación de “C”. Para poder ejecutar el archivo es necesario:
Contar una computadora con el sistema operativo Linux o en su defecto una máquina virtual que simule este tipo de sistema operativo.
En este caso, se utilizó la “Oracle VM VirtualBox” para poder simular un sistema Linux con la distribución Ubuntu.



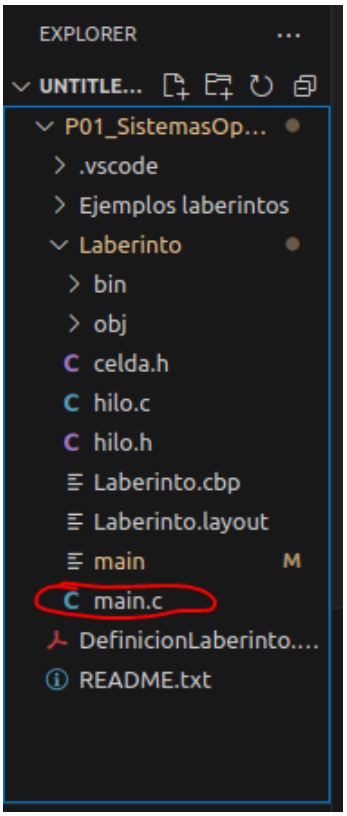
Luego debe accederse a la VirtualBox de Ubuntu



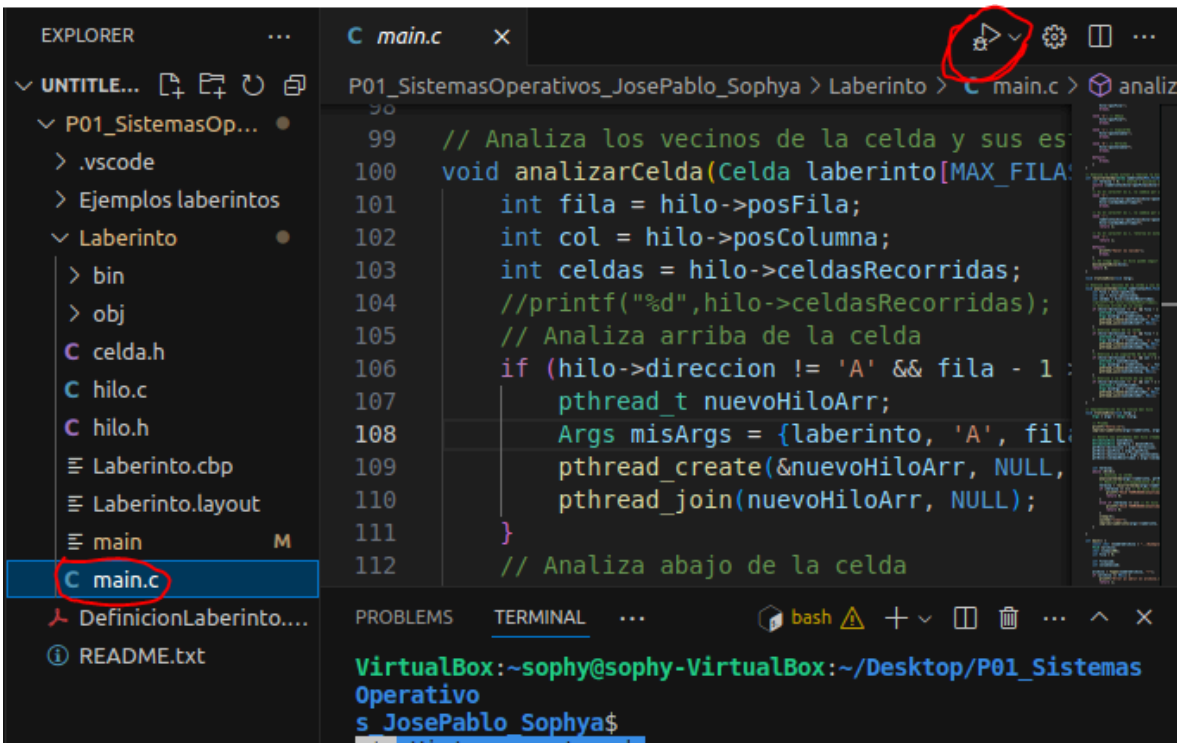
Dentro de la máquina virtual se tiene que tener instalado Visual Studio Code, junto con las extensiones pertinentes para compilar en lenguaje C.



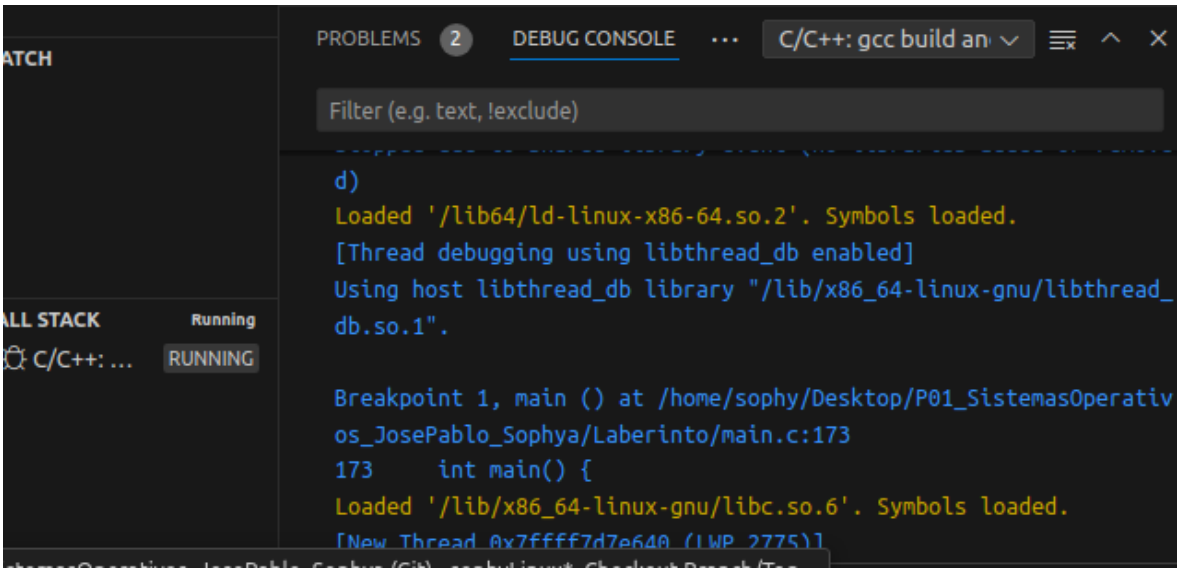
Luego, debe seleccionarse el archivo “main.c” de la lista de archivos del menú lateral izquierdo:



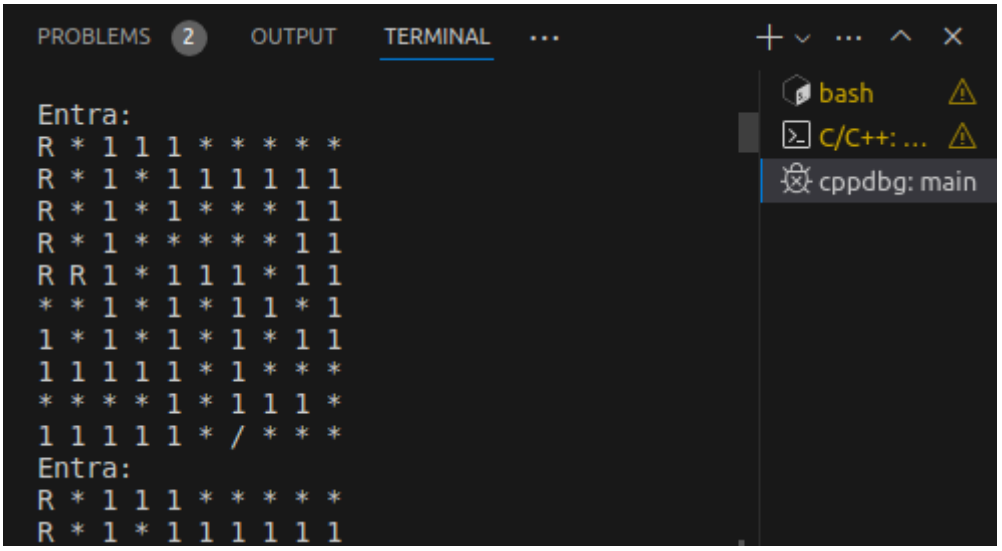
Una vez que se ha seleccionado el archivo, se desplegará una ventana con el código y un botón de un triángulo y un escarabajo, al cual deberá dar click para empezar la compilación y correr el código.



Una vez dado click, iniciará el proceso de “debuggin” del programa, para el cual se abrirá una terminal automáticamente y se mostrarán mensajes como los siguientes:



Desde la vista de la “terminal” se podrán ir observando los recorridos que hace el hilo hasta recorrer el laberinto:



h. Bitácora de trabajo durante las tres semanas de trabajo

Actividad	Participantes	Fecha
Asignación del proyecto por parte de la docente	Todos	Viernes 15 de marzo
Reunión para leer especificación	Todos	Sábado 16 de marzo
Reunión para distribuir tareas	Todos	Sábado 23 de marzo
Reunión para ver avances	Todos	
Reunión de validación de resultados	Todos	Domingo 07 de marzo

i. Bibliografía y fuentes digitales utilizadas

[Anónimo]. (s.f). LECCIÓN 7: CREACION DE PROCESOS. FORK [En línea]. Recuperado de https://www.dis.ulpgc.es/sopa/lecciones/leccion7.pdf