MINIGUÍA WSL y DEPURACIÓN CON VSCODE

El entorno de trabajo para las prácticas de EDAT es Linux. Esto significa que puedes usar las herramientas que quieras para desarrollar y probar tu código, pero que las prácticas se corregirán en un entorno Linux, y en particular usando `gcc` y `make` y probando los programas obtenidos en Linux.

El objetivo de este documento es explicar la configuración de Visual Code para poder usar el depurador, tanto en Linux como en WSL.

Configuración ultrarápida: ve al TL;DR de la sección 3.

1 Si usas Windows, instala WSL

WSL es el Windows Subsystem for Linux. Es una implementación nativa de Linux (por defecto de Ubuntu, aunque permite elegir otras distribuciones) que hace innecesarias otras alternativas como Cygwin, MinGW o git bash.

En Windows 10, WSL permite el uso de cualquier programa de Ubuntu en la terminal, pero no permite el uso de programas con interfaz gráfica. En Windows 11, se pueden instalar y ejecutar también programas con interfaz gráfica. Para ello, basta con invocarlos desde una terminal de WSL.

También puedes, por supuesto, usar otras alternativas, como hacer una instalación completa de Linux en una configuración de dual boot (la máquina puede arrancar de forma alternativa con Linux o con Windows), o instalar una máquina virtual (por ejemplo Virtual Box) dentro de Windows con una instalación de Ubuntu u otra distribución de Linux.

Instalación de WSL

Busca las instrucciones de instalación con Google, dado que pueden variar con tu versión de Windows. (Así que no las busques en este documento, no están). Pueden ser tan sencillas como escribir `wsl --install` en una terminal de PowerShell.

Una vez instalado, puedes acceder a WSL de diferentes formas, pulsando la tecla de Windows y el nombre (puede ser Terminal, Ubuntu, WSL, ...).

Al abrir una terminal, puedes ver si es WSL u otra terminal de Windows (PowerShell, símbolo del sistema) de diferentes formas, una es escribir cualquier comando de Linux y ver si funciona, puedes por ejemplo escribir `uname` o `uname -a`, que te da información sobre la versión de Linux que tienes instalada.

Si tu usuario te aparece como `root`, estás haciendo algo mal. Usar Linux como usuario raíz es la receta ideal para cargarse el sistema. Soluciona el problema antes de seguir.

Paquetes necesarios en WSL

Asegúrate de que tienes instalados los paquetes necesarios:

- aprende sobre los comandos de gestión de paquetes en Linux, en particular `sudo apt update`, `sudo apt upgrade`, `sudo apt install <nombre_paquete>` y `sudo apt remove <nombre_paquete>`

- prueba los comandos `gcc`, `make` y `gdb` en la terminal de Ubuntu. Si no los encuentras (al escribirlos en la terminal dan un error del tipo "command ... not found"), puedes instalarlos escribiendo en la terminal `sudo apt install build-essential` y si alguno sigue faltando, `sudo apt install <nombre_paquete>`, por ejemplo `sudo apt install gdb` (este es necesario para poder depurar tu código)

Sistema de archivos de WSL

El sistema de archivos de WSL es como el de cualquier Ubuntu, en particular tu carpeta personal estará en `/home/<tu_nombre_de_usuario>` y en general el directorio raíz `/` tiene la misma estructura que en Ubuntu.

Acceder a los ficheros que ves desde Windows es diferente. Desde WSL, los contenidos del disco `C` es el directorio `/mnt/c`, y a partir de ahí puedes ver toda la estructura de directorios a la que estás acostumbrado, como `/mnt/c/Archivos de Programa`, `/mnt/c/Users` (y dentro tu carpeta personal), etc.

[Nota: `/mnt/`se refiere a que el sistema de archivos de Windows está 'montado' en el directorio `/mnt/`; si tienes otras particiones en Windows, también aparecerán aquí; y `~` en WSL/Linux se refiere a la carpeta personal del usuario, por ejemplo `/home/alvaro`)]

Es muy recomendable que aprendas sobre la creación de alias y variables en bash, y en particular en el fichero `~/.basrhc/`, así como enlaces simbólicos, para facilitar la gestión de estos dos sistemas de archivos paralelos.

Por ejemplo, en `/home/alvaro` tengo enlaces simbólicos como estos:

```
alvaro@EPSINF3422G1:~$ ls -l
total 20
lrwxrwxrwx  1 alvaro alvaro   34 Feb 16  2023 Downloads ->
/mnt/c/Users/AV.5023306/Downloads/
lrwxrwxrwx  1 alvaro alvaro   32 Apr 26  2021 Dropbox ->
/mnt/c/Users/AV.5023306/Dropbox/
lrwxrwxrwx  1 alvaro alvaro   39 Mar 30  2022 OneDrive ->
'/mnt/c/Users/AV.5023306/OneDrive - UAM/'
```

para poder acceder fácilmente a ciertas partes del sistema de archivos de Windows desde WSL; y en `~/.bashrc` tengo variables como:

```
export ALVARO="/mnt/c/Users/AV.5023306/"
export EDAT="/home/alvaro/algun_dir/edat24/"
```

con el mismo objetivo. Además, en `~/.bashrc` puedes añadir alias como

```
rm="rm -i"
alias mv="mv -i"
alias cp="cp -i"
```

que te pueden ser útiles.

En todo caso, el resultado de hacer esto es que hay ficheros que "tienen dos nombres", y esto a veces se ve al usar vscode, y en concreto al usar el depurador. Por ejemplo, un fichero que tenga en el directorio de Dropbox va a estar "a la vez" en /home/alvaro/Dropbox y en /mnt/c/AV.5023306/Dropbox. Son el mismo fichero, solo que nos podemos refer a él con dos nombres diferentes.

2. Uso de VScode como entorno de programación

En vscode, instala la extensión de Microsoft para C/C++. También puedes necesitar su C/C++ Extension Pack.

```
Si usas WSL, puedes necesitar también la extensión WSL de Microsoft para
vscode.

Para usar vscode desde WSL, escribe `code <nombre_dir>` en la terminal, por
ejemplo `code practica1` para abrir la carpeta `practica1`. Además, si
abres una terminal dentro de vscode (View->Terminal o Ctrl+ñ en Windows),
asegúrate de que ésta es de tipo `bash` (que es la shell más común para
Linux, y la que viene con WSL). Puedes seleccionar el tipo de terminal de
un desplegable.
```

El resto de las instrucciones de esta guía se aplican al uso de VSCode **en cualquier Linux**, incluido WSL, pero también por supuesto una instalación de Ubuntu normal o en una máquina virtual, etc.

Un uso muy común de esta sintaxis es escribiendo en la terminal "`code .`"(con un espacio y un punto al final). Esto te abre vscode en el directorio `.`, que en Linux denota el directorio actual, igual que `..` denota el directorio superior al actual.

Hay guías varias en la documentación de VSCode (y de WSL) sobre como desarrollar en C con VSCode. Recomendamos consultarlas, aunque a veces contienen demasiada información irrelevante. Y tienes también por supuesto el material de otros cursos de la EPS.

Como probablemente ya sabes, si tienes las extensiones indicadas instaladas, deberías tener syntax highlighting e intellicode completion, es decir: el código te aparece con colores, y vscode te sugiere formas de completar lo que escribes. Además, con el ratón derecho sobre un texto, te saldrá un menú contextual con diversas opciones, como "Go to definition" o "Go to declaration". En caso de que algún símbolo no esté definido relativo a un fichero (por ejemplo, en el fichero se usa función para la que no se encuentra su declaración - su prototipo - ni en el propio fichero ni en ninguno de sus `#includes`), te saldrá una línea ondulada roja debajo.

Importante: En Settings (Ajustes o Configuración, el símbolo del engranaje abajo a la izquierda), busca **"Format on Save"** y habilítalo (marca la checkbox). Esto hará que cada vez que guardes un fichero de C (.c, .h...), el código se formatee de una forma adecuada (sangrías correctas, saltos de línea, etc.), lo que lo hace mucho más legible y permite identificar muchos fallos fácilmente (llaves que no cierran donde crees, etc.). Es posible personalizar el formato buscando "C format" en los ajustes, específicamente los ajustes de "C_Cpp: Clang_format_style" (y fallback style) se pueden cambiar (yo tengo LLVM como formato, porque no me gusta

que las llaves de apertura empiecen en una nueva línea). Es posible que sea necesario el programa clang-format, que puedes instalar del mismo modo que cualquier otro paquete.

3. Depuración en VScode: Preparación

Aquí queríamos llegar 😃

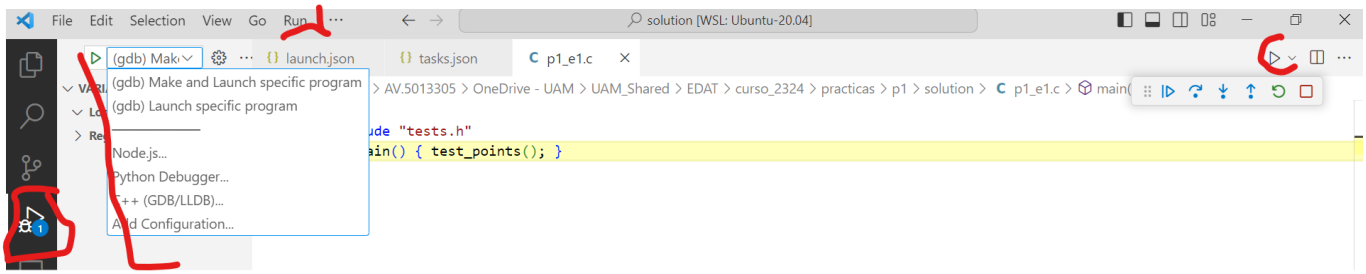
Antes de empezar: asegúrate de que compilas todos tus programas con la opción `-g`. Esto hace que el compilador `gcc` almacene información de depuración dentro de los ficheros ejecutables (por ejemplo, nombres de variables y funciones, nombres de ficheros y números de línea donde está definida cada función).

TL/DR (resumen rápido): Copia el fichero `launch.json` proporcionado al subdirectorio `.vscode` dentro de la carpeta con el código de la práctica (si el subdirectorio no existe, créalo), **y modifica la línea `program` de `launch.json` para que tenga el nombre del programa que quieres ejecutar**, por ejemplo, que su valor sea `"${workspaceFolder}/p1_e1"` para ejecutar `p1_e1`desde el directorio de trabajo (el que tienes abierto con Code).

Una vez hecho esto, puedes ejecutar el programa usando

- con el menú Run->Start debugging.
- o F5
- o con el botón de play que está arriba a la derecha o en la barra izquierda

Tendrás que seleccionar la opción `(gdb) Launch specific program`. Todo esto, cuando tienes un fichero de C abierto. También tienes otra opción `(gdb) Make and Launch specific program` que compila los programas ejecutando `make` antes de correr el programa seleccionado. (Ver más abajo si quieres cambiar los argumentos de `make`.)



En los labs!!!!

Si esto no funciona en los ordenadores de los laboratorios, en Linux, borrar los ficheros *.json contenidos del fichero `~/.config/Code/User/` para eliminar la configuración creada en otras clases, y volver a probar. (Esta configuración se restaura automáticamente en el siguiente login)

Más extensamente

Aunque los printfs pueden ser muy útiles para depurar un programa, el uso de un depurador es a menudo mucho más efectivo. Un depurador te permite ejecutar un programa (en este caso dentro de vscode) de diversas maneras, incluyendo línea a línea, o parando en determinados puntos (mediante "breakpoints"). Cuando el programa está parado, te permite examinar el valor de cualquier variable con alcance definido en

el punto de parada, y también, lo cual es *muy importante y potente*, evaluar cualquier expresión usando los nombres de variables y funciones disponibles en ese punto.

Quizás hayas oído hablar de gdb, que es uno de los depuradores por excelencia, y también el nombre de un formato para almacenar la información de depuración (de hecho `-g` es una abreviatura para `-ggdb`, existen otros posibles formatos, pero no nos interesan).

Como muchas otras IDEs, VScode tiene un depurador gráfico integrado basado en gdb. Hay una referencia de configuración del depurador aquí https://code.visualstudio.com/docs/cpp/launch-json-reference y una guía de uso aquí https://code.visualstudio.com/docs/editor/debugging, pero en este documento vamos a cubrir algunos aspectos básicos de la configuración necesaria para poder empezar a usar el depurador.

Configuración local del depurador (para el directorio de trabajo actual)

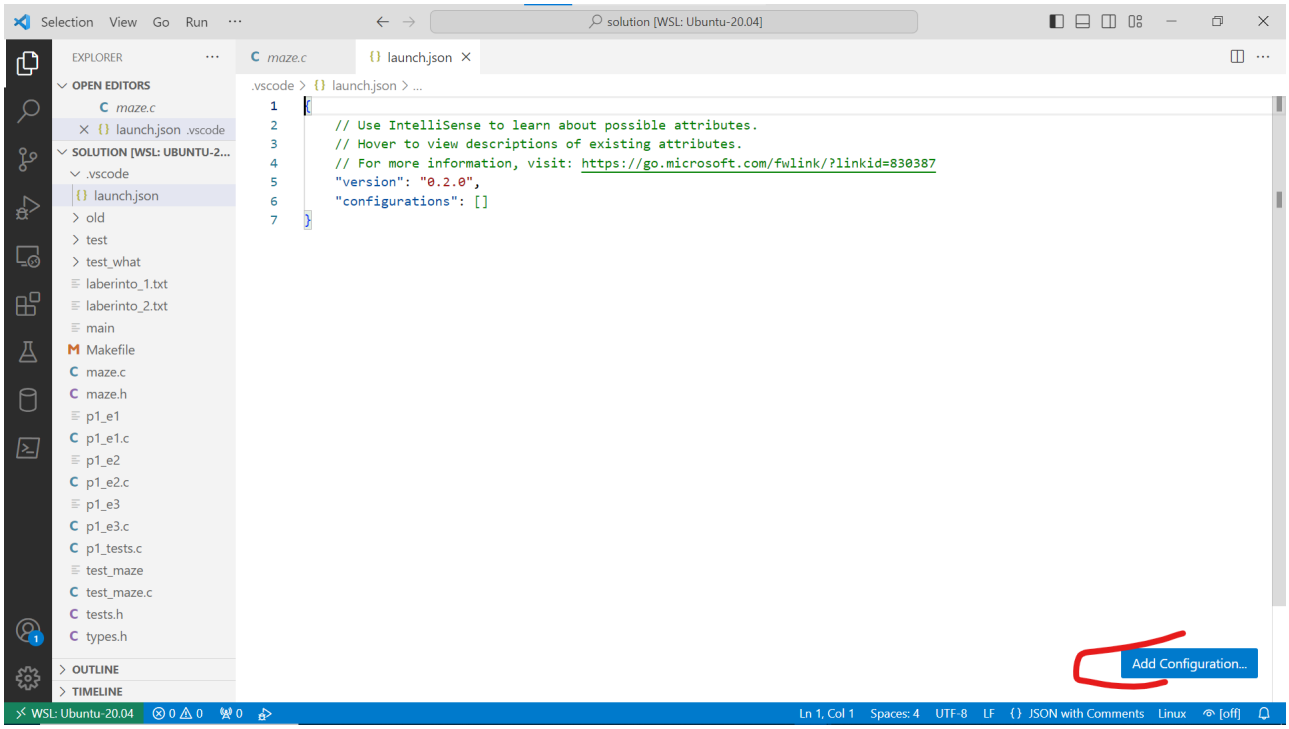
Solo para ejecución

La configuración de depuración puede ser local a un proyecto dado (donde "proyecto" es en realidad sinónimo de "directorio") o usar una configuración global. Puedes ver qué opciones tienes habilitadas pulsando el botón de play (cuando tienes un fichero de C abierto) que está arriba a la derecha o en la barra izquierda. También puedes acceder con menú Run->Start debugging.

La configuración local se almacena en un subdirectorio `.vscode` del directorio en el que está abierto vscode. En este directorio hay que crear (si no existe) un fichero `launch.json` que contiene información sobre la configuración de depuración. Un ejemplo de este fichero es el proporcionado y ya mencionado anteriormente en el TL;DR de esta sección. Si has usado el método del TL;DR, puedes saltarte esta sección (aunque pensamos que es bueno que sepas generarlo).

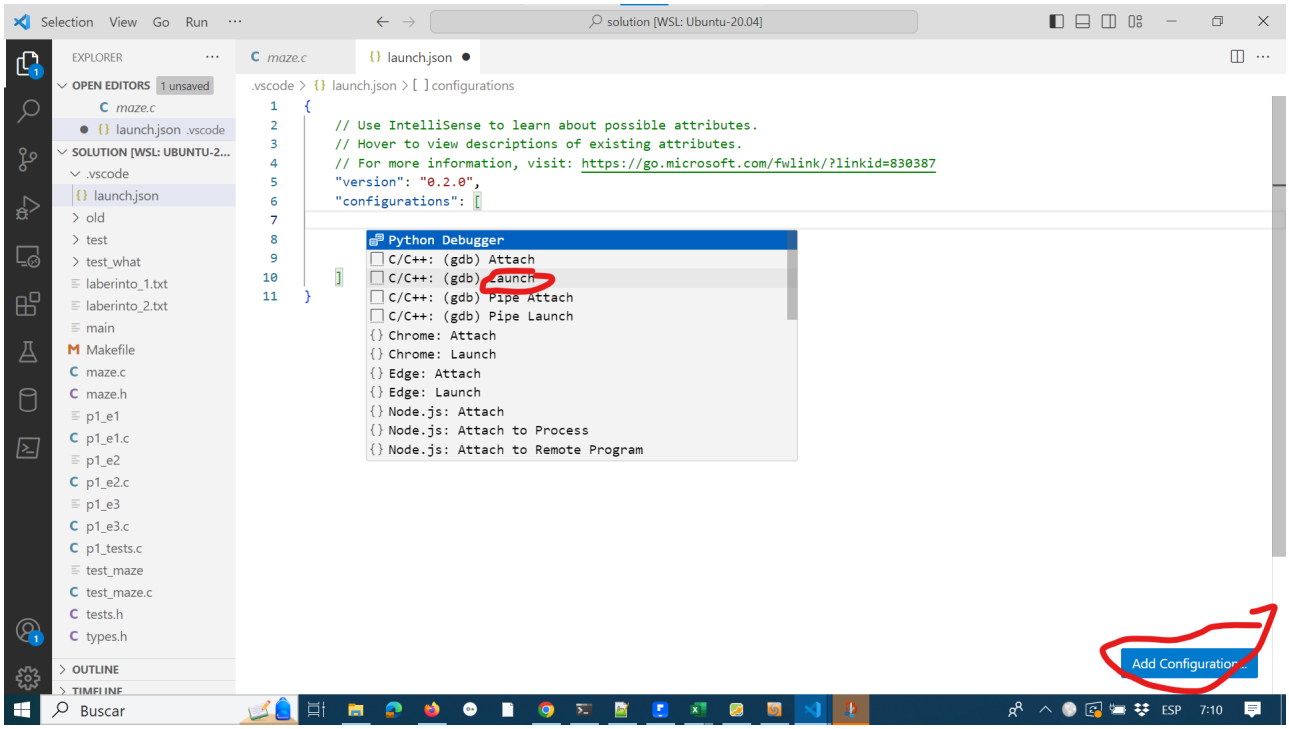
Para crear este fichero json, puedes seguir los siguientes pasos también:

- En el menú de VSCode, seleccionar Run->Add configuration. Os aparecerá lo siguiente:

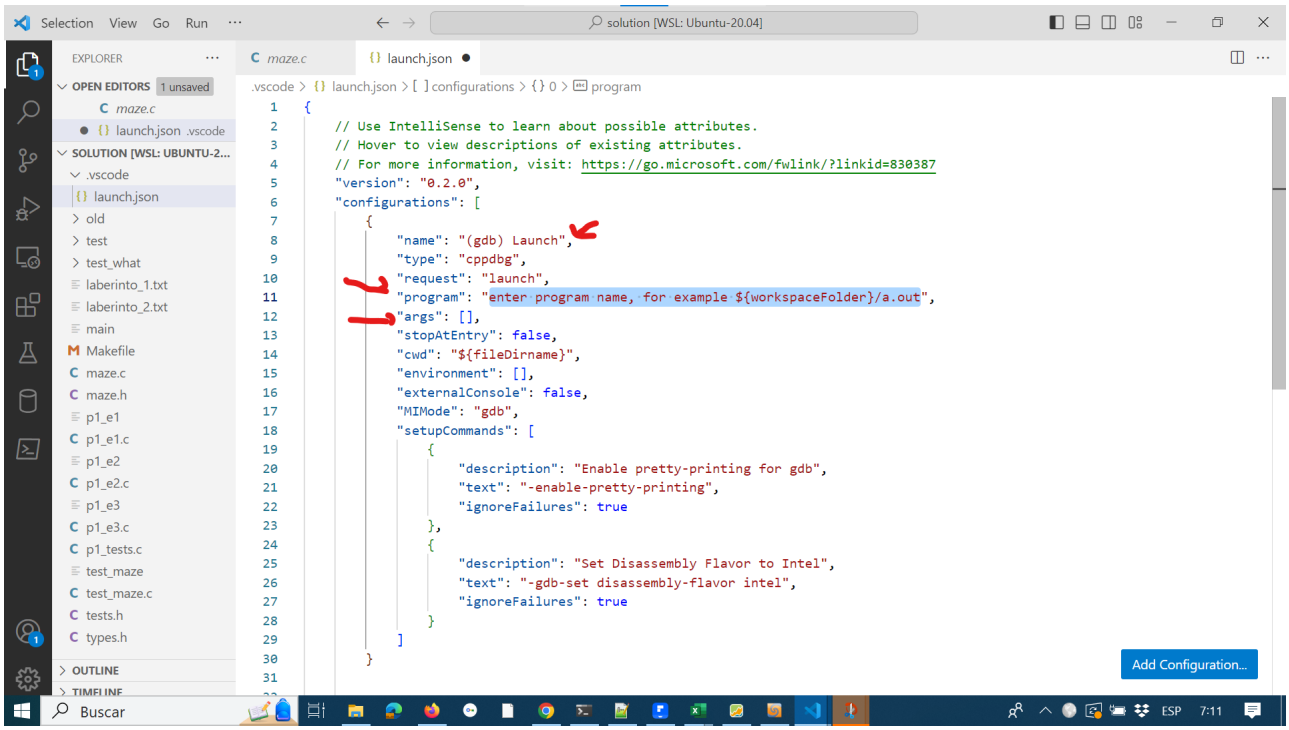


  Observad que el nombre del fichero (pestaña superior) es `launch.json`

- Pulsar el botón de Add Configuration en la esquina inferior derecha y seleccionar "C/C++: (gdb) Launch" como tipo de configuración en el menú que se abre



- El texto que se añade al fichero tras el último paso se ve así:



. El valor de "program" en este texto debe cambiarse para que sea `"${workspaceFolder}/tu_programa"`, por ejemplo `"${workspaceFolder}/p1_e1"`

Se pueden cambiar también `"stopAtEntry"` a `true` (si se quiere que el programa se detenga en la primera línea del `main`), y los `"args"` si el programa va a aceptar argumentos.

Además es recomendable cambiar el `"name"` a algo que te sea fácilmente reconocible. Por ejemplo, en el `launch.json` que os proporcionamos, lo hemos llamado "(gdb) Launch specific program", para recordarnos que es un programa específico (el de `"program"`) el que se va a ejecutar. Pero no es más que un nombre, si lo quieres llamar "Ejecuta mi programa", puedes hacerlo, y esto te saldrá entre las opciones de ejecución.

Finalmente, guarda el fichero. Puedes comprobar que `./.vscode/launch.json` existe y tiene el contenido que acabas de modificar.

Para compilación y ejecución

La solución de la subsección anterior requiere que el programa a ejecutar esté ya compilado. Siempre puedes recompilar escribiendo `make` en la terminal, que será algo más rápido si abres la terminal de `bash` dentro de VSCode. Pero si estás en un ciclo edición/compilación/prueba/edición... esto puede ser algo lento. La solución es que al pulsar F5 o darle a ejecutar, te recompile el programa antes de ejecutarlo, sin ningún paso adicional. Esto se hace añadiendo una "prelaunchTask" a una configuración similar a la que acabamos de crear. Esto ya está hecho en el `launch.json` proporcionado; el nombre de la configuración contiene "...Make and launch...".

Esta prelaunchTask se configura en otro fichero, `tasks.json`, que está configurado a nivel global (ver más abajo). Si no funcionara, lo mejor es que añadáis el fichero `tasks.json` que también os proporcionamos, al subdirectorio `.vscode`. Lo que hace esta tarea "prelanzamiento" (prelaunch) es simplemente llamar a `make` en el directorio de trabajo antes deejecutar el programa indicado en `"program"`.

Configuración global (para todos los proyectos de VSCode)

¿Y si no quieres repetir este proceso para cada proyecto (creando un nuevo fichero`launch.json` cada vez) o quieres modificar los argumentos del `make`? Es posible configurar los ajustes de forma global, de forma que solo se necesite cambiar el nombre del programa a ejecutar cada vez, sin necesidad de crear nuevos ficheros json para cada proyecto.

Puedes acceder a los ajustes globales con Ctrl+Shift+P Tasks:Open User Tasks (escribe cualquier parte del comando después del atajo de teclado para que te aparezca) o Ctrl+Shift+P Preferences:Open User Settings (json).

El primer comando te abre el fichero `tasks.json`, donde puedes ver la configuración *global* de la tarea `make` que hemos usado antes. Puedes anotar el lugar donde está almacenado este fichero (en Linux, es en `~/.config/Code/User`, en WSL en `AppData/Roaming/Code/User/`; pero siempre puedes usar `locate tasks.json` en la terminal para encontrarlo).

Se pueden modificar aquí los argumentos de `make` (cambiando el valor de `"args"`), o puedes copiar el fichero al subdirectorio `.vscode` de tu directorio de trabajo y modificarlo ahí.

El segundo comando te abre el fichero `settings.json`, que contiene los ajustes globales y que está en el mismo directorio que el `tasks.json`. Este fichero se puede usar para añadir configuraciones globales de ejecución. En este caso, no se puede definir un fichero `launch.json` global, pero sí se puede incluir sus configuraciones en este `settings.json`, con el resultado deseado. Todo lo que hay que hacer es añadir una nueva línea `"launch":` a `settings.json` (y una coma `","` al final de la propiedad que preceda a la nueva línea) y copiar ahí como valor todo el contenido del `launch.json` que os hemos proporcionado (ver https://github.com/Microsoft/vscode/issues/18401#issuecomment-272400316 para un ejemplo).

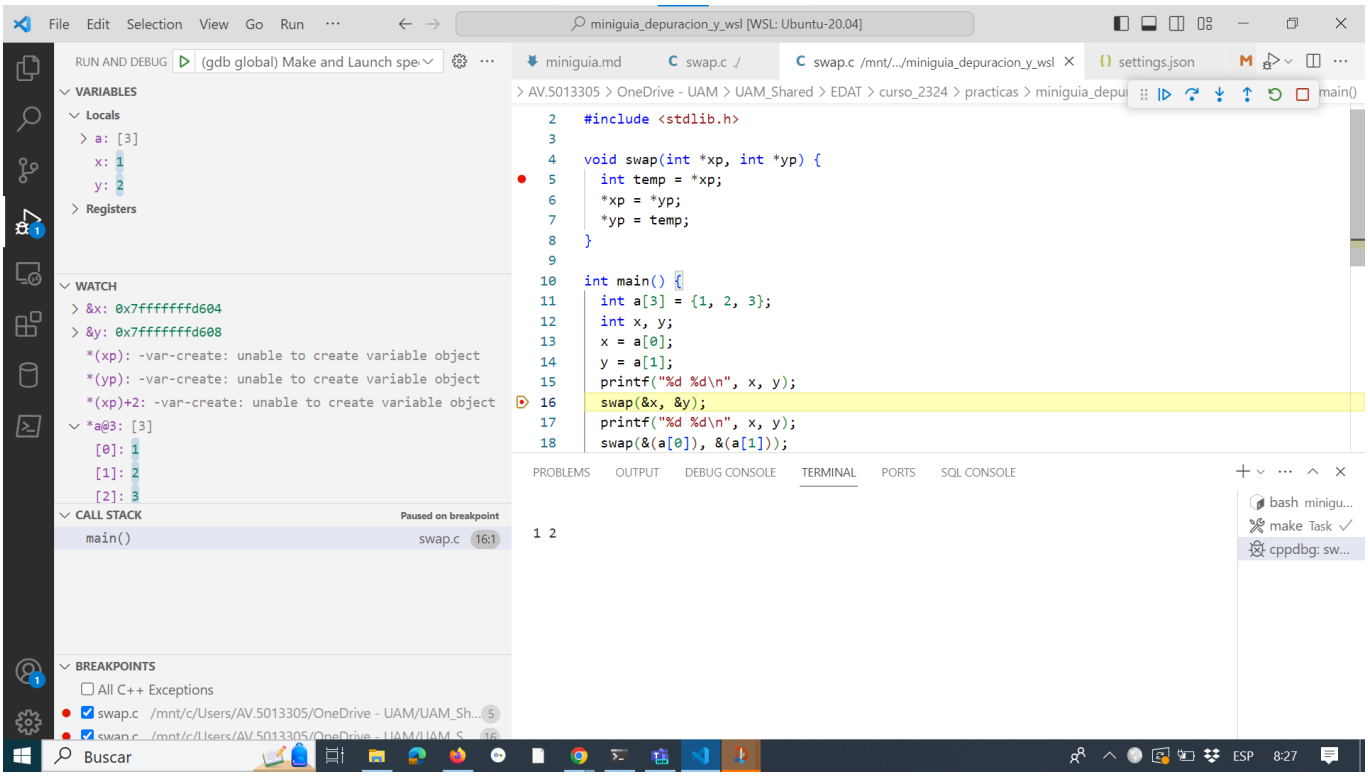
4. Depuración en Vscode: Uso

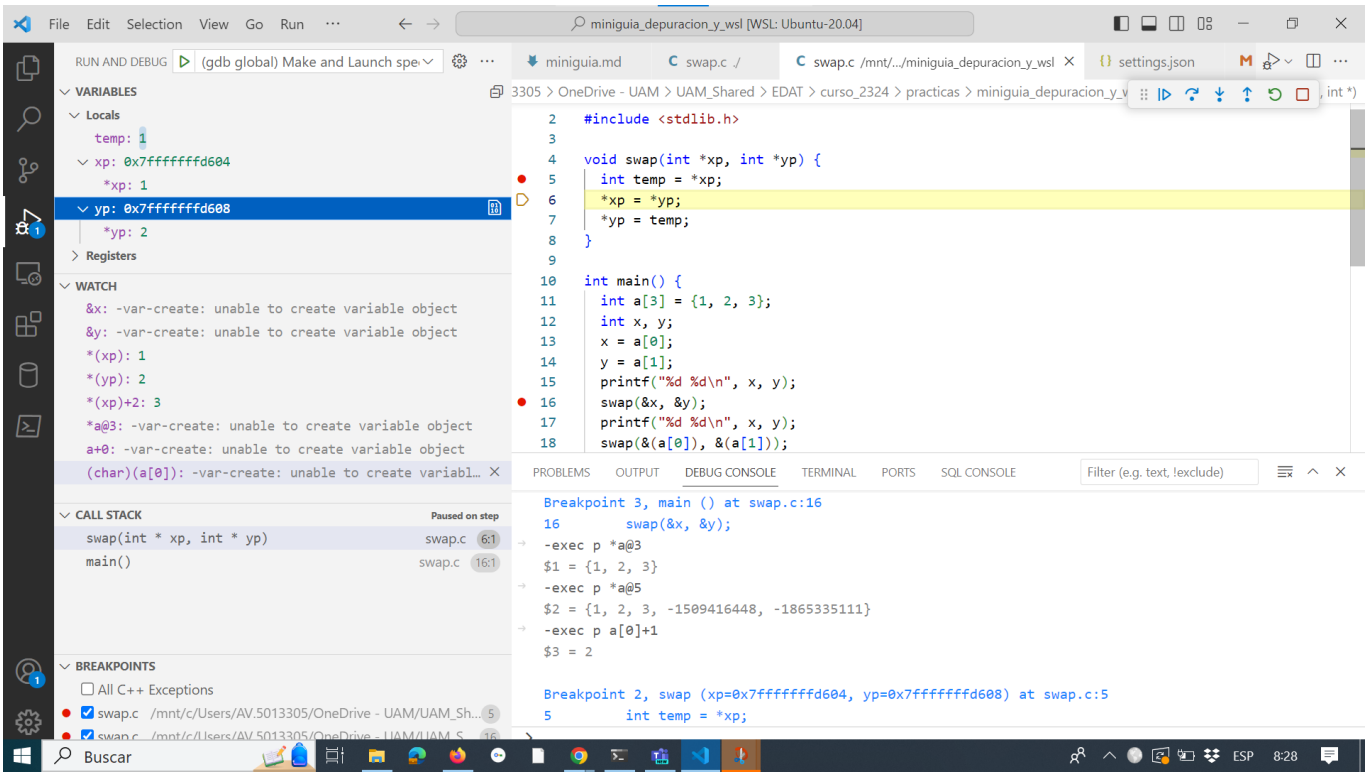
Bueno, en realidad, es **aquí** donde queríamos llegar 😃

Os proporcionamos un pequeño ejemplo con la función de swap para que podáis ver algunas de las funcionalidades del depurador. Observad:

- poned breakpoints (marcados con un punto rojo) con un click a la izquierda de la línea donde queréis que se detenga el programa
- variables y arrays pueden verse en el panel superior izquierdo ("Variables")
- El segundo panel izquierdo permite evaluar el valor de expresiones arbitrarias (añadir nuevas expresiones con '+')
- La "debug console", panel inferior derecho, os permite ejecutar cualquier comando de gdb. En la imagen se ilustra el imprimir arrays (pero también se puede hacer en el panel de Watch)

Viendo las imágenes, podéis jugar a ver los valores de las variables, incluyendo las variables de tipo puntero en diferentes puntos de la ejecución; y, en el caso de punteros, de los lugares de memoria a los que apuntan. ¿Puedes ver, por ejemplo, la diferencia entre el swap de x e y, y los dos swaps de a[0] y a[1]? ¿Afecta el primer swap al array? ¿Y el segundo? ¿Por qué?




File Edit Selection View Go Run
miniguia_depuracion_y_wsl [WSL: Ubuntu-20.04]
RUN AND DEBUG
(gdb global) Make and Launch spe
VARIABLES
Locals
temp: 1
xp: 0x7fffffffd604
*xp: 1
yp: 0x7fffffffd608
*yp: 2
Registers
WATCH
&x: -var-create: unable to create variable object
&y: -var-create: unable to create variable object
*(xp): 1
*(yp): 2
*(xp)+2: 3
*a@3: -var-create: unable to create variable object
a+0: -var-create: unable to create variable object
(char)(a[0]): -var-create: unable to create variabl…
CALL STACK
Paused on step
swap(int * xp, int * yp) swap.c 6:1
main() swap.c 16:1
BREAKPOINTS
All C++ Exceptions
swap.c /mnt/c/Users/AV.5013305/OneDrive - UAM/UAM_Sh… 5
miniguia.md
swap.c ./
swap.c /mnt/.../miniguia_depuracion_y_wsl
settings.json
3305 > OneDrive - UAM > UAM_Shared > EDAT > curso_2324 > practicas > miniguia_depuracion_y_v
2 #include <stdlib.h>
3
4 void swap(int *xp, int *yp) {
5 int temp = *xp;
6 *xp = *yp;
7 *yp = temp;
8 }
9
10 int main() {
11 int a[3] = {1, 2, 3};
12 int x, y;
13 x = a[0];
14 y = a[1];
15 printf("%d %d\n", x, y);
16 swap(&x, &y);
17 printf("%d %d\n", x, y);
18 swap(&(a[0]), &(a[1]));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE
Filter (e.g. text, !exclude)
Breakpoint 3, main () at swap.c:16
16 swap(&x, &y);
-exec p *a@3
$1 = {1, 2, 3}
-exec p *a@5
$2 = {1, 2, 3, -1509416448, -1865335111}
-exec p a[0]+1
$3 = 2
Breakpoint 2, swap (xp=0x7fffffffd604, yp=0x7fffffffd608) at swap.c:5
5 int temp = *xp;
Buscar
ESP 8:28