

## Introducción al Depurador Code::Blocks

El propósito de un depurador es permitirle al programador observar lo que ocurre en el “interior” de un programa mientras éste se está ejecutando. Es una aplicación que permite colocar puntos de parada, inspeccionar variables o ejecutar un programa paso a paso, con el fin de buscar errores lógicos. Los errores sintácticos y/o de ejecución pueden ser fácilmente detectados ya sea en el momento de la compilación (a través de mensajes), o al ejecutar el programa. Sin embargo en el caso de los errores lógicos, se puede dar que las instrucciones se ejecuten correctamente y que la compilación sea exitosa (sin mensajes de error ni warning). En este caso el programador ha proporcionado instrucciones incorrectas (inadecuada implementación del algoritmo).

Para emplear esta herramienta es necesario que el código fuente sea compilado sin errores. No obstante el programa aún puede tener un comportamiento inesperado, debido a la presencia de bugs (mal diseño de un algoritmo).

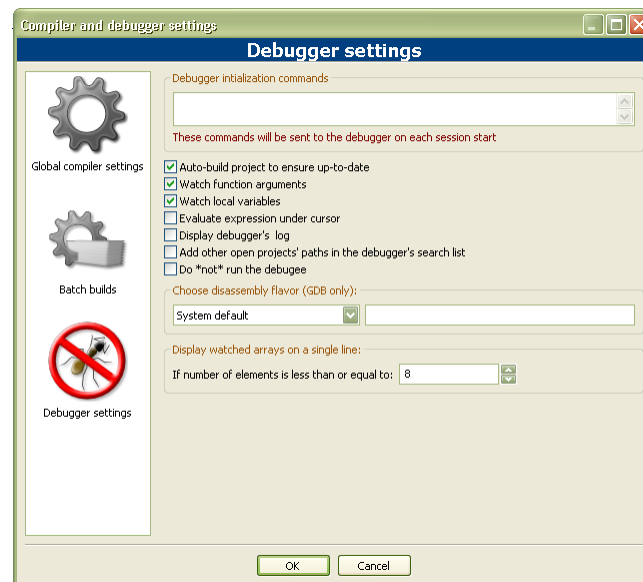
Requisitos para depurar un programa:

- Obtención de un ejecutable .exe
- Información para la depuración contenida en el ejecutable .exe.
- Añadir al menos un punto de interrupción o Breakpoint.

No es posible obtener un ejecutable en caso de errores de compilación o de enlazado, y la depuración no tiene sentido hasta que se solucionen estos problemas.

La siguiente figura muestra la configuración por defecto que define el comportamiento del depurador. Se encuentran activas:

- Auto construcción del proyecto para asegurar que siempre esté actualizado.
- Observación de argumentos de función<sup>1</sup>
- Observación de variables locales



**Fig. 1 Configuración del depurador**

1. Objeto de posterior estudio

Definiciones:

**Debugger:** Depurador de código.

**Proyecto:** El entorno de desarrollo ofrece al programador, una herramienta de administración y organización de todos los archivos fuente del programa. De esta manera el proyecto contará con una carpeta especial para almacenar los archivos de cabecera con extensión .h, otra para los archivos fuentes con extensión .c, etc.

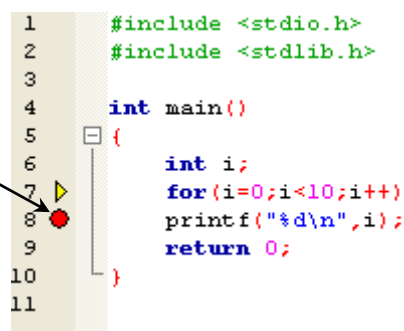
Esto podría parecer de poca utilidad para quienes recién se inician en programación, sin embargo en la práctica es bastante frecuente encontrar un programa compuesto por una gran cantidad de archivos fuente. Es importante mencionar que para poder utilizar el depurador debemos crear un proyecto.

**Breakpoint:** Es un punto de interrupción en la ejecución del proceso o programa en ejecución. Los Breakpoints permiten especificar detenciones del programa, para observar valores de memoria, registros o variables durante la ejecución.

### Puntos de interrupción o Breakpoints

Existen varias formas alternativas de activar un punto de interrupción, Breakpoint.

Ubicar el cursor haciendo clic sobre la línea de código en la que deseamos marcar un breakpoint, luego presionando la tecla **F5** o desde el menú **Debug ► Toggle Breakpoint**. Observaremos que se representan con un círculo de color rojo en el margen izquierdo de la ventana del editor de código.



```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      for(i=0;i<10;i++)
8      printf("%d\n",i);
9      return 0;
10 }
11
```

**Fig. 2 Punto de interrupción Breakpoint**

**Nota:** No se deben ubicar sobre las declaraciones de variables de programa (o línea n° 6 del programa de ejemplo). Estas instrucciones efectúan reserva de memoria estática en el momento de la ejecución del programa. El programador **puede** abstraerse de estas cuestiones ya que los recursos del hardware son administrados por el sistema operativo.

Además podrían no funcionar si la ruta/archivo donde está alojado el proyecto contiene espacios en blanco u otros caracteres especiales. Se recomienda usar solo letras y símbolos \_ en lugar de espacios.

### Inicio del depurador

Una vez construido el proyecto, si el programa se a compilado sin errores y se han marcado los Breakpoints, es posible iniciar la depuración desde el menú **Debug ► Start** o presionando la tecla **F8**.

La ejecución desde el inicio del programa hasta la primera línea de código marcada con el Brekpoint se efectúa de la manera convencional. Una vez alcanzado este punto observaremos que nuestro proceso se detiene en la línea número 8 del ejemplo, correspondiente al punto de interrupción (breakpoint) previamente configurado. Hay de

destacar que esta línea de código aun no ha sido ejecutada, lo cual es representado por una flecha amarilla (muestra cual será la siguiente línea de código a ejecutar).

La ventana o consola de nuestro programa ya no contiene el foco (interacción con el usuario), porque ahora es la ventana del editor quien lo tiene. Esto suele ser una fuente de confusión, cuando por ejemplo nuestro programa espera que ingresemos datos por teclado. En estos casos la depuración no podrá continuar hasta que el usuario ingrese los datos, independientemente de lo que se intente hacer dentro de la ventana del entorno de desarrollo o IDE (Integrated Development Environment).

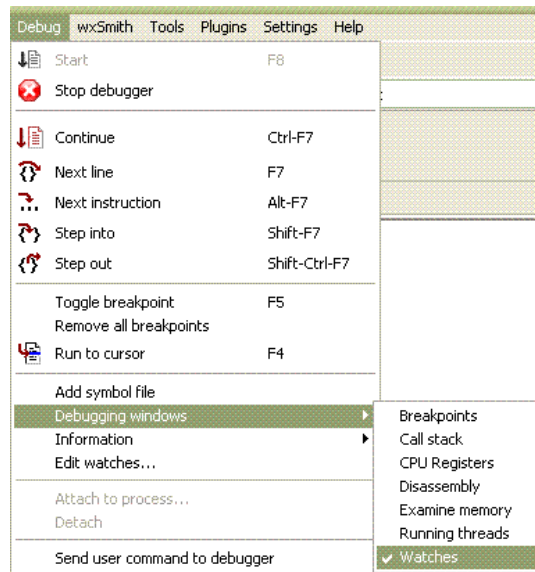


Fig. 3 Menú de herramientas del depurador

## Modos de ejecución

Las opciones que se describen a continuación, permiten al programador ejecutar en modo “manual” las partes del código donde se presume que existen errores lógicos y ameriten un análisis detallado. Como así también ejecutar de manera “automática” las partes del código que se presumen libres de errores.

La depuración posibilita la ejecución paso a paso de cada línea, presionando la tecla **F7** o desde el menú **Debug ► Next line**. Si la línea contiene una llamada función<sup>1</sup>, la misma se ejecuta completamente sin mostrar paso a paso cada una de las instrucciones que hay en ella.

1. Objeto de posterior estudio

También se puede optar por ejecutar el programa hasta el siguiente punto de interrupción, de modo que el depurador continuará la ejecución del programa hasta que encuentre el próximo Breakpoint o el final del código, seleccionando **Debug ► Continue (Ctrl-F7)**.

Continuar con la siguiente línea de código y hacer paso a paso dentro de una llamada a función<sup>1</sup>, mediante **Debug ► Step Into (Shift-F7)**.

1. Objeto de posterior estudio

Reanudar la ejecución paso a paso o ejecutar la siguiente sentencia, mediante **Debug ► Next Instruction Alt-F7**. No es recomendable el uso de ésta última debido a que está

diseñada para instrucciones en lenguaje ensamblador o lenguaje de máquina (programación de bajo nivel).

También es posible ejecutar de modo “automático” hasta la posición actual del cursor, mediante **Debug ► Run to cursor (F4)**.

Continuar con la ejecución hasta que finalice la estructura de control actual, mediante **Debug ► Step out (Shift-Ctrl-F7)**. Si actualmente estuviésemos depurando una función<sup>1</sup>, terminaría su ejecución y retornaría al origen de la llamada.

1. Objeto de posterior estudio

### Observación de variables

Luego desde el menú **Debug ► Debugging Windows**, debemos activar “**Watches**” lo cual nos abrirá una ventana con las variables locales que hemos declarado en nuestro código fuente.

Las variables locales de nuestro código, serán mostradas en la ventana “**Watches**” y aquellas recientemente actualizadas serán resaltadas en color rojo.

Esta lista de variables puede ser salvada a un archivo y posteriormente ser recuperada. Esto se logra haciendo clic con el botón derecho del Mouse en la lista de variables y seleccionando “**Save watch file**” y “**Load watch file**” para reabrirla posteriormente.

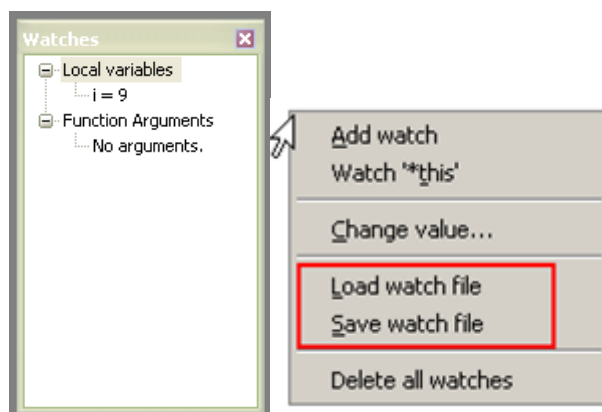


Fig. 4 Ventana de observación de variables y argumentos

El editor de variables observadas, ofrece la posibilidad de añadir/quitar y modificar el formato de representación de los datos contenidos durante la depuración. Se abrirá esta venta desde el menú **Debug ► Edit Watches ...**

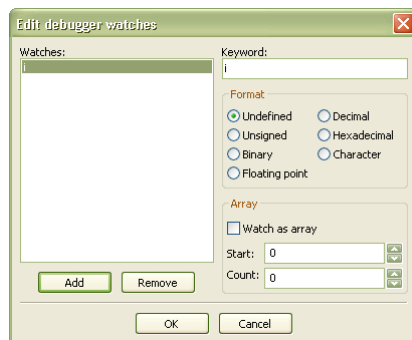


Fig. 5 Editor de variables observadas

Programación Algoritmos y Estructuras de Datos – año 2010  
Facultad de Ingeniería  
Universidad Nacional de La Plata

A continuación se ilustrará el uso del depurador con un sencillo programa de ejemplo, en el cual se pide mostrar en pantalla los caracteres desde de la A hasta la Z.  
El programador lo implementa de la siguiente manera:

1	#include <stdio.h>
2	
3	int main() {
4	
5	char a = 'A'; /* Declara una variable tipo caracter de 8 bits */
6	/* Inicializa su valor con la letra 'A' */
7	
8	while(1) { /* Este es un bucle "infinito" */
9	
10	putchar(a); /* Muestra el caracter actual, en la pantalla.
11	Empezando por la letra 'A' según valor inicial */
12	
13	if (a = 'Z') /* Si el contenido del caracter actual 'char a' es igual */
14	break; /* a 'Z' salimos del bucle "infinito" ejecutando break */
15	
16	a++; /* Aun quedan caracteres por mostrar, entonces pasamos al
17	siguiente de la tabla ASCII haciendo a = a + 1 */
18	
19	}
20	return 0;
21	}

A pesar lograr compilarlo sin mensajes de errores ni advertencias, se observa que salida no es la esperada al ejecutar el programa. En la pantalla solo se observa la letra A y luego finaliza la ejecución. El programador ha cometido un error involuntario y su programa tiene un error lógico.

Entonces estableciendo un breakpoint en la línea nº 10 e iniciando la depuración se observa que la variable **char a** toma el valor Z y luego la condición se evalúa como verdadera por ser distinta de cero (ver código ASCII), lo cual según las instrucciones conduce a la ejecución de break y luego a return 0 provocando el fin del programa.

Conclusión al observar la ventana de variables locales, vemos que **char a** toma el valor Z cuando lo que se intentó codificar fue que:

Si (**char a** es igual a Z)

Salir del bucle “infinito”

FinSi

Fin del módulo principal.

De modo que el error está en la condición evaluada dentro del paréntesis en la línea nº 13, debiéndose reemplazar por:

```
if (a == 'Z') /* Si el contenido del caracter actual 'char a' es igual */
```

La diferencia es sutil y es un error bastante frecuente el confundir los símbolos = con == (doble igual).

**Pasos para construir un proyecto**

Ir al menú **File ► New ► Project ...**

**Se abre una ventana que nos ofrece diferentes plantillas (Templates) para crear proyectos. A continuación seleccionamos proyecto vacío (Empty Project), luego hacer clic en siguiente, con lo cual se abrirá una ventana donde asignaremos el nombre del proyecto, las rutas donde se almacenarán los archivos.**