

Depuración de código con Nemiver

Programación 2

Grado en Ingeniería Informática
Universidad de Alicante
Curso 2024-2025



- Características
- Ventana principal de Nemiver
- Cargar un programa ejecutable
- Acciones básicas
- Líneas de parada
- Contexto
- Creación de expresiones
- Ejercicio

Características (1/2)

- Entorno de perspectivas: ahora sólo depuración
- Usa como motor el depurador de GNU gdb
- Obligatorio: añadir en compilación parámetro -g
- Acciones básicas:
 - Establecer líneas de parada en la ejecución de nuestras aplicaciones
 - Ejecutar instrucción a instrucción la aplicación
 - Analizar los valores que van tomando las variables y expresiones

Características (2/2)

- Finalidad: descubrir errores de ejecución de manera rápida
- Inicio:
 - Desde el icono de Nemiver
 - Desde línea de comando:

Terminal

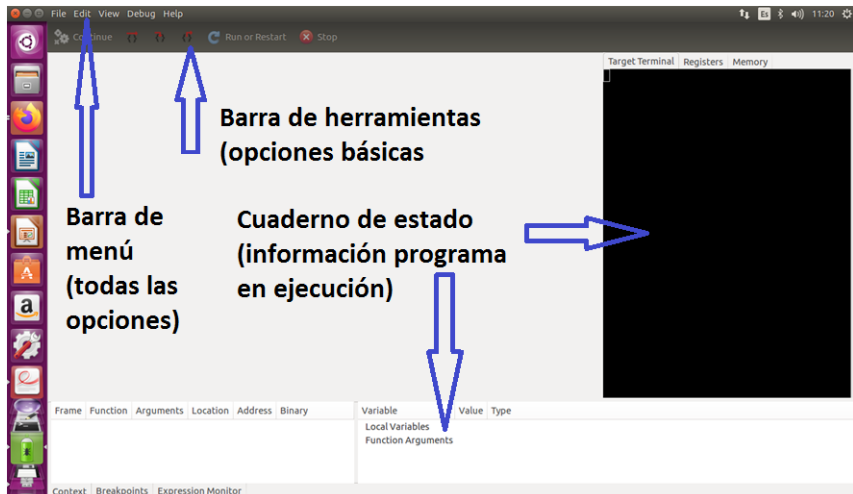
```
$ nemiver
```

- Consejo: ejecutar como proceso de fondo
- Libera el terminal para compilar:

Terminal

```
$ nemiver &
```

Ventana principal de Nemiver (1/2)



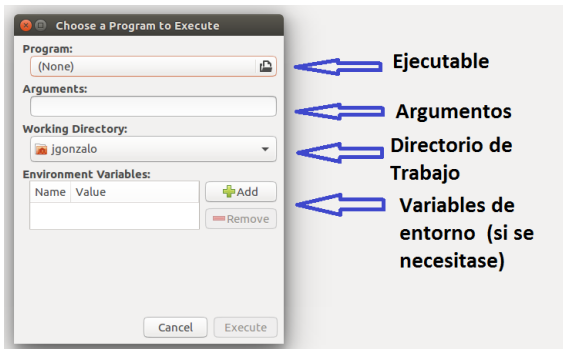
Ventana principal de Nemiver (2/2)

- Si no aparece así inicialmente, seleccionamos *Editar > Preferencias*:



Cargar un programa ejecutable (1/2)

- Desde dentro de Nemiver: *Archivo > Cargar ejecutable*:



- Desde línea de comando:

Terminal

```
$ nemiver <ejecutable> &  
$ nemiver <ejecutable> [param1] [param2] ... &
```

Cargar un programa ejecutable (2/2)

The screenshot shows the GDB interface with the source code of a program named `ejemplo-gdb.cc`. The code defines a `findCharacter` function and a `main` function. A breakpoint is set at line 25, which is the first line of the `main` function. A red circle with a yellow arrow points to this line, labeled "Punto de parada (breakpoint)". A double-headed arrow points from the text "Código fuente del ejecutable. NO ES EDITABLE: solo referencia para ejecución." to the source code area. Another arrow points from the text "Siguiete instrucción a ejecutar" to the breakpoint.

```
1
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6
7 /* Función que dada una cadena, busca la primera ocurrencia de un caracter en dicha cadena e imprime la s
8 Parametros: la cadena donde buscar y el carácter a buscar
9 Retorna: la posición dentro de la cadena del carácter a buscar. Retorna -1 si no
10 se encuentra.
11 */
12 int findCharacter(char word[], char c)
13 {
14     unsigned i;
15     int position=-1;
16     for (i=0; i<strlen(word) && word[i]!=c; i++) { // debe ser con '&&' e 'i=0';
17         cout << word[i];
18     }
19     if (i<strlen(word))
20         position=i;
21     return position;
22 }
23
24 int main()
25 {
26     int p;
27     char helloworld[]="hello, world!";
28
29     cout << "-----" << endl;
30     p=findCharacter(helloworld,'W');
31     cout << endl << "W) p=" << p << endl;
32
33     cout << "-----" << endl;
34     p=findCharacter(helloworld,'o');
35     cout << endl << "o) p=" << p << endl;
36
37     cout << "-----" << endl;
38     p=findCharacter(helloworld,'r');
39     cout << endl << "r) p=" << p << endl;
40
41     cout << "-----" << endl;
42     p=findCharacter(helloworld,'h');
43     cout << endl << "h) p=" << p << endl;
44
45 }
```

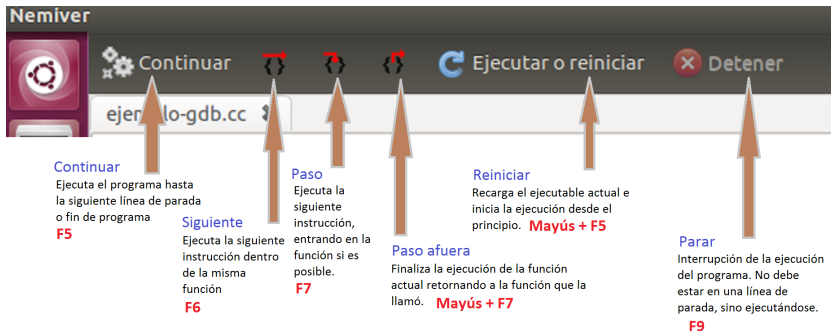
Terminal de destino: Registros Memoria

Línea: 25, columna: 1

Marco	Función	Parámetros	Ubicación	Dirección	Binario	Variable	Valor	Tipo
1	main	()	ejemplo-gdb.cc:25	0x0000555555554a76		Variables locales		
						p	21845	int
						helloworld	[13]	char [13]
						Parámetros de función		

Contexto: Puntos de parada Monitor de expresiones

Acciones básicas



Líneas de parada

The screenshot shows the NetBeans IDE with a C++ project named 'ejemplo-gdb.cc'. The code defines a function `findCharacter` that searches for a character in a string and returns its position. The `main` function calls `findCharacter` with the string "hello, world" and the character 'W'. The program outputs the position of 'W' (1) and the position of 'o' (9).

Annotations on the code:

- A blue arrow points to line 21, with the text "Siguiete instrucción a ejecutar" (Next instruction to execute).
- A yellow arrow points to line 24, with the text "Siguiete instrucción a ejecutar" (Next instruction to execute).
- A red circle is placed over the `main` function.

Terminal output:

```
hello, world
(W) p=1
-----
hell
(o) p=9
-----
hello, wo
(r) p=9
-----
(h) p=9
-----
```

Annotations on the terminal:

- A white arrow points to the terminal output, with the text "Terminal de salida de la ejecución" (Execution output terminal).

Breakpoints table:

ID	Nombre del archivo	Línea	Función	Dirección	Condición	Cambiar punto de conteo	Tipo	Impactos	Expresión	Ignorar cuenta
1	ejemplo-gdb.cc	19	findCharacter(char*, char)	0x00005555555544ca			punto de parada 1	0		
5	ejemplo-gdb.cc	25	main()	0x00005555555544fe			punto de parada 0	0		


Annotations on the breakpoints table:

- A white arrow points to the breakpoint at line 25, with the text "Desactivar/Activar líneas de parada con el ratón" (Deactivate/Activate breakpoints with the mouse).


Context: Puntos de parada | Monitor de expresiones

Contexto


Id	Marco	Función	Parámetros	Ubicación	Variable	Valor	Tipo
0		findCharacter	(word = 0x7fffffffddab "hello, world", c = 87 'W')	ejemplo-gd	Variables locales		
1		main	()	ejemplo-gd	i	12	unsigned int
					position	-1	int
					Parámetros de función		
					word	0x7fffffffddab "hello, world"	char *
					c	87 'W'	char



Variables



**Valores que
van tomando**



Tipos

Contexto

Puntos de parada

Monitor de expresiones

Creación de expresiones

38 p=findCharacter(helloworld,'r');
39 cout << endl << "(r) p=" << p << endl;;
40

Línea: 21, columna: 1

Variable Valor Tipo

En las expresiones del alcance
i<strlen(word)
word[i]!=c
word[i]
Fuera del ámbito de las expresiones

Nueva...
Quitar

1. Click dcho.

2. Nueva

3. Introducir expresión o variable

4. Pulsar

5. Pulsar

Contexto Puntos de parada **Monitor de expresiones**

Inspeccionar una expresión

Nombre de la variable: strlen(word) Inspeccionar

Variable	Valor	Tipo
strlen(word)	12	int

Añadir al monitor Cerrar

Ejercicio

- Practicar con Nemiver usando un código que contenga errores de ejecución:

```
int findCharacter(char word[],char c)
{
    unsigned i;
    int position=-1;
    for(i=1; i<strlen(word) || word[i]!=c; i++) { // Mal: debe ser con '&&' e i=0;
        cout << word[i];
    }
    if(i<strlen(word)){
        position=i;
    }
    return position;
}

int main(){
    int p;
    char helloworld[]="hello, world";
    p=findCharacter(helloworld,'W');
    cout << endl << "(W) p=" << p << endl;
    p=findCharacter(helloworld,'o');
    cout << endl << "(o) p=" << p << endl;
    p=findCharacter(helloworld,'r');
    cout << endl << "(r) p=" << p << endl;
    p=findCharacter(helloworld,'h');
    cout << endl << "(h) p=" << p << endl;
}
```