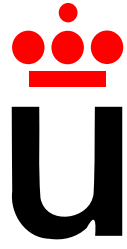
9. Depuración con GDB y documentación con Doxygen

Julio Vega

julio.vega@urjc.es



Universidad
Rey Juan Carlos

Contenidos

- Exisen dos tipos de errores: de compilación y lógicos (*bugs*).
- El compilador nos ayuda con los primeros, pero no con los segundos.
- Para ayudarnos con los errores lógicos está el depurador o *debugger*.
- Para sistemas GNU tenemos el GNU Debugger (GDB), *open-source*.
  - Si no lo tenemos instalado, instalar con: `sudo apt install gdb`.
- Existen IDEs con sus propios depuradores, y también con GDB.
  - E.g. los entornos de desarrollo *Visual Studio* o *Eclipse*.

- Lo primero es compilar el programa para que pueda ser depurado.
  - Para ello está la opción `-g`: `g++ -g -o myProgram myProgram.cpp`.
- Lo siguiente es iniciar el *debugger*: `gdb myProgram`
  - El `$` del Terminal desaparece; en su lugar se indica `(gdb)`
- El comando `list` nos permite ver el código del programa.
- Antes de lanzar el programa, establecer **puntos de interrupción**.
  - Son como marcadores que se pueden ponen a lo largo del código.
  - Cuando la ejecución del programa llega a estos, se detiene.
  - Para ponerlos se usa `break numlinea` (e.g. `(gdb)break 27`).
  - Para ver la lista de *breakpoints*: `info break`
  - Si queremos elimiar alguno: `delete numbreakpoint`

- Y ya se puede dar comienzo a la depuración, con el comando `run`
- El comando `continue` permite reanudar la ejecución tras *breakpoint*.
  - `next` avanza solo una línea del programa.
  - `step` avanza solo una línea y, si esta es una función, entra dentro.
- `print` nos permite ver el contenido de una variable.
- Si hay algún *bug*, el comando `where` nos hace un *backtrace*.
- Finalmente, `quit` termina la sesión de depuración.
- También se pueden usar las iniciales de los anteriores comandos.
  - E.g. r (run), s (step), n (next), p (print), b (break), q (quit).

```
01. #include <iostream>
02. using namespace std;
03.
04. double getDivision (int num, int den) {
05.   return static_cast<double> (num) / den;
06. }
07.
08. int main() {
09.   int n1 = 7, n2 = 3;
10.   cout << getDivision (n1, n2);
11.   n1 = 3; n2 = 0;
12.   cout << getDivision (n1, n2);
13.   return 0;
14. }
```

```
$ g++ -g -o div div.cpp
$ gdb div
```

```
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
This is free software: you are free to change and redistribute.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from div...
(gdb)
```

```
(gdb) break 9
Punto de interrupcion 1 at 0x11c5: file div.cpp, line 9.
(gdb) break 10
Punto de interrupcion 2 at 0x11d3: file div.cpp, line 10.
(gdb) break 12
Punto de interrupcion 3 at 0x1209: file div.cpp, line 12.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000000011c5 in main()
                                                   at div.cpp:9
2       breakpoint     keep y   0x00000000000011d3 in main()
                                                   at div.cpp:10
3       breakpoint     keep y   0x0000000000001209 in main()
(gdb)
```

```
(gdb) r
Starting program: div
[Depuracion de hilo usando libthread_db enabled]
Using host libthread_db library
    "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at div.cpp:9
9    int n1 = 7, n2 = 3;
(gdb) print n1
$1 = 0
(gdb) s

Breakpoint 2, main () at div.cpp:10
10   cout << getDivision (n1, n2);
(gdb) print n1
$2 = 7
(gdb) print n2
$3 = 3
(gdb)
```

```
(gdb) s
getDivision (num=7, den=3) at div.cpp:5
5    return static_cast<double> (num) / den;
(gdb) s
6  }
(gdb) s
main () at div.cpp:11
11   n1 = 3; n2 = 0;
(gdb) s

Breakpoint 3, main () at div.cpp:12
12   cout << getDivision (n1, n2);
(gdb) s
getDivision (num=3, den=0) at div.cpp:5
5    return static_cast<double> (num) / den;
(gdb) s
6  }
(gdb)
```

- El anterior programa no generaba fallo, por el casting a `double`.
  - La ejecución de la división por 0 mostrará INF o inf (infinito).
- Ahora cambiaremos la línea 5, para que se haga una división entera.
  - Y el tipo del valor devuelto, será `int` en lugar de `double`.
- Al ejecutar el programa como habitualmente (`./div`) se muestra:
  - `Excepción de coma flotante ('core' generado)`

```
01. #include <iostream>
02. using namespace std;
03.
04. int getDivision (int num, int den) {
05.   return num / den;
06. }
07.
08. int main() {
09.   int n1 = 7, n2 = 3;
10.   cout << getDivision (n1, n2);
11.   n1 = 3; n2 = 0;
12.   cout << getDivision (n1, n2);
13.   return 0;
14. }
```

```
(gdb) r
Starting program: div
[Depuracion de hilo usando libthread_db enabled]
Using host libthread_db library
    "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGFPE, Arithmetic exception.
0x000055555555519b in getDivision (num=3, den=0) at div.cpp:5
5    return num / den;
(gdb) where
#0  0x000055555555519b in getDivision (num=3, den=0) at div.cpp:5
#1  0x00005555555551f7 in main () at div.cpp:13
(gdb)
```

- Herramienta estandar *de facto* para generar documentación de C++.
  - También soporta otros muchos lenguajes: C, C#, PHP, Java, Python...
  - Aunque Java o Python usan normal/ sus *tools*: Javadoc y Pydoc.
- Con código debida/ anotado, Doxygen genera documentación formal.
  - HTML, para manual/doc. on-line; LaTeX, como manual de referencia.
  - Otros: RTF, PostScript, *hyperlinked* PDF, páginas `man` de UNIX.
- Permite extraer la estructura del código desde los ficheros fuente.
  - Y visualizar relaciones con diferentes diagramas (colab., herencia, etc.).

- SFML: Simple and Fast Multimedia Library. Desarrollo de juegos.
  - Doxygen doc: *https://github.com/SFML/SFML/tree/master/doc*
  - Relacionada: OGRE3D, Object-Oriented Graphics Rendering Engine.
- OpenCV: Open Source Computer Vision. Algoritmos de visión.
  - Doxygen doc: *https://github.com/opencv/opencv/tree/master/doc*
- Magnum Graphics: middleware gráfico para juegos y visualizar datos.
  - Doxygen doc: *https://github.com/mosra/magnum/tree/master/doc*
- OpenFoam: dinámica fluidos, reacciones químicas, turbulencias, etc.
  - Doc: *https://github.com/OpenFOAM/OpenFOAM-6/tree/master/doc*
- CERN's Root Framework: desarrollo del entorno raíz del CERN.
  https://github.com/root-project/root/tree/master/documentation/doxy
  - Relacionada: GslWrapper, clase wrapper para la GNU Scientific Library.

- Instalación de todos los *features* de Doxygen:
  - `sudo apt-get install doxygen doxygen2man doxygen-doc doxygen-doxyparse doxygen-gui doxygen-latex`
- Para poder visualizar los árboles de relaciones entre clases:
  - `sudo apt-get install graphviz`
- Comprueba tu instalación/versión: `doxygen --version`

- El comportamiento de Doxygen se rige por un fichero: `Doxyfile`.
- Para generar un fichero `Doxyfile`: `doxygen -g <fichero>`
  - Normalmente se ejecuta sin el último parámetro.
- Este fichero se puede modificar.
- Cada proyecto tiene su fichero `Doxyfile`.
- Para usar `Doxygen`: ir al directorio del proyecto y lanzarlo (`doxygen`).
  - Este busca los códigos fuente, los *parsea* y genera la documentación.
  - La documentación estará en la carpeta especificada.
    - E.g. línea 61: `OUTPUT_DIRECTORY = "doxygen-doc"`
  - Doc. HTML (`doxygen-doc/html`): `$firefox index.html`
  - Doc. LaTeX(`doxygen-doc/latex`): `$make && evince refman.pdf`

```
#Linea 35:
PROJECT_NAME           = "My first Doxygen test"
#Linea 47:
PROJECT_BRIEF          = "This is a simple project to use Doxygen"
#Linea 54:
PROJECT_LOGO           = "./MyIcon.ico"
#Linea 61:
OUTPUT_DIRECTORY       = "doxygen-doc"
#Linea 363:
DISTRIBUTE_GROUP_DOC = YES
#Linea 438:
EXTRACT_ALL            = YES
#Linea 444:
EXTRACT_PRIVATE        = YES
```

```
#Linea 450:
EXTRACT_PACKAGE        = YES
#Linea 456:
EXTRACT_STATIC         = YES
#Linea 481:
EXTRACT_ANON_NSPACES = YES
#Linea 759:
WARN_NO_PARAMDOC       = YES
#Linea 867:
RECURSIVE              = YES
#Linea 876:
EXCLUDE                = README.md
#Linea 998:
SOURCE_BROWSER         = YES
```

```
#Linea 1004:
INLINE_SOURCES        = YES
#Linea 1017:
REFERENCED_BY_RELATION = YES
#Linea 1023:
REFERENCES_RELATION   = YES
#Linea 1031:
REFERENCES_LINK_SOURCE = NO
#Linea 2283:
UML_LOOK              = YES
#Linea 2334:
CALL_GRAPH            = YES
#Linea 2346:
CALLER_GRAPH          = YES
```

Los comentarios de Doxygen comienzan por `/**` y terminan por `*/`:

```
/**
 *  This is my first example using Doxygen
 *
 *  The asterisks to the left could be omitted, but they help to
 *  get a nice comment.
 */
```

Los comentarios de cabecera suelen contener *keywords*:

```
/**
 *  @file  MyClass.h
 *  @brief This is my first example using Doxygen
 *  @author Julio Vega
 *  @date  2022-11-07
 ***********************************************/

class MyClass {
  // [...]
};
```

Lo ideal es poner los comentarios mayor/ en los archivos cabecera (.h).

```
/**
 * \brief Adds two numbers.
 *
 * This function takes two numbers, adds them, and then returns
     the result.
 *
 * \param x The first number to add.
 * \param y The second number to add.
 * \return The sum of the two numbers.
 */
int add(int x, int y) {
    return x + y;
}
```

9. Depuración con GDB y documentación con Doxygen

Julio Vega

julio.vega@urjc.es