# What is gdb?

gdb is a debugger: a program that can show what happens inside another program as the latter executes. The debugger allows you to execute your program line by line, make it stop at a specific point or when a certain condition is satisfied, check the value of an expression at some point during execution, etc. For a more complete discussion on the functions of gdb see chapter 6 of "Programming With Gnu Software" by Mike Loukides which is on reserve at SEL.

# Getting started

In the discussion that follows, `blue text` is what you type, `black text` is what comes up on the screen and *red text* is comments and explanations.

Download the files `info.h`, `info.cpp` and `driver.cpp` from the webpage. Modify the `main()` function in `driver.cpp` as follows:

```
int main () {
        EmployeeInfo *president, copy;

        president = new EmployeeInfo("John Doe", 100000);
        copy = *president;

        delete president;
        president = NULL;

        cout << president->get_salary() << endl;

        return 0;
}
```

Compile your code. Make sure you use the `-g` option to make the compiler generate debugging info:

```
[vdoufexi@tlab-10 ] g++ -g driver.cpp info.cpp
```

Run the executable:

```
[vdoufexi@tlab-10 ] ./a.out
```

Your program segfaults so it's time to start the debugger and find out why.

```
[vdoufexi@tlab-10 ] gdb
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu".
(gdb) file a.out This tells gdb that the executable is called a.out
Reading symbols from a.out...done.
(gdb)
```

# Basic commands

Before we go on, let's have a look at the commands you will use most often:

### run

This starts the program. If you haven't set any breakpoints (see below) the program will run until it terminates (normally or by crashing).

### backtrace

Shows your current location in the program as well as a stack trace showing how you got there. It is very useful when you want to trace the sequence of function calls up until a segfault. Its shorthand is `bt`.

### break

This sets up a breakpoint. Breakpoints allow you to stop execution temporarily so that you can examine the values of variables, functions, etc. at that point. You may have several breakpoints in a program and there are several ways to set one:

- `break` *function_name*
  sets a breakpoint at the beginning of the given function. Examples:

  ```
  break main
  break 'EmployeeInfo::print()'
  ```

- `break` *filename:line_number*
  sets a breakpoint at the specified line of the given file. Example:

  ```
  break info.cpp:10
  ```

### continue

If you type `continue` or `c` after having stopped at a breakpoint, execution resumes until either the program terminates or anothe breakpoint is reached.

### info breakpoints

This lists the current breakpoints. You may then remove a breakpoint by typing `delete` and the number of that breakpoint.

### next

This executes the next command. Its shorthand is `n`. See also `step`.

### step

This steps through the next command. The difference between `step` and `next` is that if the next command contains a function call, then typing `next` will execute the function without stepping through it, while typing `step` will take you to the first line of the function. The shorthand for `step` is `s`.

### print

This prints out the value of a variable or expression. The shorthand is `p`. Examples:

```
print num
print president->name
print arr[4]
```

### list

This shows you the code surrounding the current line.

### q

Quit gdb.

## A sample run

The following is a transcript of my debugging the code you downloaded and modified (after gdb has been started and a.out loaded). Let me know if you have any questions or if anything is unclear.

```
(gdb) file a.out
Reading symbols from a.out...done.
(gdb) break main
Breakpoint 1 at 0x80487f5:  file driver2.cpp, line 4.
(gdb) info breakpoints
Num Type         Disp Enb Address    What
1  breakpoint    keep y   0x080487f5 in main at driver2.cpp:4
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) run
Starting program:  /home/vdoufexi/a.out

Program received signal SIGSEGV, Segmentation fault.
0x08048c06 in EmployeeInfo::get_salary() (this=0x0) at info.cpp:105
105         return salary;
(gdb) bt
#0 0x08048c06 in EmployeeInfo::get_salary() (this=0x0) at info.cpp:105
#1 0x080488cf in main () at driver2.cpp:12
#2 0x42015574 in __libc_start_main () from /lib/tls/libc.so.6
Read this from bottom to top:  first, main() was called and from within main() we called get_salary()
and that's where the program crashed.  Note also that this=0x0 .  This immediately tells us
that the current object is NULL. The segfault obviously occured because we tried to derefence
this object.  But let's go on anyway...
(gdb) break main
Breakpoint 2 at 0x80487f5:  file driver2.cpp, line 4.
(gdb) run
The program being debugged has been started already.
Start it from the beginning?  (y or n) y
```

```
Starting program:  /home/vdoufexi/a.out

Breakpoint 2, main () at driver2.cpp:4 we reached the breakpoint
4         EmployeeInfo *president, copy; this is the next line to be executed
(gdb) n
6         president = new EmployeeInfo("John Doe", 100000);
(gdb) s s will take us inside the constructor
EmployeeInfo (this=0x804a2a8, thename=0x8048d34 "John Doe", thesalary=100000)
at info.cpp:24
24         name = new char[strlen(thename)+1];
(gdb) n
25         strcpy(name, thename);
(gdb) p name this is before the strcpy
$1 = 0x804a2b8 ""
(gdb) n
26         salary = thesalary;
(gdb) p name after the strcpy:
$2 = 0x804a2b8 "John Doe"
(gdb) n
27     }
(gdb) n
main () at driver2.cpp:7 we left the constructor and are back at main()
7         copy = *president; this is the next statement in line.  If we hit s, we'll step into
the copy constructor
(gdb) p president->name
$3 = 0x804a2b8 "John Doe"
(gdb) s
EmployeeInfo::operator=(EmployeeInfo const&) (this=0xbffff6b0, rhs=@0x804a2a8)
at info.cpp:56
56         if (this == &rhs)
(gdb) n
64         if (name)
(gdb) here I just hit Enter which repeated the last command, n
69         name = new char[strlen(rhs.name)+1];
(gdb)
70         strcpy(name, rhs.name);
(gdb)
71         salary = rhs.salary;
(gdb)
77         return *this;
(gdb) p this
$4 = (EmployeeInfo * const) 0xbffff6b0 0xbffff6b0 is the value of the 'this' pointer.
(gdb) p *this now, we'll get the contents at address 'this'
$5 = {name = 0x804a2c8 "John Doe", salary = 100000}
(gdb) n
85     }
(gdb) n
main () at driver2.cpp:9 back in main()
9         delete president;
(gdb) n
10         president = NULL;
(gdb) p president
$6 = (EmployeeInfo *) 0x804a2a8
```

```
(gdb) n
12          cout << president->get_salary() << endl;
(gdb) p president
$7 = (EmployeeInfo *) 0x0 president is NULL, which explains the segfault
(gdb) p president->salary
Cannot access memory at address 0x4 obviously...
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0x08048c06 in EmployeeInfo::get_salary() (this=0x0) at info.cpp:105
105          return salary;
(gdb) q
The program is running.  Exit anyway?  (y or n) y
[vdoufexi@tlab-10]
```