```
#include <iostream>
using namespace std;

int divint(int, int);
int main()
{
   int x = 5, y = 2;
   cout << divint(x, y);

   x =3; y = 0;
   cout << divint(x, y);

   return 0;
}

int divint(int a, int b)
{
   return a / b;
}
```

To enable debugging, the program must be compiled with the -g option.

```
$g++ -g crash.cc -o crash
Floating point exception (core dumped)
```

a *core* file (attention ulimit -c)

```
$gdb crash
# Gdb prints summary information and then the (gdb) prompt

(gdb) r
Program received signal SIGFPE, Arithmetic exception.
0x08048681 in divint(int, int) (a=3, b=0) at crash.cc:21
21        return a / b;

# 'r' runs the program inside the debugger
# In this case the program crashed and gdb prints out some
# relevant information.  In particular, it crashed trying
# to execute line 21 of crash.cc.  The function parameters
# 'a' and 'b' had values 3 and 0 respectively.

(gdb) l
# l is short for 'list'.  Useful for seeing the context of
# the crash, lists code lines near around 21 of crash.cc

(gdb) where
#0  0x08048681 in divint(int, int) (a=3, b=0) at crash.cc:21
#1  0x08048654 in main () at crash.cc:13
# Equivalent to 'bt' or backtrace.  Produces what is known
# as a 'stack trace'.  Read this as follows:  The crash occurred
# in the function divint at line 21 of crash.cc.  This, in turn,
# was called from the function main at line 13 of crash.cc

(gdb) up
# Move from the default level '0' of the stack trace up one level
# to level 1.
```

```
(gdb) list
# list now lists the code lines near line 13 of crash.cc

(gdb) p x
# print the value of the local (to main) variable x
```

the attempt to divide an integer by 0.

```
gdb crash core
```

non-initialized memory.

```
#include <iostream>
using namespace std;

void setint(int*, int);
int main()
{
   int a;
   setint(&a, 10);
   cout << a << endl;

   int* b;
   setint(b, 10);
   cout << *b << endl;

   return 0;
}

void setint(int* ip, int i)
{
   *ip = i;
}
```

```
$g++ -g crash2.cc -o crash2
segmentation fault (core dumped)
$ gdb crash2
(gdb) r
Starting program: crash2
10
10
Program received signal SIGSEGV, Segmentation fault.
0x4000b4d9 in _dl_fini () from /lib/ld-linux.so.2

(gdb) where
#0  0x4000b4d9 in _dl_fini () from /lib/ld-linux.so.2
#1  0x40132a12 in exit () from /lib/libc.so.6
#2  0x4011cdc6 in __libc_start_main () from /lib/libc.so.6

#3  0x080485f1 in _start ()
(gdb)
```

Unfortunately, the program will not crash in either of the user-defined functions, **main** or **setint,** so there is no useful trace or local variable information. In this case, it may be more useful to single-step through the program.

```
(gdb) b main
# Set a breakpoint at the beginning of the function main

(gdb) r
# Run the program, but break immediately due to the breakpoint.

(gdb) n
# n = next, runs one line of the program

(gdb) n
(gdb) s
setint(int*, int) (ip=0x400143e0, i=10) at crash2.C:20
# s = step, is like next, but it will step into functions.
# In this case the function stepped into is setint.

(gdb) p ip
$3 = (int *) 0x400143e0

(gdb) p *ip
1073827128
```

The value of *ip is the value of the integer pointed to by ip. In this case, it is an unusual value and is strong evidence that there is a problem. The problem in this case is that the pointer was never properly initialized, so it is pointing to some random area in memory (the address 0x40014e0). By pure luck, the process of assigning a value to *ip does not crash the program, but it creates some problem that crashes the program when it finishes.

Segfault example:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
 char *buf;

   buf = malloc(1<<31);

   fgets(buf, 1024, stdin);
   printf("%s\n", buf);

   return 1;
 }
```

(gdb) run

SIGSEGV : we tried to access an invalid memory address

backtrace

```
#0  0x4007fc13 in _IO_getline_info () from /lib/libc.so.6
#1  0x4007fb6c in _IO_getline () from /lib/libc.so.6
#2  0x4007ef51 in fgets () from /lib/libc.so.6
#3  0x80484b2 in main (argc=1, argv=0xbffffaf4) at segfault.c:10
#4  0x40037f5c in __libc_start_main () from /lib/libc.so.6
```

frame 3

print buf
$1 = 0x0
 -> NULL pointer

kill
break 8
run (it stops at the breakpoint)

print buf
$2 = 0xbffffaa8 "Иъяї#\177\003@t`\001@\001"
next
10 fgets(buf, 1024, stdin);
(gdb) print buf
$3 = 0x0
 -> NULL pointer

after the malloc, buf is NULL!

1<< 31 (1 right-shifted 31 x is 429497295 (4GB) malloc fail
1 << 9 is 1024

Infinite Loop

```
#include <stdio.h>
#include <ctype.h>

int main(int argc, char **argv)
{
  char c;

  c = fgetc(stdin);
  while(c != EOF){

    if(isalnum(c))
      printf("%c", c);
    else
      c = fgetc(stdin);
  }

  return 1;
}
```

gdb a.out
run
^c -> sigint
backtrace

#0 0x00007ffff7afcba0 in __write_nocancel () from /lib64/libc.so.6
#1 0x00007ffff7a872f3 in _IO_new_file_write () from /lib64/libc.so.6
#2 0x00007ffff7a88b0e in __GI__IO_do_write () from /lib64/libc.so.6

#3 0x00007ffff7a890cb in __GI__IO_file_overflow () from /lib64/libc.so.6
#4 0x00007ffff7a7f75e in putchar () from /lib64/libc.so.6
#5 0x000000000040067b in main (argc=1, argv=0x7fffffffddd8) at inf.c:12

frame in the write() of libc
frame 5 -> in the main

print c
next several time -> line 11 & 12 forever!

solution...

Valgrind : a memory mismanagement detector (memcheck and more)

- Use of uninitialised memory
- Reading/writing memory after it has been free'd
- Reading/writing off the end of malloc'd blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks -- where pointers to malloc'd blocks are lost forever
- Mismatched use of malloc/new/new [] vs free/delete/delete []
- Overlapping src and dst pointers in memcpy() and related functions
- Some misuses of the POSIX pthreads API

```
#include <stdio.h>

int main()
{
  char *p;

  // Allocation #1 of 19 bytes
  p = (char *) malloc(19);

  // Allocation #2 of 12 bytes
  p = (char *) malloc(12);
  free(p);

  // Allocation #3 of 16 bytes
  p = (char *) malloc(16);

  return 0;
}
```

```
valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 --track-fds=yes ./test
```

```
==15231== Memcheck, a memory error detector
==15231== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15231== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==15231== Command: ./test
==15231==
```

```
==15231==
==15231== FILE DESCRIPTORS: 3 open at exit.
==15231== Open file descriptor 2: /dev/pts/4
==15231==    <inherited from parent>
==15231==
==15231== Open file descriptor 1: /dev/pts/4
==15231==    <inherited from parent>
==15231==
==15231== Open file descriptor 0: /dev/pts/4
==15231==    <inherited from parent>
==15231==
==15231==
==15231== HEAP SUMMARY:
==15231==     in use at exit: 35 bytes in 2 blocks
==15231==   total heap usage: 3 allocs, 1 frees, 47 bytes allocated
==15231==
==15231== 16 bytes in 1 blocks are definitely lost in loss record 1 of 2
==15231==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==15231==    by 0x4005B6: main (testvalgrid.c:15)
==15231==
==15231== 19 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15231==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==15231==    by 0x40058E: main (testvalgrid.c:8)
==15231==
==15231== LEAK SUMMARY:
==15231==    definitely lost: 35 bytes in 2 blocks
==15231==    indirectly lost: 0 bytes in 0 blocks
==15231==      possibly lost: 0 bytes in 0 blocks
==15231==    still reachable: 0 bytes in 0 blocks
==15231==         suppressed: 0 bytes in 0 blocks
==15231==
==15231== For lists of detected and suppressed errors, rerun with: -s
==15231== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

[...]