

C++ Programming Debugger

Mostafa S. Ibrahim

Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher

PhD from Simon Fraser University - Canada

Bachelor / Msc from Cairo University - Egypt

Ex-(Software Engineer / ICPC World Finalist)



Software Bugs

- A software **bug** is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.
 - Invalid Array Indexing
 - Divide by zero
 - Undefined behaviours in a language
 - Wrong usage for uninitialized variables
 - Memory leaks / infinite loops
 - Reference a local variable returned from a function
 - Logic / Design bug / implementation bugs { if (n%2 == 1) lamOdd!
- It takes time to minimize your bugs
 - Coding guidelines, validating inputs, initialize always, modularity, testing, Careful copy-paste

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

Rick Osborne

Signals Interruption

- **Signals** are the interrupts that force an OS to stop its ongoing task and attend the task for which the interrupt has been sent. These interrupts can pause a service in any programs of an OS.
- Similarly, C++ also offers various signals which it can catch and process in a program
- Future [reading](#)

Signals	Operations
SIGINT	Produces receipt for an active signal
SIGTERM	Sends a termination request to the program
SIGBUS	Bus error which indicates access to an invalid address
SIGILL	Detects an illegal command
SIGALRM	This is used by alarm() function and indicates expiration of timer
SIGABRT	Termination of a program, abnormally
SIGSTOP	The signal cannot be blocked, handled and ignored and can stop a process
SIGSEGV	Invalid access to storage
SIGFPE	Overflow operations or mathematically incorrect operations like divide by zero.
SIGUSR1 SIGUSR2	User defined signals

Signals Interruption

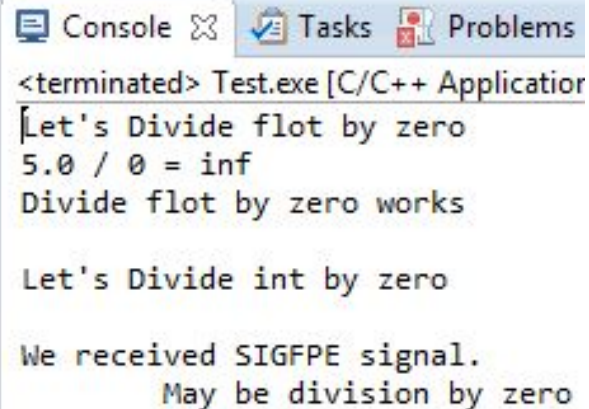
```
#include <csignal>
using namespace std;

void SIGFPE_handler(int a) {
    // a value is SIGFPE = 8. Use endl/flush
    cout << "\nWe received SIGFPE signal.\n"
         << "\tMay be division by zero" << endl;
}

int main() {
    signal(SIGFPE, SIGFPE_handler);

    cout << "Let's Divide flot by zero\n";
    cout << "5.0 / 0 = " << 5.0 / 0 << "\n";
    cout << "Divide flot by zero works\n";

    cout << "\nLet's Divide int by zero\n";
    cout << "5 / 0 = " << 5 / 0 << "\n";
    cout << "Divide integer by zero works\n";
}
```



The screenshot shows a C++ IDE with three tabs: Console, Tasks, and Problems. The Console tab is active, displaying the output of a program. The output shows the program attempting to divide a float by zero, which results in an infinity value. Then, it attempts to divide an integer by zero, which causes a SIGFPE signal to be sent to the process. The program then prints a message indicating it received the SIGFPE signal and that it may be a division by zero error.

```
<terminated> Test.exe [C/C++ Application]
Let's Divide flot by zero
5.0 / 0 = inf
Divide flot by zero works

Let's Divide int by zero

We received SIGFPE signal.
    May be division by zero
```

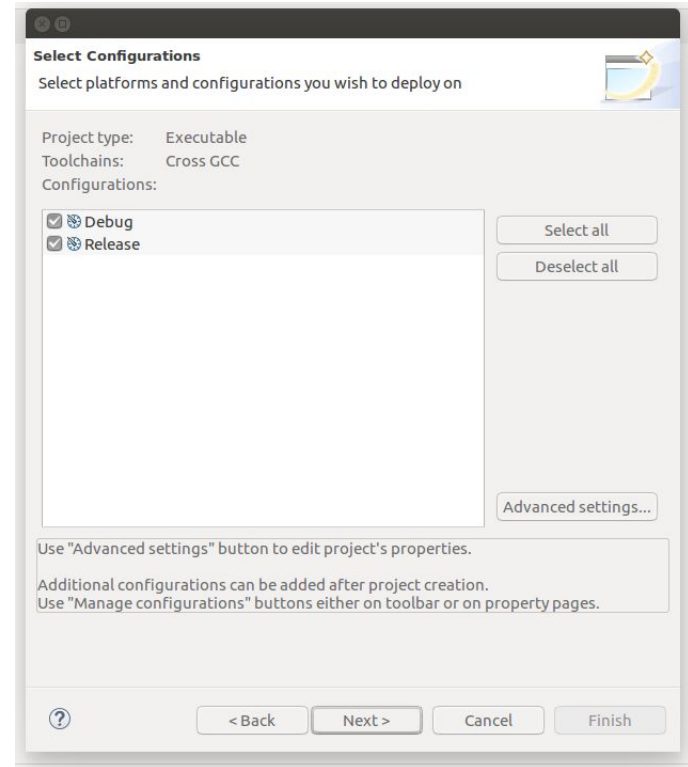
Debugging

- Finding the bug
- 2 Approaches
- **Printing approach**
 - Like you do all the time.
 - Print statements where you suspect errors
 - Seems naive, but still strong one. In many big projects or some remote sessions is used
 - In real projects we also use **loggers**: logging several kind of information
 - Later with system issues we read logs guess where is the problem
- **Debugger approach**
 - Trace your code line by line. Very powerful if can be used

Debug Symbols

- We learned compiler generate 2 symbols tables for linker later
- A **debug symbol** is a special kind of symbol that attaches **additional** information to the symbol table of an object file
- The **debugger** use **this info** to do deep line-by-line **tracing** for your code

Debug Mode vs Release Mode



Debug Mode vs Release Mode

- Release mode: we don't embed these symbols
 - Can't use Debugger
 - Used in production
- Debug mode: code is compiled with **debug symbols**
 - Debugger can be used
 - Generated files are **bigger** in size than release files
 - Debugger execution time is **bigger** than release time
 - Used for development and bugs inspection
- Behaviour
 - In some scenarios, the behaviour of them is NOT same, which is tricky!
 - Some illegal memory access may work in debug and fail in release
 - In debugging with double values, possible different precisions
 - We must test in release before production

Debuggers programs

- A debugger is a program, so several ones may exist
- GNU debugger is named: **gdb**
- Microsoft visual debugger is a nice one (at least vs old gdb versions)
- We can debug from IDE or from terminal
- We may debug local and remote codes

“Acquire knowledge and impart it to the people.”

“Seek knowledge from the Cradle to the Grave.”