

Debugging
(or where you'll spend 90% of your time)

The first step in effective debugging is
understanding the tools in your bag

Some tools...

Print statements

Logging frameworks (strategies)

Debuggers

Some tools...

Print statements

Logging frameworks (strategies)

Debuggers ← **Today**

Print statements
There exist many breeds...

Print statements

There exist many breeds...

```
printf("here");
```

Print statements

There exist many breeds...

```
printf(“here”);
```

```
printf(“here2”);
```

Print statements

There exist many breeds...

```
printf(“here”);
```

```
printf(“here2”);
```

```
printf(“Shouldn’t be here.....”);
```


Print statements

There exist many breeds...

```
printf("here");
```

```
printf("here2");
```

```
printf("Shouldn't be here.....");
```

```
printf("WTF!!!!");
```

Print statements

There exist many breeds...

Program Flow:

```
printf("here");
```

```
printf("here2");
```

```
printf("Shouldn't be here.....");
```

```
printf("WTF!!!!");
```

Print statements

There exist many breeds...

Program Flow:

```
printf("here");
```

```
printf("here2");
```

```
printf("Shouldn't be here.....");
```

```
printf("WTF!!!!");
```

Value checking:

```
printf("value = %d", some_value);
```

extremely important for numerical errors

Logging

Printing on steroids

```
Logger Log;

// ...

Matrix** Invert(Matrix** myMatrix) {
    Log("Invert()", Log::Arguments, myMatrix);

    Matrix** InvMatrix;
    //... Code to invert matrix ... //

    Log("Invert()", Log::Return, InvMatrix);
    return InvMatrix;
}
```

Logging

Printing on steroids

```
Logger Log;

// ...

Matrix** Invert(Matrix** myMatrix) {
    Log("Invert()", Log::Arguments, myMatrix);

    Matrix** InvMatrix;
    //... Code to invert matrix ... //

    Log("Invert()", Log::Return, InvMatrix);
    return InvMatrix;
}
```

Some open-source logging frameworks I found:

<https://github.com/easylogging/easyloggingpp>

<http://www.drdobbs.com/cpp/a-lightweight-logger-for-c/240147505>

<http://www.codeproject.com/Articles/584794/Simple-logger-for-Cplusplus>

<https://github.com/gabime/spdlog> (fast, header only)

Debuggers

Programs that assist in the detection of errors
in other programs

Debuggers

Programs that assist in the detection of errors
in other programs

All platforms

<https://www.gnu.org/software/gdb/>

Windows

<https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx>

Terminal based

```
passpointer.c
1  #include <iostream>
2  #include <cstdlib>
3  #include <string.h>
4
5  using namespace std;
6
7  //-----
8  void PrintArray(int* array, size_t size) {
9      for(int i=0; i<size; i++) {
10         cout << array[i] << " ";
11     } cout << endl;
12 }
13 //-----
14 void AllocateAndSet(int* array, int val, size_t size) {
15     array = (int*)malloc(size*sizeof(int));
16     for (int i=0;i<size;i++) {
17         array[i] = val;
18     }
19 }
20 //-----
21
22 //-----
23
24 int main() {
25     int set_value = 10;
26     size_t size = 5;
27
28     int* arrayA = (int*) malloc(size*sizeof(int));
29     for (int i=0;i<size;i++) {
30         arrayA[i]=set_value;
31     }
32     PrintArray(arrayA,size);
33 }
```

(gdb)

GDB with a graphic front end

Chris Sullivan

Open program (compiled with -g)

The screenshot shows the GDB GUI with the following components:

- Source List:** A list of source files, currently showing `passpointer.c`.
- Source Window:** Displays the source code of `passpointer.c`. A red dot on line 24 indicates a breakpoint. Annotations with arrows point to specific icons in the toolbar:
 - Run until return (blue arrow)
 - Step over a single line of source code (blue arrow)
 - Step into function (brown arrow)
 - Continue execution (purple arrow)
 - Pause execution (orange arrow)
 - Restart program (blue arrow)
 - Restart program (green arrow)
- Watch Window:** A panel for watching variables. It contains the text:
 - Watch:** Values of variables in current scope are displayed
 - Registers:** View of special memory locations that the cpu uses to pass around information
- GDB Terminal:** A terminal window showing GDB commands and output:

```
(gdb) file /projects/ceclub/sullivan/cpp_workshop/debugging/production/passpointer
Reading symbols from /projects/ceclub/sullivan/cpp_workshop/debugging/production/passpointer...done.
(gdb) break /projects/ceclub/sullivan/cpp_workshop/debugging/production/passpointer.c:24
Breakpoint 1 at 0x4009d0: file passpointer.c, line 24.
(gdb)
```
- Command Window:** A text area for entering GDB commands.
- Bottom Panel:** A row of tabs for different views: GDB Terminal, Stack, Breakpoints, Source List, Thread, Memory, and Snapshot.

Most operations in this application call GDB functions which are written in text here. One can also enter gdb commands manually using the embedded terminal:

Stack: View of the history of all calls (how deep into the program you are)

Breakpoints: List of all applied breakpoints

Memory: A view of the application memory

Snapshot: Save application state to resume later