C++ Toolchain
Justin M. Turney
## Linux Distros

<table>
<thead>
<tr>
<th>Distro</th>
<th>Released</th>
<th>Compiler</th>
<th>EOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL 7</td>
<td>June 2014</td>
<td>gcc 4.8</td>
<td>June 2024</td>
</tr>
<tr>
<td>RHEL 7.4</td>
<td>August 2017</td>
<td>gcc 4.8</td>
<td>June 2024</td>
</tr>
<tr>
<td>SLES 11</td>
<td>March 2009</td>
<td>gcc 4.3</td>
<td>March 2022</td>
</tr>
<tr>
<td>SLES 12</td>
<td>October 2014</td>
<td>gcc 4.8</td>
<td>October 2027</td>
</tr>
<tr>
<td>Ubuntu 14.04 LTS</td>
<td>April 2015</td>
<td>gcc 4.8/clang 3.4</td>
<td>April 2019</td>
</tr>
<tr>
<td>Ubuntu 16.04 LTS</td>
<td>April 2016</td>
<td>gcc 5/clang 3.8</td>
<td>April 2021</td>
</tr>
<tr>
<td>Release</td>
<td>Release Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.8.0 / 4.8.5</td>
<td>March 2013 / June 2015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>December 2015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.4*</td>
<td>July 2017</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.2*</td>
<td>August 2017</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Officially supported versions
## LLVM Releases

<table>
<thead>
<tr>
<th>Release</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4</td>
<td>January 2014</td>
</tr>
<tr>
<td>3.8</td>
<td>March 2016</td>
</tr>
<tr>
<td>5.0</td>
<td>September 2017</td>
</tr>
</tbody>
</table>
Release Cycles

• 6 months to 1 year for a C++ toolchain release.
• 6 months to N-years for a distro release
• Unknown amount of time for someone to then install distro on systems.
What’s the alternative?

• Compile the toolchain on your own…
What’s the alternative?

• LLVM/Clang is very easy to compile.

• https://clang.llvm.org/get_started.html
Address Sanitizer (asan)

Asan is a memory error detector for C/C++.

1. Use after free (dangling pointer dereference)*
2. Heap buffer overflow*
3. Stack buffer overflow
4. Global buffer overflow
5. Use after return
6. Use after scope
7. Initialization order bugs
8. Memory leaks* (on Linux)

* Only errors Valgrind/Memcheck can detect.
Examples
Example 1

```c++
// To compile: clang++ -g -fsanitize=address asan1.cpp -o asan1
int main(int argc, char **argv)
{
    int *array = new int[100];
    delete[] array;
    return array[argc]; // BOOM
}
```
Example 2

```cpp
// To compile: clang++ -g -fsanitize=address asan2.cpp -o asan2
int main(int argc, char **argv)
{
    int *array = new int[100];
    array[0] = 0;
    int res = array[argc + 100]; //BOOM
    delete[] array;
    return res;
}
```
Example 3

// To compile: clang++ -g -fsanitize=address asan3.cpp -o asan3
int main(int argc, char **argv)
{
    int stack_array[100];
    stack_array[1] = 0;
    return stack_array[argc + 100]; //BOOM
}
Example 4

// To compile: clang++ -g -fsanitize=address asan4.cpp -o asan4
int global_array[100] = {-1};

int main(int argc, char **argv)
{
    return global_array[argc + 100]; // BOOM
}
// To compile: clang++ -fsanitize=leak -g asan5.cpp -o asan5
#include <stdlib.h>

void *p;

int main() {
    p = malloc(7);
    p = 0; // The memory is leaked here.
    return 0;
}
• Learn C++ through conference videos on YouTube.
  • C++Con
  • PacifiC++
  • LLVM