

C++ Workshop

Jack Leightcap

`{nuieeeofficers,wirelessclub}@gmail.com`

Spring 2021 – October 4, 2021

Background: Workshop Structure

- ▶ the C++ you see in classes is very different than the C++ you might see on co-op
- ▶ compare/contrast with 2 examples you may have seen in embedded design
- ▶ some tools that can help writing C++ programs

general notes:

- ▶ ask questions at any time!
- ▶ reach out with any lingering questions

Background: C++

what even is C++?

- ▶ *compiled*: a *compiler* (g++, clang) translates human-readable text-files into machine code
- ▶ *high-level*: use C or assembly if you want more control
- ▶ *old*: new features have been added regularly since 1985

“There are only two kinds of languages: those that people [complain] about and those that nobody uses.” – Bjarne Stroustrup, `comp.lang.c++`

“Within C++, there is a much smaller and cleaner language struggling to get out” – Bjarne Stroustrup, “The Design and Evolutions of C++” [1994]

Example: Linked Lists – Memory Layout

What does a linked list actually *look like*? Memory is just a sequence of ‘words’, each with an ‘address’:

```
+---+---+---+---+---+
|   |   |   |   |   | ...
+---+---+---+---+---+
  1   2   3   4   5
```

a *linked list* containing {1, 2, 3} might look like:

```
+---+---+---+---+---+---+---+
| 1 | 4 |   | 2 | 6 | 3 | 0 |
+---+---+---+---+---+---+
  1   2   3   4   5   6   7
 \_____/ \_____/ \_____/
node 1      node 2 node 3
```

Example: Linked Lists – C++, Old-Style

```
class node {  
    public:  
        int val;  
        node *next;  
};
```

let's use this...

```
node * n1 = new node(); // memory for 3 nodes...  
node * n2 = new node();  
node * n3 = new node();  
n1->val = 1; // set up the values...  
n1->next = n2;  
n2->val = 2;  
n2->next = n3;  
n3->val = 3;  
n3->next = nullptr; // fin
```

Example: Linked Lists – C++, Standard Library

```
#include <forward_list>

using std::forward_list;

int main(void) {
    forward_list<int> list;
    list.assign({1, 2, 3});

    return 0;
}
```

- ▶ concise, less error-prone
- ▶ don't need to worry about the internals of the memory representation
- ▶ want a doubly-linked list? drop in `list` in place of `forward_list`.

Example: Memory – Manual

```
void function(void) {  
    node * n = new node();  
    // do some stuff...  
    return;  
  
    // oops, we forgot to delete n.  
    // memory leak!  
}
```

this new and delete stuff is annoying. doesn't this seem like something that the compiler should be able to figure out for us?

Example: Memory – Automated

```
void function(void) {  
    unique_ptr<node> n = new node();  
    // do some stuff...  
    return;  
  
    // at then end we are leaving this *scope*.  
    // the compiler knows to delete n.  
}
```


Tools: Linters (clang-check)

a 'linter' looks at the text of your program (i.e. doesn't actually *run* it), and flags potential issues

- ▶ syntactic issues: forgot a semicolon, etc.
- ▶ semantic issues: unfreed memory, types of arguments, etc.

Syntax

```
std::cout << fib(22) << std::endl
```

```
// "expected ';' after expression line 2"
```

Semantics

```
int * a = new int;  
*a = 2;
```

```
// "Potential leak of memory pointed to by 'a'"
```

Tools: Formatter (clang-format)

- ▶ re-formats your program for consistency
- ▶ helpful to agree on conventions when working with a team

some inconsistent code:

```
int ii=0;
for(ii=0; ii<    10; ii++)
{
    if(ii%2 ==1) std::cout << ii; }

```

using clang-format:

```
int ii = 0;
for (ii = 0; ii < 10; ii++) {
    if (ii % 2 == 1) {
        std::cout << ii;
    }
}

```

Tools: Building

C++ is a *specification*; there are multiple compilers that *implement* that specification: g++, clang++, etc.

compilers have a lot of options to help you!

- ▶ *warnings*: diagnostics from the compiler itself
 - ▶ -Wall, -Wextra, -Werror, -pedantic, -Wno-*, etc.
- ▶ *optimization*: what are you optimizing for?
 - ▶ -pipe, -DNDEBUG, -O2, -Os, etc.
- ▶ *standards*: are features available?
 - ▶ -std=c++13

Tools: Debugging

Valgrind

hooks into executing program and notes memory usage. can be used more generally as a profiler.

```
$ valgrind \  
    --leak-check=full \  
    --show-leak-kinds=all \  
    --track-origins=yes \  
    --trace-children=yes \  
    --show-reachable=no \  
    [your compiled program]
```

GDB

interactive debugger. can be esoteric to use, but extremely useful for runtime debugging.