C++ Workshop

Spring 2021

Jack Leightcap[1,2]

[1]IEEE – nuieeeofficers@gmail.com

[2]Wireless Club – nuwirelessclub@gmail.com

March 15, 2021

Background, Workshop Structure

- The C++ you see in classes is very different than the C++ you might see on co-op
- Compare both with 3 examples you might've seen in Embedded Design
- Some tools that can help with writing C++ programs

General notes:

- Ask questions at any time!
- Reach out to me on Slack with any lingering questions

C++ versus C

C	C++
. . . well, C	old: "C with classes"
	new: distinct language, roots in C
control, specificity	faster development
"close to the machine"	high-level

"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]

Linked Lists: C with Classes Style

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

int main(void) {
  node *n1 = new node();
  node *n2 = new node();
  n1->val = 2;
  n1->next = n2;
  n2->val = 3;
  n2->next = NULL;
  return 0;
}
```

- Replace a few things and this is a valid C program
- Fittingly with a memory leak
- Obvious here, not so much with more complexity

Linked Lists: Modern-ish C++

```cpp
#include <forward_list>

using std::forward_list;
int main(void) {
  forward_list<int> list;
  list.assign({2, 3});
  return 0;
}
```

- Concise, less error-prone
- Don't need to worry about internals of a linked list, just the interface
- Want a doubly linked list? Just drop in `list` in place of `forward_list`

Memory: C with Classes Style, `new` and `delete`

```
void PointersSuck() {
  Song *pSong =
    new Song("Stacy's Mom",
             "Fountains of Wayne");

  // use pSong...
  // don't lose track of it...

  // many lines, files away:
  delete pSong;
}
```

- *Explicit* memory management
- `new` pair `delete` $\leftrightarrow$ `malloc` pair `free`

Memory: Modern-ish C++, Smart Pointers

```
#include <memory>
using std::unique_ptr;
void SmartPointer() {
  unique_ptr<Song> pSong(
    new Song("Stacy's Mom",
             "Fountains of Wayne")
  );

  // go crazy!

  // pSong is automatically deleted
}
```

- *Automatic* memory management
- The language figures out where to free memory
- Very simple form of *Garbage Collection*

Sorting: C with Classes Style

```
void ssort(int a[], int n) {
  int ii, jj, min_idx;
  for (ii=0; ii<n-1; ii++) {
    min_idx = i;
        for (jj=ii+1; jj<n; jj++)
        if ([jj]<a[min_idx]
          min_idx = j;
        swap(&a[min_idx], &a[ii]);
  }
}
```

- Implement a sorting algorithm I guess?
- Lots of control, can very carefully pick algorithm

Sorting: Modern-ish C++

```cpp
#include <algorithm>

// STANDARD SORTING
std::sort(a.begin(), a.end());

// SORTING IN REVERSE ORDER
std::sort(a.begin(), a.end(),
  [](auto a1, auto a2) {
     return (a1 > a2)
  }
);

// SORTING ABSOLUTE VALUE
using std::abs;
std::sort(a.begin(), a.end(),
  [](auto a1, auto a2) {
    return (abs(a1) < abs(a2));
  }
);
```

- Less control over the actual algorithm used
- But super flexible!
- Trust experts' implementation

Tools: Linters (`clang-check`)

- Looks at your program (doesn't run it), and flags potential issues
- Syntact issues: forgot a semicolon, etc.
- Symantic issues: unfreed memory, types of arguments, etc.

Lint: Syntax

```
    1 int main(int ac, char* av[]) {
!!17     std::cout << fib(22) << std::endl
!!   1     return 0;
    2 }
~
~
~
fib.cc
expected ';' after expression
```

Lint: Symantic

```
    4
    3 int main(int ac, char* av[]) {
~   2     int *a = new int;
+   1     *a = 2;
!?19      return 0;
    1 }
~
~
fib.cc
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

Formatter: Example

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

Figure: Inconsistent formatting

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

Figure: Automatically consistent

Tools: Building

Compiler choices:

- Which compiler, `g++`, `clang++`, etc.
- Warnings: `-Wall`, `-Wextra`, `-Werror`, `-pedantic`, `-Wno-*`, etc.
- Optimization: `-pipe`, `-DNDEBUG`, `-O2`, `-Os`, etc.
- Standards: `-std=c++20`
- Debugging: `-ggdb`

Example compilation,

```
# All warnings, with debugging information
$ clang++ -Wall -Wextra -Werror -pedantic -ggdb file.cpp
```