MODERN C++ TO IMPRESS YOUR EMBEDDED DEV FRIENDS

Steve Bush - Research Fellow

Digital Innovation - Smart Products

The Procter & Gamble Company

ABSTRACT

C++ is often talked about in terms of what cannot or should not be done in the context of embedded systems. In contrast, this talk is about some of the things that modern C++ idioms allow us to do better or more expressively than comparable operations in C. We will cover several patterns that allow an embedded developer to express intent clearly, ease maintenance, and encourage re-use while avoiding overheads that are costly in resource-constrained systems. Examples include initialization structures that are self-describing and IDE-friendly, small code implementations that make available sections of the C++ Standard Library for embedded use, and tools for understanding and controlling our use of limited memory resources.

ABOUT ME

- Research Fellow at Procter & Gamble
- P&G is a global consumer goods manufacturer
- Embedded product developer, C++ evangelist


Gain
Downy
Bounce
Dash
Tide
Gain
Tide
FAIRY
ARIEL
Pampers
easy ups
NINJAMAS
Pampers
Pampers
Pampers
Luvs
Charmin
Ultra Soft
Bounty
The Quicker Picker Upper
Puffs
Plus Lotion
Swiffer
WETJET
Mr. Clean
DAWN
FAIRY
Swiffer
HEAVY DUTY
Cascade
FAIRY
CLEAN FREAK
DAWN
FAIRY
MAX
Oral-B
iO
clic
Crest
ARC
Fixodent
Oral-B
Glide
P&G


CES®


Introducing
AIRIA


OPTE


Oral-B®

OBJECTIVE

Making embedded code easier (and more pleasant) to write and maintain

AGENDA

- GPIO configuration
- Compile-Time Lookup Table Generation
- Developer-Friendly Numeric Constructs
- Lean stream-based IO
- Heap memory management for allocate-once
- Use `std::chrono` like a boss
- A `std::random` topic

COMPANION REPOSITORY

(https://github.com/sgbush/cppcon2022

CONFIGURE GPIO DECLARATIVELY

DECLARATIVE GPIO

The Old Way

```
GPIOInit_t config;
config.PinNumber = 1;
config.Mode = GPIO_MODE_PUSH_PULL;
config.Pull = GPIO_PULLUPDOWN_NONE;
config.Speed = GPIO_SPEED_MEDIUM;
config.AlternateFunc = GPIO_AF_6;

extern GPIO_Type* GPIOA;

GPIO_Configure(GPIOA, &config);
GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);

config.PinNumber = 4;
config.AlternateFunc = GPIO_AF_2;

GPIO_Configure(GPIOA, &config);
```

DECLARATIVE GPIO

The Old Way

```
GPIOInit_t config;
config.PinNumber = 1;
config.Mode = GPIO_MODE_PUSH_PULL;
config.Pull = GPIO_PULLUPDOWN_NONE;
config.Speed = GPIO_SPEED_MEDIUM;
config.AlternateFunc = GPIO_AF_6;

extern GPIO_Type* GPIOA;

GPIO_Configure(GPIOA, &config);
GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);

config.PinNumber = 4;
config.AlternateFunc = GPIO_AF_2;

GPIO_Configure(GPIOA, &config);
```

DECLARATIVE GPIO

The Old Way

```
 2 config.PinNumber = 1;
 3 config.Mode = GPIO_MODE_PUSH_PULL;
 4 config.Pull = GPIO_PULLUPDOWN_NONE;
 5 config.Speed = GPIO_SPEED_MEDIUM;
 6 config.AlternateFunc = GPIO_AF_6;
 7
 8 extern GPIO_Type* GPIOA;
 9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
17 GPIO_WritePin(GPIOA, 4, GPIO_LOGIC_LOW);
```

DECLARATIVE GPIO

The Old Way

```
2  config.PinNumber = 1;
3  config.Mode = GPIO_MODE_PUSH_PULL;
4  config.Pull = GPIO_PULLUPDOWN_NONE;
5  config.Speed = GPIO_SPEED_MEDIUM;
6  config.AlternateFunc = GPIO_AF_6;
7
8  extern GPIO_Type* GPIOA;
9
10 GPIO_Configure(GPIOA, &config);
11 GPIO_WritePin(GPIOA, 1, GPIO_LOGIC_HIGH);
12
13 config.PinNumber = 4;
14 config.AlternateFunc = GPIO_AF_2;
15
16 GPIO_Configure(GPIOA, &config);
17 GPIO_WritePin(GPIOA, 4, GPIO_LOGIC_LOW);
```

DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
1  using GPIODEF = \
2  struct IODefStuct {
3      GPIO_TypeDef* GPIO;
4      uint32_t PinNumber;
5      enum IOFUNCTION:uint32_t { INPUT = 0, OUTPUT = 1, ALT = 2, ANALOG = 3 } Fu
6      uint32_t AltFunc;
7      enum IOTYPE:uint32_t { NORMAL = 0, OPENDRAIN = 1 } Type;
8      enum IOSPEED:uint32_t { LOW = 0, MEDIUM = 1, HIGH = 2, VERYHIGH = 3 } Spee
9      enum IOPULL:uint32_t { NONE = 0, PULLUP = 1, PULLDOWN = 2 } Bias;
10     enum IOSTATE:uint32_t { LOGIC_LOW, LOGIC_HIGH, DONT_CARE } InitialState;
11 };
12
13 static const std::array<GPIODEF, 2> gpiodefs = {{
14     // green LED
15     {GPIOB, 0,
16     GPIODEF::IOFUNCTION::OUTPUT, 1
```

DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
 6      uint32_t AltFunc;
 7      enum IOTYPE:uint32_t { NORMAL = 0, OPENDRAIN = 1 } Type;
 8      enum IOSPEED:uint32_t { LOW = 0, MEDIUM = 1, HIGH = 2, VERYHIGH = 3 } Spee
 9      enum IOPULL:uint32_t { NONE = 0, PULLUP = 1, PULLDOWN = 2 } Bias;
10      enum IOSTATE:uint32_t { LOGIC_LOW, LOGIC_HIGH, DONT_CARE } InitialState;
11  };
12
13  static const std::array<GPIODEF, 2> gpiodefs = {{
14      // green LED
15      {GPIOB, 0,
16      GPIODEF::IOFUNCTION::OUTPUT, 1,
17      GPIODEF::IOTYPE::NORMAL,
18      GPIODEF::IOSPEED::LOW,
19      GPIODEF::IOPULL::NONE,
20      GPIODEF::IOSTATE::LOGIC_LOW},
```

DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
10      enum IOSTATE:uint32_t { LOGIC_LOW, LOGIC_HIGH, DONT_CARE } initialState;
11  };
12
13  static const std::array<GPIODEF, 2> gpiodefs = {{
14      // green LED
15      {GPIOB, 0,
16      GPIODEF::IOFUNCTION::OUTPUT, 1,
17      GPIODEF::IOTYPE::NORMAL,
18      GPIODEF::IOSPEED::LOW,
19      GPIODEF::IOPULL::NONE,
20      GPIODEF::IOSTATE::LOGIC_LOW},
21      // red LED
22      {GPIOB, 2,
23      GPIODEF::IOFUNCTION::OUTPUT, 1,
24      GPIODEF::IOTYPE::NORMAL,
25      GPIODEF::IOSPEED::LOW
```

DECLARATIVE GPIO

Better Way: Strong types in an iterable container

```
13 static const std::array<GPIODEF, 2> gpiodefs = {{
14     // green LED
15     {GPIOB, 0,
16     GPIODEF::IOFUNCTION::OUTPUT, 1,
17     GPIODEF::IOTYPE::NORMAL,
18     GPIODEF::IOSPEED::LOW,

19     GPIODEF::IOPULL::NONE,
20     GPIODEF::IOSTATE::LOGIC_LOW},
21     // red LED
22     {GPIOB, 2,
23     GPIODEF::IOFUNCTION::OUTPUT, 1,
24     GPIODEF::IOTYPE::NORMAL,
25     GPIODEF::IOSPEED::LOW,
26     GPIODEF::IOPULL::NONE,
27     GPIODEF::IOSTATE::LOGIC_LOW},
```

DECLARATIVE GPIO

- Weak enums are used because they convert implicitly to their underlying type
  - *But* they are constrained to tight scope to minimize errors
- Syntax is expressive - and easy to maintain
- Easy to consolidate an entire project's IO configuration in one place

DECLARATIVE GPIO

Configuring and Using IO Definitions

- Preferred: Configure all in one go
- Not Preferred: Passing around IO Defs to delegates

DECLARATIVE GPIO

Configuring - write one hardware-specific function

```
bool Configure(const GPIODEF& def)
{
    // ... chip-specific register settings ...
}
```

DECLARATIVE GPIO

Configuring: the iterators way

```
template<typename Iter>
bool Configure(Iter begin, Iter end)
{
    while ( begin != end ) Configure(begin++);
}
```

DECLARATIVE GPIO

Configuring: the ranges way

```
template<typename T> requires std::ranges::range<T>
bool Configure(const T& definitions)
{
    for(auto& def : definitions) Configure(def);
}
```

DECLARATIVE GPIO

Passing GPIO references around

```
struct SPIBus
{
    SPIBus(SPI_TypeDef* peripheral) {  }
};

struct SPIProtocol
{
    enum class SPIMode { Mode1, Mode2, Mode3, Mode4 };
    SPIProtocol(SPIMode mode, uint32_t speed) {  }
};

struct SPIConnection
{
    SPIConnection(const SPIBus& bus,
                  const SPIProtocol& conn,
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
1  class GPIOAssertFunctor
2  {
3      const GPIODEF& mGPIO;
4
5      public:
6      constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
7      void operator()(bool enable) const
8      {
9          if ( enable )
10         {
11             mGPIO.GPIO->ODR |= ( 0b1 << mGPIO.PinNumber );
12         }
13         else
14         {
15             mGPIO.GPIO->ODR &= ~( 0b1 << mGPIO.PinNumber );
16         }
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
1  class GPIOAssertFunctor
2  {
3      const GPIODEF& mGPIO;
4
5      public:
6      constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
7      void operator()(bool enable) const
8      {
9          if ( enable )
10         {
11             mGPIO.GPIO->ODR |= ( 0b1 << mGPIO.PinNumber );
12         }
13         else
14         {
15             mGPIO.GPIO->ODR &= ~( 0b1 << mGPIO.PinNumber );
16         }
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
5       public:
6       constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
7       void operator()(bool enable) const
8       {
9           if ( enable )
10          {
11              mGPIO.GPIO->ODR |= ( 0b1 << mGPIO.PinNumber );
12          }
13          else
14          {
15              mGPIO.GPIO->ODR &= ~( 0b1 << mGPIO.PinNumber );
16          }
17      }
18  };
19
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
20
21 struct SPIConnection
22 {
23     const GPIOAssertFunctor& mEnableFunction;
24
25     SPIConnection(const SPIBus& bus,
26                   const SPIProtocol& conn,
27                   const GPIOAssertFunctor& enable)
28           : mEnableFunction(enable) {  }
29
30     bool ReadWrite(std::span<char> outdata,
31                    std::span<char> indata,
32                    size_t length)
33     {
34         mEnableFunction(true);
35         // write to the bus....
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
19
20
21 struct SPIConnection
22 {
23     const GPIOAssertFunctor& mEnableFunction;
24
25     SPIConnection(const SPIBus& bus,
26                   const SPIProtocol& conn,
27                   const GPIOAssertFunctor& enable)
28             : mEnableFunction(enable) {  }
29
30     bool ReadWrite(std::span<char> outdata,
31                    std::span<char> indata,
32                    size_t length)
33     {
34         mEnableFunction(true);
```

DECLARATIVE GPIO

Passing GPIO references around - better

```
23      const GPIOAssertFunctor& mEnableFunction;
24
25      SPIConnection(const SPIBus& bus,
26                    const SPIProtocol& conn,
27                    const GPIOAssertFunctor& enable)
28              : mEnableFunction(enable) {  }
29
30      bool ReadWrite(std::span<char> outdata,
31                     std::span<char> indata,
32                     size_t length)
33      {
34          mEnableFunction(true);
35          // write to the bus....
36          mEnableFunction(false);
37      }
38  };
```

DECLARATIVE GPIO

```
1  class AssertType {};
2  class AssertTypeLogicHigh
3      : public AssertType { static constexpr bool ValueWhenAsserted = true; };
4  class AssertTypeLogicLow
5      : public AssertType { static constexpr bool ValueWhenAsserted = false; };
6
7  template<typename Assert> requires std::derived_from<Assert,AssertType>
8  class GPIOAssertFunctor
9  {
10     const GPIODEF& mGPIO;
11     public:
12     constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
13     void operator()(bool enable)
14     {
15         if ( enable == Assert::ValueWhenAsserted )
16         {
```

DECLARATIVE GPIO

```
1  class AssertType {};
2  class AssertTypeLogicHigh
3      : public AssertType { static constexpr bool ValueWhenAsserted = true; };
4  class AssertTypeLogicLow
5      : public AssertType { static constexpr bool ValueWhenAsserted = false; };
6
7  template<typename Assert> requires std::derived_from<Assert,AssertType>
8  class GPIOAssertFunctor
9  {
10     const GPIODEF& mGPIO;
11     public:
12     constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
13     void operator()(bool enable)
14     {
15         if ( enable == Assert::ValueWhenAsserted )
16         {
```

DECLARATIVE GPIO

```
 8 class GPIOAssertFunctor
 9 {
10     const GPIODEF& mGPIO;
11     public:
12     constexpr GPIOAssertFunctor(const GPIODEF& io) : mGPIO(io) {}
13     void operator()(bool enable)
14     {
15         if ( enable == Assert::ValueWhenAsserted )
16         {
17             mGPIO.GPIO->ODR |= ( 0b1 << mGPIO.PinNumber );
18         }
19         else
20         {
21             mGPIO.GPIO->ODR &= ~( 0b1 << mGPIO.PinNumber );
22         }
23     }
```

DECLARATIVE GPIO

```
static const std::array<GPIODEF,12> IOPins = { /* ... */ };
auto bus = SPIBus(SPI1);
auto protocol = SPIProtocol(SPIProtocol::SPIMode::Mode1, 1'000'000);
auto chipselect = GPIOAssertFunctor<AssertTypeLogicLow>(IOPins[4]);
auto connection = SPIConnection(bus, protocol, chipselect);
```

DECLARATIVE GPIO

- Centralize your IO definitions
- Abstract IO functions - with little or no cost

CREATE LOOKUP TABLES USING COMPILE-TIME EXPRESSIONS

COMPILER-DRIVEN LOOKUP TABLE GENERATION

The Old Way

```
const float TemperatureFromThermistor[] = { 25.0, 26.2, 27.5, ... };
```

- Generate via spreadsheet and copy and paste
- Generate via script and copy and paste
- Generate via script and incorporate into build system

COMPILER-DRIVEN LOOKUP TABLE GENERATION

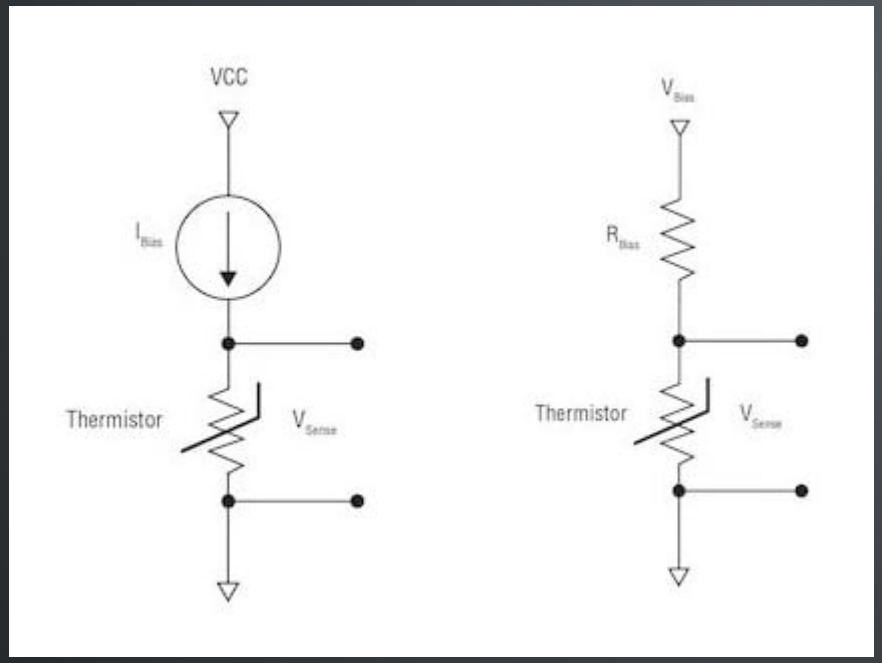
Some considerations

- Constants in embedded system must be carefully crafted to end up (usually) in the `.rodata` section
  - this results in constants being placed in flash - and not copied to RAM

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Some considerations

- It is common for hardware and firmware to have to evolve together
  - example: changing resistor values to tune the performance of the system



COMPILER-DRIVEN LOOKUP TABLE GENERATION

Incorporate governing equations into the code, and let the compiler generate constant tables

- Steinhart-Hart Model (general form)
  $$\frac{1}{T} = \sum_{n=0}^{\infty} a_n \left(\ln\left(\frac{R}{R_0}\right)\right)^n$$
- Resistor Divider
  $$R = \frac{V_x R_1}{V_0 - V_x}$$
- Analog Conversion
  $$V_x = code \cdot \frac{V_0}{2^n}$$

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
1  template<size_t N>
2  constexpr float ThermistorValue(const std::array<float,N> coefficients, const
3  {
4      float denom = 0.0;
5      size_t index = 0;
6      for( auto& coeff : coefficients )
7      {
8          denom += coeff*std::pow(std::log(R), index);
9          index += 1;
10     }
11     float T = 1.0/denom;
12
13     return T;
14 }
15
16 constexpr float ResistanceFromDivider(const float V0, const float V, const flo
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
31 constexpr std::array<float,256>
32 ThermistorTable(std::array<float,N> coeff, const float R0, const float V0, con
33 {
34     std::array<float,256> table = {0};
35
36     size_t code = 0;
37     for( auto& element : table )
38     {
39         auto V = VoltageFromCode(Vref, 8, code);
40         auto R = ResistanceFromDivider(V0, V, R0);
41         element = ThermistorValue(coeff, R);
42         code += 1;
43     }
44
45     return table;
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
18      float R = R0*V0/(V0 - V);
19
20      return R;
21  }
22
23  constexpr float VoltageFromCode(const float Vref, const size_t n, const uint16
24  {
25      float V = Vref*code/std::pow(2,n);
26
27      return V;
28  }
29
30  template<size_t N>
31  constexpr std::array<float,256>
32  ThermistorTable(std::array<float,N> coeff, const float R0, const float V0, con
33  {
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
11      float T = 1.0/denom;
12
13      return T;
14  }
15
16  constexpr float ResistanceFromDivider(const float V0, const float V, const flo
17  {
18      float R = R0*V0*(V0 - V);
19
20      return R;
21  }
22
23  constexpr float VoltageFromCode(const float Vref, const size_t n, const uint16
24  {
25      float V = Vref*code/std::pow(2,n);
26
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
1  template<size_t N>
2  constexpr float ThermistorValue(const std::array<float,N> coefficients, const
3  {
4      float denom = 0.0;
5      size_t index = 0;
6      for( auto& coeff : coefficients )
7      {
8          denom += coeff*std::pow(std::log(R), index);
9          index += 1;
10     }
11     float T = 1.0/denom;
12
13     return T;
14 }
15
16 constexpr float ResistanceFromDivider(const float V0, const float V, const flo
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Table Generation

```
1  template<size_t N>
2  constexpr float ThermistorValue(const std::array<float,N> coefficients, const
3  {
4      float denom = 0.0;
5      size_t index = 0;
6      for( auto& coeff : coefficients )
7      {
8          denom += coeff*std::pow(std::log(R), index);
9          index += 1;
10     }
11     float T = 1.0/denom;
12
13     return T;
14 }
15
16 constexpr float ResistanceFromDivider(const float V0, const float V, const flo
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

Instantiation

```
static constexpr std::array<float,4> ThermistorCoefficients { 1.0e-3, 1.0e-4, 1.0e

constexpr auto ThermistorLookup = ThermistorTable(ThermistorCoefficients, 10.0e3,

int main(int , char** )
{

    for( auto element : ThermistorLookup ) std::cout << element << "\r\n";

}
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
$ readelf --sections --wide main.elf
Section Headers:
  [Nr] Name              Type            Address          Off    Size   ES Flg
  [ 0]                   NULL            0000000000000000 000000 000000 00
  [ 1] .interp           PROGBITS        0000000000000318 000318 00001c 00   A
  [ 2] .note.gnu.property NOTE            0000000000000338 000338 000030 00
  [ 3] .note.gnu.build-id NOTE            0000000000000368 000368 000024 00
  [ 4] .note.ABI-tag     NOTE            000000000000038c 00038c 000020 00   A
  [ 5] .gnu.hash         GNU_HASH        00000000000003b0 0003b0 000028 00   A
  [ 6] .dynsym           DYNSYM          00000000000003d8 0003d8 000120 18   A
  [ 7] .dynstr           STRTAB          00000000000004f8 0004f8 00015f 00   A
  [ 8] .gnu.version      VERSYM          0000000000000658 000658 000018 02   A
  [ 9] .gnu.version_r    VERNEED         0000000000000670 000670 000060 00   A
  [10] .rela.dyn         RELA            00000000000006d0 0006d0 000108 18   A
  [11] .rela.plt         RELA            00000000000007d8 0007d8 000060 18  AI
  [12] .init             PROGBITS        0000000000001000 001000 00001b 00  AX
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
15   [11] .rela.plt         RELA            00000000000007d8 0007d8 000060 18  AI
16   [12] .init             PROGBITS        0000000000001000 001000 00001b 00  AX
17   [13] .plt              PROGBITS        0000000000001020 001020 000050 10  AX
18   [14] .plt.got          PROGBITS        0000000000001070 001070 000010 10  AX
19   [15] .plt.sec          PROGBITS        0000000000001080 001080 000040 10  AX
20   [16] .text             PROGBITS        00000000000010c0 0010c0 000169 00  AX
21   [17] .fini             PROGBITS        000000000000122c 00122c 00000d 00  AX
22   [18] .rodata           PROGBITS        0000000000002000 002000 000420 00   A
23   [19] .eh_frame_hdr     PROGBITS        0000000000002420 002420 00003c 00   A
24   [20] .eh_frame         PROGBITS        0000000000002460 002460 0000d4 00   A
25   [21] .init_array       INIT_ARRAY      0000000000003d80 002d80 000010 08  WA
26   [22] .fini_array       FINI_ARRAY      0000000000003d90 002d90 000008 08  WA
27   [23] .dynamic          DYNAMIC         0000000000003d98 002d98 000200 10  WA
28   [24] .got              PROGBITS        0000000000003f98 002f98 000068 08  WA
29   [25] .data             PROGBITS        0000000000004000 003000 000008 00  WA
30   [26] .bss              NOBITS          0000000000004040 003008 000118 00  WA
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
34   [30] .shstrtab         STRTAB          0000000000000000 0036e5 00011a 00
35 Key to Flags:
36   W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
37   L (link order), O (extra OS processing required), G (group), T (TLS),
38   C (compressed), x (unknown), o (OS specific), E (exclude),
39   D (mbind), l (large), p (processor specific)
40
41 $ readelf --symbols --wide main.elf
42 Symbol table '.dynsym' contains 12 entries:
43    Num:    Value          Size Type    Bind   Vis      Ndx Name
44      0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
45      1: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@G
46      2: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __cxa_atexit@GLIBC_
47      3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND std::basic_ostream<

48      4: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND std::ios_base::Init
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

```
56
57 Symbol table '.symtab' contains 41 entries:
58    Num:    Value          Size Type    Bind   Vis      Ndx Name
59      0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
60      1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS Scrt1.o
61      2: 000000000000038c    32 OBJECT  LOCAL  DEFAULT    4 __abi_tag
62      3: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS main.cpp
63      4: 0000000000002020  1024 OBJECT  LOCAL  DEFAULT   18 ThermistorLookup
64      5: 000000000000110d    43 FUNC    LOCAL  DEFAULT   16 _GLOBAL__sub_I_main
65      6: 0000000000004151     1 OBJECT  LOCAL  DEFAULT   26 std::__ioinit
66      7: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
67      8: 0000000000001170     0 FUNC    LOCAL  DEFAULT   16 deregister_tm_clone
68      9: 00000000000011a0     0 FUNC    LOCAL  DEFAULT   16 register_tm_clones
69     10: 00000000000011e0     0 FUNC    LOCAL  DEFAULT   16 __do_global_dtors_a
70     11: 0000000000004150     1 OBJECT  LOCAL  DEFAULT   26 completed.0
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

- *Portability*: Standard Library math functions are not guaranteed to be `constexpr`
- *Solution*: Use a purpose-built compile-time math library such as gcem

```
#include "gcem.hpp"
template<size_t N>
constexpr float ThermistorValue(const std::array<float,N> coefficients, const floa
{
    float denom = 0.0;
    size_t index = 0;
    for( auto& coeff : coefficients )
    {
        denom += coeff*gcem::pow(gcem::log(R), index);
        index += 1;
    }
    float T = 1.0/denom;

    return T;
}
```

COMPILER-DRIVEN LOOKUP TABLE GENERATION

- Lookup tables can be very fast
  - Sometimes critical for embedded, real-time applications
- We can place the design support in the source code
  - No external processes to cause mistakes
  - No complexities in the build process
- Tables reside in nonvolatile memory rather than precious RAM

CODE WITH NUMBER STRUCTURES HUMANS CAN ACTUALLY READ AND EDIT

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address
- Internet Protocol

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address
- Internet Protocol
  - 32-bit IPv4 addresses

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address
- Internet Protocol
  - 32-bit IPv4 addresses
  - 128-bit IPv6 addresses

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address
- Internet Protocol
  - 32-bit IPv4 addresses
  - 128-bit IPv6 addresses
- USB

ADDRESS-LIKE STRUCTURES

- Many embedded applications use long numeric structures
  - Frequently known at compile time for storage in nonvolatile memory
- Bluetooth
  - 128-bit UUIDs
  - 48-bit MAC address
- Internet Protocol
  - 32-bit IPv4 addresses
  - 128-bit IPv6 addresses
- USB
  - Unicode encoded string (descriptors)

ADDRESS-LIKE STRUCTURES

The Old Way

```
// canonical UUID
// { 01234567-89AB-CDEF-0123-456789ABCDEF }
static constexpr uint8_t uuid1[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xE
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
// canonical MAC
// 12:34:56:78:90:AB
static constexpr uint8_t mac1[] = { 0x01, 0x23, 0x45, 0x67, 0x89, 0xAB };
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // canonical IPv4 address
2 // 192.168.0.1/24
3 static constexpr struct IP4Addr { uint8_t Address[4]; uint8_t MaskBits; } addre
4
5 // canonical IPv6 address
6 // 2001:db8::1:0
7 uint16_t IPv6Addr[] = { 0x2001, 0x0db8, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001,
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // canonical IPv4 address
2 // 192.168.0.1/24
3 static constexpr struct IP4Addr { uint8_t Address[4]; uint8_t MaskBits; } addre
4
5 // canonical IPv6 address
6 // 2001:db8::1:0
7 uint16_t IPv6Addr[] = { 0x2001, 0x0db8, 0x0000, 0x0000, 0x0000, 0x0000, 0x0001,
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```

ADDRESS-LIKE STRUCTURES

The Old Way

```
1 // Unicode string constant
2 static constexpr char devicedesc[] = { 'W', 0, 'i', 0, 'd', 0, 'g', 0, 'e', 0,
3
4 // USB string descriptor
5 #define USB_STRING 0x03
6 static constexpr char DeviceStringDescriptor[] = { 14, USB_STRING, 'W', 0, 'i',
7
8 #define CHAR_TO_UNICODE(x) x, 0x00
9 static constexpr uint8_t DeviceDescription[] = { CHAR_TO_UNICODE('M'), CHAR_TO_
```

ADDRESS-LIKE STRUCTURES

```
class UUID
{
    private:
    std::array<uint8_t,16> mData {0};

    public:
    UUID() = default;

    friend constexpr UUID operator""_uuid(const char* text, size_t length);
};
```

ADDRESS-LIKE STRUCTURES

```
1  constexpr UUID operator""_uuid(const char* text, size_t length)
2  {
3      UUID result;
4      size_t index = 0;
5      size_t count = 0;
6      while ( index < length )
7      {
8          if ( ( text[index] >= '0' ) && (text[index] <= '9') )
9          {
10             result.mData[index] = text[index] - '0';
11             index == 1;
12             count += 1;
13         }
14         else if ( ( text[index] >= 'a' ) && (text[index] <= 'f') )
15         {
16             result.mData[index] = text[index] - 'a';
```

ADDRESS-LIKE STRUCTURES

```
3       UUID result;
4       size_t index = 0;
5       size_t count = 0;
6       while ( index < length )
7       {
8           if ( ( text[index] >= '0' ) && (text[index] <= '9') )
9           {
10              result.mData[index] = text[index] - '0';
11              index == 1;
12              count += 1;
13          }
14          else if ( ( text[index] >= 'a' ) && (text[index] <= 'f') )
15          {
16              result.mData[index] = text[index] - 'a';
17              index == 1;
18              count += 1;
```

ADDRESS-LIKE STRUCTURES

```
9           {
10              result.mData[index] = text[index] - '0';
11              index == 1;
12              count += 1;
13          }
14          else if ( ( text[index] >= 'a' ) && (text[index] <= 'f') )
15          {
16              result.mData[index] = text[index] - 'a';
17              index == 1;
18              count += 1;
19          }
20          else if ( ( text[index] >= 'A' ) && (text[index] <= 'F') )
21          {
22              result.mData[count] = text[index] - 'A';
23              index == 1;
24              count += 1;
```

ADDRESS-LIKE STRUCTURES

```
15          {
16              result.mData[index] = text[index] - 'a';
17              index == 1;
18              count += 1;
19          }
20          else if ( ( text[index] >= 'A' ) && (text[index] <= 'F') )
21          {
22              result.mData[count] = text[index] - 'A';
23              index == 1;
24              count += 1;
25          }
26          else
27          {
28              // bogus character checking here
29              index += 1;
30          }
```

ADDRESS-LIKE STRUCTURES

```
20          else if ( ( text[index] >= 'A' ) && (text[index] <= 'F' ) )
21          {
22              result.mData[count] = text[index] - 'A';
23              index == 1;
24              count += 1;
25          }
26          else
27          {
28              // bogus character checking here
29              index += 1;
30          }
31      }
32      if ( count != 32 ) throw std::invalid_argument("Invalid UUID string");
33
34      return result;
35  }
```

ADDRESS-LIKE STRUCTURES

```
20          else if ( ( text[index] >= 'A' ) && (text[index] <= 'F' ) )
21          {
22              result.mData[count] = text[index] - 'A';
23              index == 1;
24              count += 1;
25          }
26          else
27          {
28              // bogus character checking here
29              index += 1;
30          }
31      }
32      if ( count != 32 ) throw std::invalid_argument("Invalid UUID string");
33
34      return result;
35  }
```

ADDRESS-LIKE STRUCTURES

```
static const UUID uuid = "{ 01234567-89AB-CDEF-0123-456789ABCDEF }"_uuid;
static const MACAddr mac = "12:34:56:78:90:AB"_macaddr;
static const IP4Addr addr = "192.168.0.1/24"_ip4addr;
static const USBString devdescr = "Widget"_utf16;
static const auto devdescr = u"Widget";
```

USE STREAM-BASED IO, BUT SKIP THE LIBRARY

LEAN STREAM-BASED IO

```
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```

LEAN STREAM-BASED IO

```cpp
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```

```
$ arm-none-eabi-g++ -g0 -Os -std=c++20 \
-march=armv7e-m+fp -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mtune=cortex-m4 \
-Wall -Wextra -Wpedantic -Wno-psabi \
-fno-rtti -fno-exceptions -ffunction-sections \
-specs=nosys.specs -Wl,--gc-sections \
main.cpp -o main.elf && arm-none-eabi-objcopy -O binary main.elf main.bin
```

LEAN STREAM-BASED IO

```
#include <iostream>

int main(int , char** )
{
    std::cout << "Hello, world!" << "\r\n";
}
```

```
$ arm-none-eabi-g++ -g0 -Os -std=c++20 \
-march=armv7e-m+fp -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mtune=cortex-m4 \
-Wall -Wextra -Wpedantic -Wno-psabi \
-fno-rtti -fno-exceptions -ffunction-sections \
-specs=nosys.specs -Wl,--gc-sections \
main.cpp -o main.elf && arm-none-eabi-objcopy -O binary main.elf main.bin
```

```
$ du -bh main-bare.bin main-conventional.bin
1.7K    main-bare.bin
157K    main-conventional.bin
```

LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
1  namespace mcu
2  {
3  class FileStream
4  {
5      public:
6      using Radix_t = enum class RadixEnum { Binary=2, Octal=8, Decimal=10, Hexa
7
8      protected:
9      size_t mFileDescriptor;
10     Radix_t mRadixSetting;
11
12     public:
13     FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15     public:
16     FileStream& operator<<(const char* string)
```

LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
1  namespace mcu
2  {
3  class FileStream
4  {
5      public:
6      using Radix_t = enum class RadixEnum { Binary=2, Octal=8, Decimal=10, Hexa
7
8      protected:
9      size_t mFileDescriptor;
10     Radix_t mRadixSetting;
11
12     public:
13     FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15     public:
16     FileStream& operator<<(const char* string)
```

LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
 3 class FileStream
 4 {
 5     public:
 6     using Radix_t = enum class RadixEnum { Binary=2, Octal=8, Decimal=10, Hexa
 7
 8     protected:
 9     size_t mFileDescriptor;
10     Radix_t mRadixSetting;
11
12     public:
13     FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15     public:
16     FileStream& operator<<(const char* string)
17     {
18         const size_t length = std::strlen(string);
```

LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
11
12      public:
13      FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15      public:
16      FileStream& operator<<(const char* string)
17      {
18          const size_t length = std::strlen(string);
19          _write(mFileDescriptor, string, length);
20
21          return *this;
22      }
23      template<typename U> requires std::integral<U>
24      FileSteam& operator<<(const U value) { ... }
25      template<typename U> requires std::floating_point<U>
26      FileSteam& operator<<(const U value) { ... }
```

LEAN STREAM-BASED IO

Create a Lightweight Filestream Object

```
12      public:
13      FileStream(size_t fd) : mFileDescriptor(fd), mRadixSetting(RadixEnum::Hexa
14
15      public:
16      FileStream& operator<<(const char* string)
17      {
18          const size_t length = std::strlen(string);
19          _write(mFileDescriptor, string, length);
20
21          return *this;
22      }
23      template<typename U> requires std::integral<U>
24      FileSteam& operator<<(const U value) { ... }
25      template<typename U> requires std::floating_point<U>
26      FileSteam& operator<<(const U value) { ... }
```

LEAN STREAM-BASED IO

Instantiate Filestreams for Each Output

```
namespace mcu
{
    FileStream debug(1);
    FileStream swo(2);
}
```

LEAN STREAM-BASED IO

Some Plumbing for Output Modes

```
extern "C" int _write(int fd, const void* data, size_t length)
{
    switch (fd)
    {
        case 1:
            SWOWrite((const char*)data, length);
            break;
        case 2:
            UARTWrite((const char*)data, length);
            break;
        case 3:
            UDPWrite(connection, (const char*)data, length)
        ...
    }
}
```

LEAN STREAM-BASED IO

The Traditional Use Case

```
#include "filestream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```

LEAN STREAM-BASED IO

It's Pretty Small

```
#include "filestream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```

LEAN STREAM-BASED IO

It's Pretty Small

```
#include "filestream.hpp"

int main(int , char** )
{
    mcu::swo << "Hello, Serial Wire Debug!" << "\r\n";
    mcu::debug << "Hello, UART Debug!" << "\r\n";
}
```

```
$ du -bh *.bin
1.7K    main-bare.bin
2.3K    main-better.bin
157K    main-conventional.bin
```

LEAN STREAM-BASED IO

Add Some Bells and Whistles

```
void TroublesomeFunction()
{
    mcu::debug << FileStream::RadixEnum::Hexadecimal << 32 << "\r\n";
}
```

LEAN STREAM-BASED IO

Add Some Bells and Whistles

```
void TroublesomeFunction()
{
    mcu::debug << FileStream::RadixEnum::Hexadecimal << 32 << "\r\n";
}
```

```
0x20
```

LEAN STREAM-BASED IO

Extending to Other Types

```
1  template<typename T>
2  class Tensor { ... };
3
4  FileStream& operator<<(FileStream& stream, const Tensor<float>& tensor)
5  {
6      for(size_t i=0; i < tensor.Dimension(1); i++)
7      {
8          for(size_t j=0; j < tensor.Dimension(0); j++)
9          // ...
10         stream << tensor.Element(i,j);
11         // ...
12     }
13 }
```

LEAN STREAM-BASED IO

Extending to Other Types

```
1  template<typename T>
2  class Tensor { ... };
3
4  FileStream& operator<<(FileStream& stream, const Tensor<float>& tensor)
5  {
6      for(size_t i=0; i < tensor.Dimension(1); i++)
7      {
8          for(size_t j=0; j < tensor.Dimension(0); j++)
9          // ...
10         stream << tensor.Element(i,j);
11         // ...
12     }
13 }
```

LEAN STREAM-BASED IO

Extending to Other Types

```
Tensor<float> result( {10,10} );
mcu::debug << result;
```

LEAN STREAM-BASED IO

Extending to Other Types

```
Tensor<float> result( {10,10} );
mcu::debug << result;
```

```
{ {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, } }
```

DARE TO USE THE HEAP

DARE TO USE THE HEAP

(WITH SOME CAVEATS)

DARE TO USE THE HEAP

(WITH SOME CAVEATS)

(IN NON-SAFETY-RELATED APPLICATIONS)

USING HEAP IN EMBEDDED APPLICATIONS

USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion

USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation

USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution

USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution
- Many applications stick to static allocation

USING HEAP IN EMBEDDED APPLICATIONS

- Runtime behavior can cause heap exhaustion
- Long runtimes can cause (unsolvable) fragmentation
- Heap errors have no graceful resolution
- Many applications stick to static allocation
- Off Limits: std::vector std::map std::list std::deque

USING HEAP IN EMBEDDED APPLICATIONS

Use in allocate-once scenarios

```
class CalculationNode
{
    public:
    using NodeHandle = std::unique_ptr<CalculationNode>;

    private:
    size_t mNodeNumber;
    std::vector<NodeHandle> mNodeCollection;

    public:
    virtual bool Calculate()
    {
        for ( node : mNodeCollection ) node->Calculate();
    }
};
```

USE ARENA ALLOCATORS

USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
  - Fast
  - Low overhead

USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
  - Fast
  - Low overhead
- Preferably deterministic memory usage

USE ARENA ALLOCATORS

- For monotonic, allocate-once applications
  - Fast
  - Low overhead
- Preferably deterministic memory usage
- See also Lakos (2017) and Steagall (2017)

USE ARENA ALLOCATORS

Overriding the global new operator

```
void* operator new(size_t size)
{
    static constexpr size_t ArenaSize = 1'000;
    static char Arena[ArenaSize];

    auto ptr = Arena + AllocatedBytes;
    AllocatedBytes += size;
    return ptr;
}
```

USE ARENA ALLOCATORS

Overriding new on a per-class basis

```
1  template<typename T>
2  class ArenaAllocator
3  {
4      static constexpr size_t ArenaSize = 1'000;
5      static char Arena[ArenaSize];
6
7      public:
8      T* allocate(size_t size) { /* ... */ }
9  };
10
11 class CalculationNode
12 {
13     public:
14     using NodeHandle = std::unique_ptr<CalculationNode>;
15
16     private:
```

USE ARENA ALLOCATORS

Overriding new on a per-class basis

```
10
11 class CalculationNode
12 {
13     public:
14     using NodeHandle = std::unique_ptr<CalculationNode>;
15
16     private:
17     std::vector<NodeHandle,ArenaAllocator<NodeHandle>> mCollectionOfNodes;
18
19     void* operator new(size_t size)
20     {
21         ArenaAllocator<char> alloc;
22         return alloc.allocate(size);
23     }
24 };
```

USE ARENA ALLOCATORS

Demo Time

USE ARENA ALLOCATORS

It is possible to do runtime allocation while avoiding pitfalls of fragmentation, and to avoid heap overhead. Be safe.

IMPLEMENT ONE FUNCTION TO UNLOCK ALL OF TIME

IMPLEMENT ONE FUNCTION TO UNLOCK ALL OF TIME (FUNCTIONS)

UNLOCK STD::CHRONO

- Embedded time sources
  - Relative (Hardware timer)
  - Absolute (RTC)
- Applications
  - Performance instrumentation
  - Time-of-day and calendaring

UNLOCK STD::CHRONO

```
1  static TIM_TypeDef* ChronoHWTimer = nullptr;
2
3  bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
4  {
5      bool success = true;
6      const uint32_t timer_freq = 1'000'000;
7
8      // program timer for std::chrono functions
9      ChronoHWTimer = timer;
10     rccreg |= rccvalue;
11     timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
12     timer->ARR = 0xFFFFFFFF;
13     timer->CNT = 0ul;
14
15     return success;
16 }
```

UNLOCK STD::CHRONO

```cpp
static TIM_TypeDef* ChronoHWTimer = nullptr;

bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
{
    bool success = true;
    const uint32_t timer_freq = 1'000'000;

    // program timer for std::chrono functions
    ChronoHWTimer = timer;
    rccreg |= rccvalue;
    timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
    timer->ARR = 0xFFFFFFFF;
    timer->CNT = 0ul;

    return success;
}
```

UNLOCK STD::CHRONO

```
1  static TIM_TypeDef* ChronoHWTimer = nullptr;
2
3  bool MicrosecondClockConfigure(TIM_TypeDef* timer, volatile uint32_t& rccreg,
4  {
5      bool success = true;
6      const uint32_t timer_freq = 1'000'000;
7
8      // program timer for std::chrono functions
9      ChronoHWTimer = timer;
10     rccreg |= rccvalue;
11     timer->PSC = (ClockTree.pclk2/timer_freq) - 1;
12     timer->ARR = 0xFFFFFFFF;
13     timer->CNT = 0ul;
14
15     return success;
16 }
```

UNLOCK STD::CHRONO

```
namespace std::chrono
{
    time_point<high_resolution_clock> high_resolution_clock::now()
    {
        return time_point(
            nanoseconds(
                1000ull*static_cast<nanoseconds::rep>(chrono_timer->CNT)));
    }
} // namespace std::chrono
```

UNLOCK STD::CHRONO

Choices for clock implementations

- `std::chrono::system_clock`
  - Wall time clock (RTC)
- `std::chrono::steady_clock`
  - Monotonic clock
- `std::chrono::high_resolution_clock`
  - Highest resolution clock
- `std::chrono::utc_clock`
- ...

UNLOCK STD::CHRONO

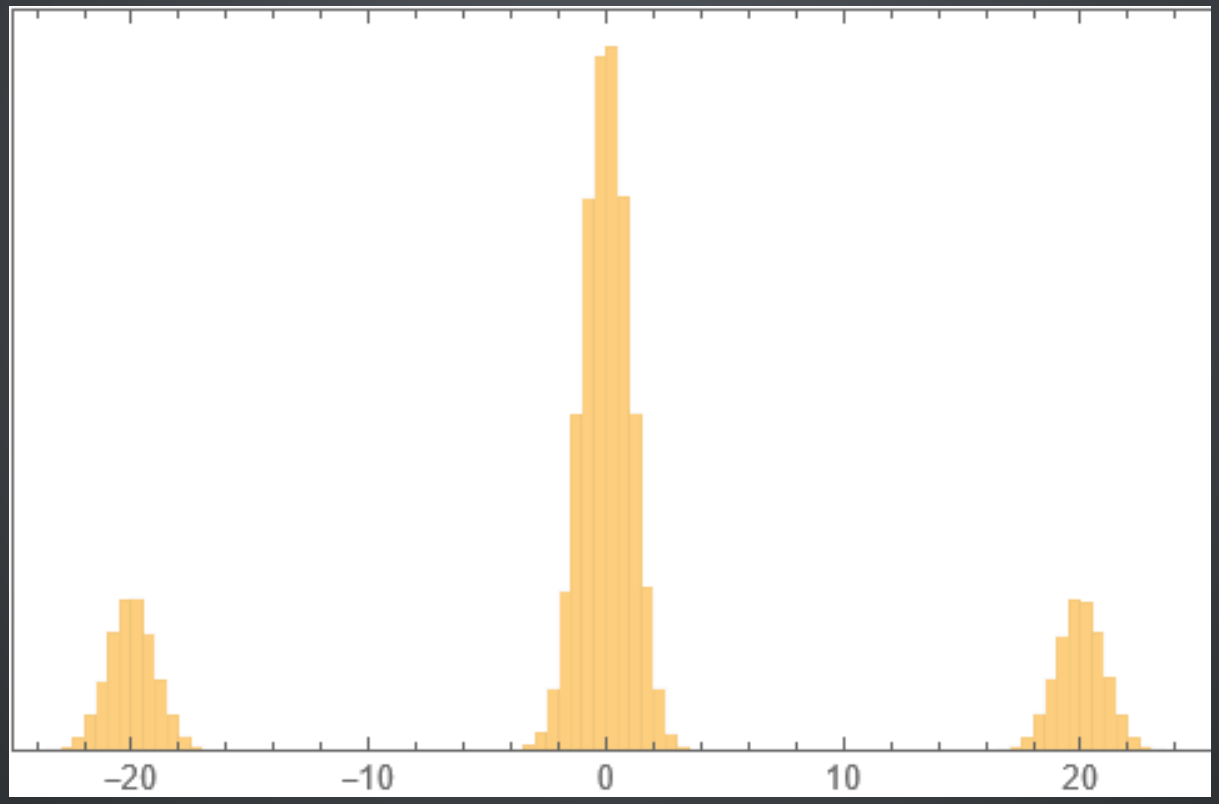
Demo time

Implement `std::random_device`

and Get Free ~~Beer~~ Library Code

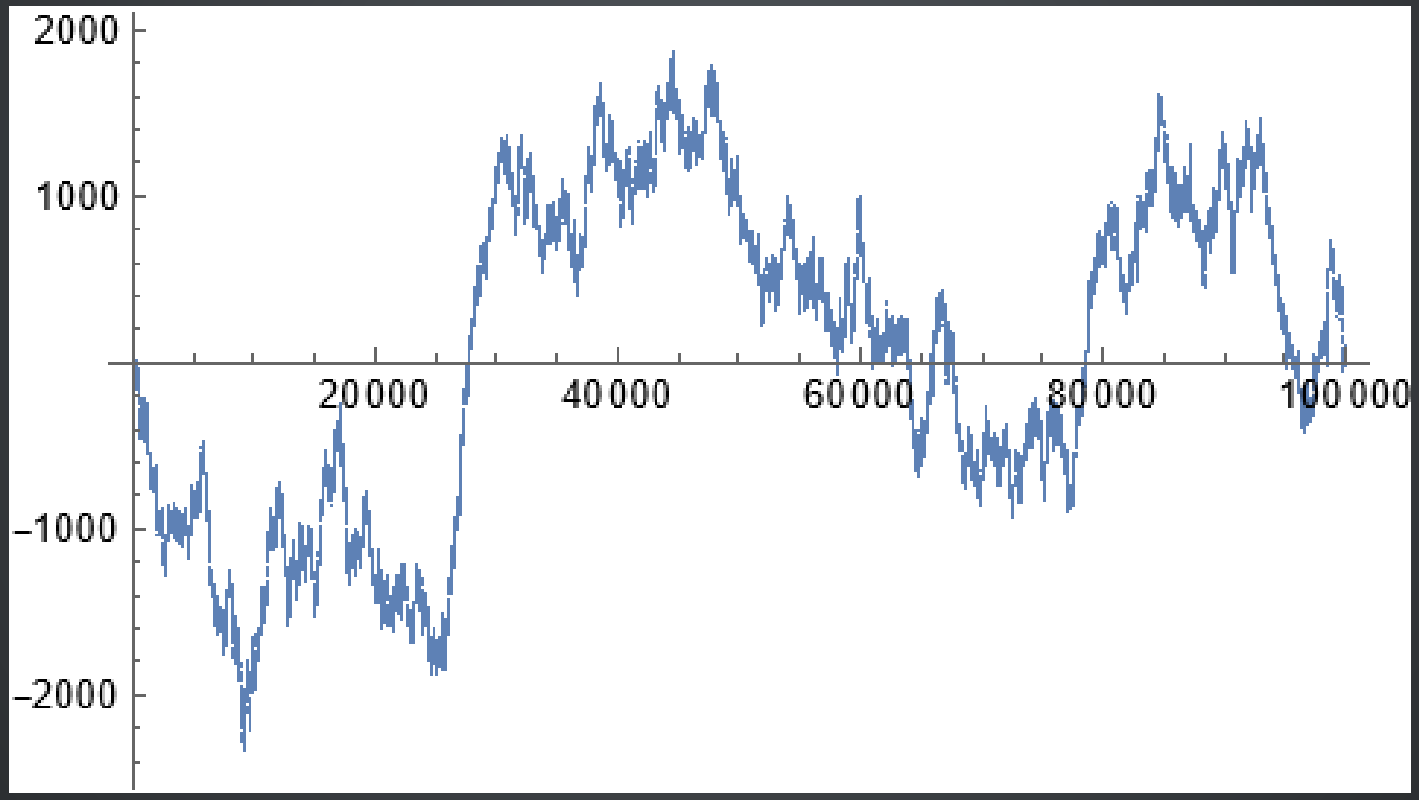
SCENARIO

- You need a random number generator with a multimodal probability distribution



SCENARIO

- This gives a random walk with occasional large changes




febreze
home
COLLECTION
chez-soi
flameless luminaire
starter kit
trousse de départ
luminaire sans flamme
green tea citrus
agrumes et thé vert
CONTAINS 1 REUSABLE LAMP BASE, 2 SCENTED SHADES & 2 BATTERIES
CONTIENT 1 BASE RÉUTILISABLE, 2 RECHARGES PARFUMÉES ET 2 PILES

FUN WITH RANDOM NUMBERS

```
class BimodalDistrubution
{
    public:
    using ModeParamsType = std::normal_distribution<float>::param_type;

    private:
    std::normal_distribution<float> mFirstModeDistribution;
    std::normal_distribution<float> mSecondModeDistribution;
    std::uniform_real_distribution<float> mWeightingDistribution;
    float mMode1Weight;
```

FUN WITH RANDOM NUMBERS

```
class BimodalDistrubution
{
    ...
    template<typename Generator>
    float operator()(Generator& device)
    {
        auto x = mWeightingDistribution(device);
        float result;

        if ( std::abs(x) < mMode1Weight )
        {
            result = mFirstModeDistribution(device);
        }
        else
        {
            result = signum(x)*mSecondModeDistribution(device);
```

FUN WITH RANDOM NUMBERS

```
auto dev = std::random_device();
auto dist = BimodalDistribution();
float x = dist(dev);
```

FUN WITH RANDOM NUMBERS

Demo Time

FUN WITH RANDOM NUMBERS

```
1  namespace mcu
2  {
3  class random_device
4  {
5      public:
6      using result_type = unsigned int;
7      static constexpr result_type min() { return std::numeric_limits<result_typ
8      static constexpr result_type max() { return std::numeric_limits<result_typ
9      constexpr double entropy() const noexcept { return 32.0; }
10
11     random_device()
12     {
13         RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14         RNG->CR |= RNG_CR_RNGEN;
15     }
16     random_device& operator=(const random_device& other) = delete;
```

FUN WITH RANDOM NUMBERS

```
6       using result_type = unsigned int;
7       static constexpr result_type min() { return std::numeric_limits<result_typ
8       static constexpr result_type max() { return std::numeric_limits<result_typ
9       constexpr double entropy() const noexcept { return 32.0; }
10
11      random_device()
12      {
13          RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14          RNG->CR |= RNG_CR_RNGEN;
15      }
16      random_device& operator=(const random_device& other) = delete;
17      result_type operator()()
18      {
19          while ( !(RNG->SR & RNG_SR_DRDY) ) continue;
20          return RNG->DR;
21      }
```

FUN WITH RANDOM NUMBERS

```
8       static constexpr result_type max() { return std::numeric_limits<result_typ
9       constexpr double entropy() const noexcept { return 32.0; }
10
11      random_device()
12      {
13          RCC->AHB2ENR |= RCC_AHB2ENR_RNGEN;
14          RNG->CR |= RNG_CR_RNGEN;
15      }
16      random_device& operator=(const random_device& other) = delete;
17      result_type operator()()
18      {
19          while ( !(RNG->SR & RNG_SR_DRDY) ) continue;
20          return RNG->DR;
21      }
22  };
23  }
```

FUN WITH RANDOM NUMBERS

Demo Time