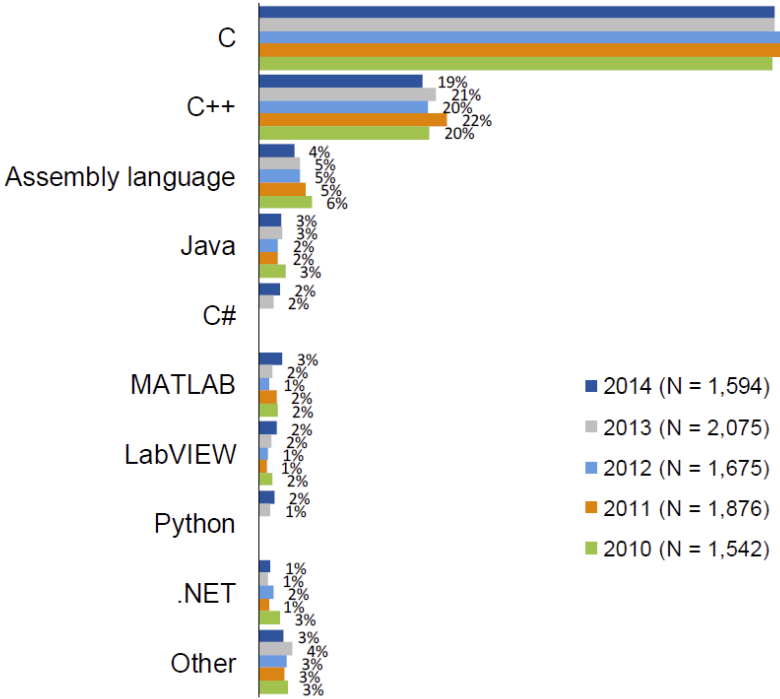
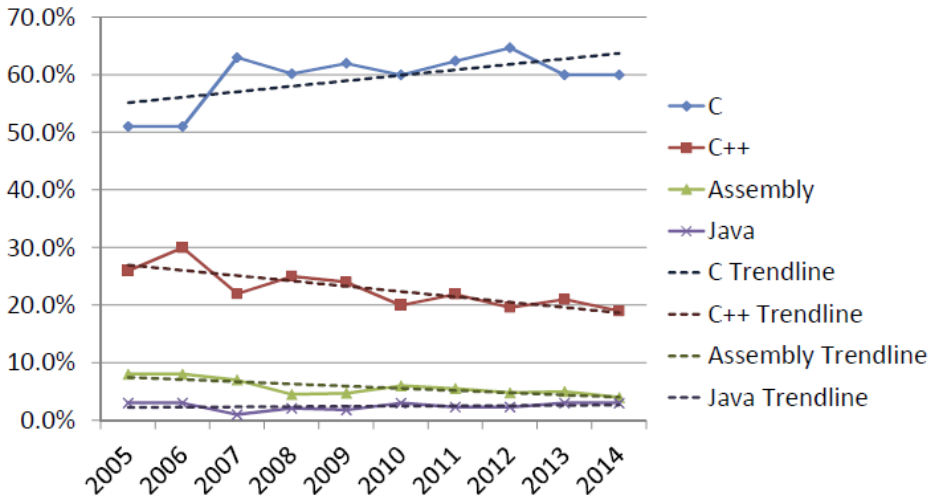
EMBEDDED C++

Henri Kuuste

Master’s seminar 2017

Embedded market study by UBM





The C and ASM view

1. C++ is **slow**
2. C++ produces **bloated** code
   - Objects are large
   - Libraries are large
3. C++ isn't **ROM**able
4. C++ is **dangerous**
5. Abstraction leads to **inefficiency**
6. Lack of **toolchains**

The high level view

1. C++ is **complex**
2. C++ means **pointers**
3. C++ means **bit manipulation**
4. Looks like **C** to me

Language style

```
while (Life){
 live();
 laugh++;
 love=new ♥;
}
```

Modern C++ by Herb Sutter

```
circle* p = new circle( 42 );
vector<shape*> v = load_shapes();
for( vector<shape*>::iterator i = v.begin(); i != v.end(); ++i ) {
   if( *i && **i == *p )
       cout << **i << " is a match\n";
}
// ... later, possibly elsewhere ...
for( vector<shape*>::iterator i = v.begin();
       i != v.end(); ++i ) {
    delete *i;
}
delete p;
```

```
auto p = make_shared<circle>( 42 );
auto v = load_shapes();
for( auto& s : v ) {
    if( s && *s == *p )
        cout << *s << " is a match\n";
}
```

C with Classes

```
uint8_t sch_handle_command(const sch_command_t *command) {
  static const sch_handler_entry_t *handler_entry = NULL;
  static uint8_t retval = 0;

  if(command == NULL)
    THROW(EXM_CMD_SCH, EX_SCH_INV_COMMAND);
  if (g_sch_handler_state.busy)
    return 0;

  TRY {
    g_sch_handler_state.busy = 1;
    g_sch_handler_state.last_cmd = command;

    if (g_sch_handler_state.last_cmd->header.id >SCH_HANDLER_TABLE_NUM_ENTRIES) {
      THROW(EXM_CMD_SCH, EX_SCH_NO_HANDLER);
    } else {
      handler_entry = SCH_READ_HANDLER_TABLE(g_sch_handler_state.last_cmd->header.id);

      if (handler_entry != NULL && handler_entry->handler != NULL) {
        retval = handler_entry->handler(command);
        g_sch_handler_state.last_result = retval;
        if (retval == 1)
          g_sch_handler_state.num_ok++;
        else
          g_sch_handler_state.num_failed++;
      } else {
        THROW(EXM_CMD_SCH, EX_SCH_INV_HANDLER);
      }
    }
  } FINALLY {
    if (THROWN) {
      g_sch_handler_state.last_result = EXCEPTION;
      CONCEAL;
    }
    g_sch_handler_state.busy = 0;
  } ETRY;
  return retval;
}
```

```
commands::retval_t
handler::handle_command(NotNull<const command_t> command) {
  auto guard = get_handler_lock();
  state.last_cmd = command.get();

  try {
    state.last_result.ret = commands::lookup(command->id)();
    state.num_ok++;
  } catch (ex_t &e) {
    state.last_result.ret = commands::retval_t::EXCEPTION;
    state.num_failed++;
    state.last_result.ex = e;
  }

  return state.last_result.ret;
}
```

But wait, there's more...

```
uint8_t sch_ping_handler(sch_command_t *command) {
  typedef struct {
    int time;
  } command_type_t;
  command_type_t *cmd = (command_type_t *)command->args;

  command_output("PONG %X\n", cmd->time);
  return DONE;
}
```

```
struct ping_command : public command_t {
  int time;
  auto handle() const {
    out << "PONG " << hex << time <<endl;
    return retval_t::DONE;
  }
};
register_command(commands::PING, ping_command);
```

```
__READ_ONLY sch_handler_entry_t
   g_handler_table[SCH_HANDLER_TABLE_NUM_ENTRIES] = {
        [0x00] = {sch_ping_handler},    [0x01] = {sch_version_handler},
        [0x02] = {sch_default_handler}, [0x03] = {sch_default_handler},
        [0x04] = {sch_default_handler}, [0x05] = {sch_default_handler},
        [0x06] = {sch_default_handler}, [0x07] = {sch_default_handler},
        [0x08] = {sch_default_handler}, [0x09] = {sch_default_handler},
        [0x0A] = {sch_default_handler}, [0x0B] = {sch_default_handler},
        ....
};
```

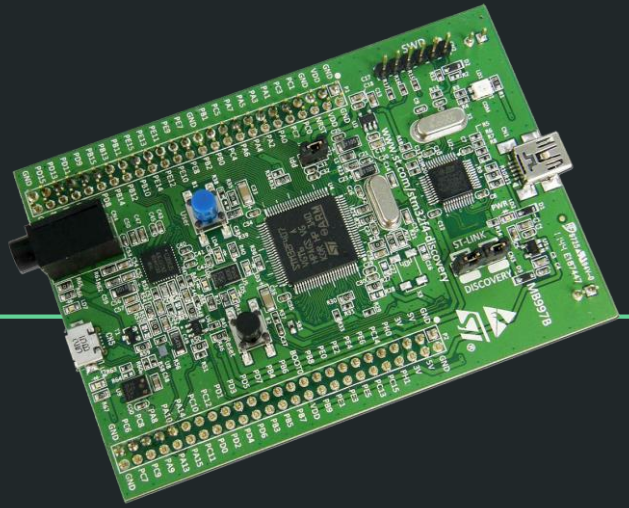
```
init_commands();
```

```
sch_command_t test1 = *((sch_command_t *)tc1);
test1.args = &tc1[3];
sch_handle_command((sch_command_t *)&test1);
```

```
auto cmd = scheduler::as_command(tc1);

scheduler::handler::handle_command(cmd);
```

A BASIC CASE STUDY



Baseline

```
int main(void) {
        // do nothing
}
```

```
int main(void) {
        // do nothing
}
```

STM32F407VG

gcc-arm-none-eabi version **6.2**

-mthumb -mcpu=**cortex-m4** -mlittle-endian -mfpu=fpv4-sp-d16 -mfloat-abi=hard

-Os -fdata-sections -ffunction-sections -fno-stack-protector **--gc-sections -flto** -finline-limit=150

-std=**gnu11**

-std=**c++1z** -fno-rtti -fno-enforce-eh-specs

text	data	bss	dec
1176	1076	1564	**3816**
0	0	0	**0**

Difference
64 bytes
0 bytes

text	data	bss	dec
1112	1076	1564	**3752**
0	0	0	0

Register access

```
__attribute__((unused)) volatile
uint32_t dummy = 0;

RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
dummy = RCC->AHB1ENR;
GPIOD->MODER = (1 << (12 * 2));

while (1) {
  GPIOD->ODR ^= (1 << 12);
  delay_cycles(5000000);
}
```

```
[[gnu::unused]] volatile
uint32_t dummy = 0;

RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
dummy = RCC->AHB1ENR;
GPIOD->MODER = (1 << (12 * 2));

while (1) {
  GPIOD->ODR ^= (1 << 12);
  delay_cycles(5'000'000);
}
```

text	data	bss	dec
1232	1076	1564	**3872**
56	0	0	**56**

Difference
64 bytes
0 bytes

text	data	bss	dec
1168	1076	1564	**3808**
56	0	0	**56**

Low level abstraction

```
#define LED_G          12
#define LED_IO_NUM      3
#define LED_IO       GPIOD

enable_io(LED_IO_NUM);
pin_init(LED_IO, LED_G, GPIO_MODE_OUT);

while (1) {
  toggle_pin(LED_IO, LED_G);
  delay_cycles(5000000);
}
```

```
using led_g = gpio<port::D>::pin<12>;

led_g::port::enable();
led_g::init(pin_mode::out);

while (1) {
  led_g::toggle();
  delay_cycles(5'000'000);
}
```

text	data	bss	dec
1232	1076	1564	**3872**
56	0	0	**56**

Difference
64 bytes
0 bytes

text	data	bss	dec
1168	1076	1564	**3808**
56	0	0	**56**

Hardware Abstraction Layer

```
hal_status_t err = hal_init();
if (err != HAL_STATUS_OK) {
  while (1) {
    // do something nasty
  }
}
```

```
using namespace stm32f4discovery;
initialize();
```

```
while (1) {
  hal_pin_toggle(LED_G);
  hal_delay(500);
}
```

```
while (1) {
  LedGreen::toggle();
  delayMilliseconds(500);
}
```

text	data	bss	dec
1300	1172	1564	**4036**
124	96	0	**164**

Difference
196 bytes
132 bytes

text	data	bss	dec
1200	1076	1564	**3840**
32	0	0	**32**

A more “complex” example

```
UART_HandleTypeDef huart2;
void EXTI0_IRQHandler() { HAL_GPIO_EXTI_IRQHandler(B1_PIN); }
void GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
   GPIO_WritePin(LD6_PORT, LD6_PIN, GPIO_PIN_RESET);
   GPIO_WritePin(LD4_PORT, LD4_PIN, GPIO_PIN_RESET);
}

HAL_Init();
SystemClock_Config();
GPIO_Init();
USART2_UART_Init(115200, 12);

while (1) {
 if (GPIO_ReadPin(B1_GPIO_PORT, B1_PIN)) {
    GPIO_TogglePin(LD6_GPIO_PORT, LD6_PIN);
    if (UART_Transmit(&huart2, (uint8_t *)"B", 1, 5000) != HAL_OK) {  Error_Handler(); }
    HAL_Delay(125);
 } else {
    GPIO_TogglePin(LD4_GPIO_PORT, LD4_PIN);
    if (UART_Transmit(&huart2, (uint8_t *)".", 1, 5000) != HAL_OK) { Error_Handler(); }

    HAL_Delay(500);
 }
}
```

37614 cycles ≈ 224 µs

```
XPCC_ISR(Button::ISR) {
   Button::acknowledgeExternalInterruptFlag();

   LedBlue::reset();
   LedGreen::reset();
}

initialize();
Usart2::initialize<systemClock, 115200>(12);

while (1) {
   if (Button::read()) {
      LedBlue::toggle();
      Usart2::writeBlocking('B');
      xpcc::delayMilliseconds(125);
   } else {
      LedGreen::toggle();
      Usart2::writeBlocking('.');

      xpcc::delayMilliseconds(500);
   }
}
```

3875 cycles ≈ 23 µs

text	data	bss	dec
4132	1084	1632	**6848**
2832	-88	68	**2812**

Difference
2024 bytes
1828 bytes

text	data	bss	dec
1900	1084	1840	**4824**
700	8	276	**984**

Busting myths

Zero overhead

- All of C
- Classes
  - Non-virtual member functions
  - Static functions and data
- Namespaces
- Function and operator overloading
- Default parameters
- Automatic variable type deduction
- Literal types and digit separators
- Automatic type deduction
- Range-based for loops
- Initializer lists

- Constructors and destructors
- Single inheritance
- Lambda functions
- Virtual functions
- Multiple inheritance
- Generic template programming
- Threading and atomic operations

Considerable overhead

- Dynamic memory management
  - Smart pointers
- Run-time type information
- Exceptions*
- Standard library (mostly)*
  - Embedded alternatives on GitHub

Negative overhead

- R-value references and move semantics
- static_assert
- constexpr
- Metaprogramming
  - Static templates and polymorphism
  - Variadic templates
  - Expression templates

The C and ASM view

1. C++ is **slow**
2. C++ produces **bloated** code
   - Objects are large
   - Libraries are large
3. C++ isn't **ROM**able
4. C++ is **dangerous**
5. Abstraction leads to **inefficiency**
6. Lack of **toolchains**

The C and ASM view

1. **Myth busted**
2. **Myth busted**
3. Not all parts of C++ are ROMable
4. Yes, especially the C and ASM parts of it.
   - MISRA C++, JSF AV++ etc
5. **Myth busted**
6. The situation is **improving**
   - Major embedded compilers are behind
   - Tools C does not have

The high level view

1. C++ is **complex**
2. C++ means **pointers**
3. C++ means **bit manipulation**
4. Looks like **C** to me

The high level view

1. For a regular user, it is closer to Python than C
2. You have **access** to pointers
3. Talk to your API provider
4. Legacy

And now, for my next trick...

```
uint8_t sch_handle_command(const sch_command_t *command) {
  static const sch_handler_entry_t *handler_entry = NULL;
  static uint8_t retval = 0;

  if(command == NULL)
    THROW(EXM_CMD_SCH, EX_SCH_INV_COMMAND);
  if (g_sch_handler_state.busy)
    return 0;

  TRY {
    g_sch_handler_state.busy = 1;
    g_sch_handler_state.last_cmd = command;

    if (g_sch_handler_state.last_cmd->header.id >SCH_HANDLER_TABLE_NUM_ENTRIES) {
      THROW(EXM_CMD_SCH, EX_SCH_NO_HANDLER);
    } else {
      handler_entry = SCH_READ_HANDLER_TABLE(g_sch_handler_state.last_cmd->header.id);

      if (handler_entry != NULL && handler_entry->handler != NULL) {
        retval = handler_entry->handler(command);
        g_sch_handler_state.last_result = retval;
        if (retval == 1)
          g_sch_handler_state.num_ok++;
        else
          g_sch_handler_state.num_failed++;
      } else {
        THROW(EXM_CMD_SCH, EX_SCH_INV_HANDLER);
      }
    }
  } FINALLY {
    if (THROWN) {
      g_sch_handler_state.last_result = EXCEPTION;
      CONCEAL;
    }
    g_sch_handler_state.busy = 0;
  } ETRY;
  return retval;
}
```

```
commands::cmd_retval_t
handler::handle_command(NotNull<const command_t> command) {
  auto guard = get_handler_lock();
  state.last_cmd = command.get();

  try {
    state.last_result.ret = commands::lookup(command->id)(command.get());
    state.num_ok++;
  } catch (ex_t &e) {
    state.last_result.ret = commands::cmd_retval_t::EXCEPTION;
    state.num_failed++;
    state.last_result.ex = e;
  }

  return state.last_result.ret;
}
```

4x FASTER!!

GIVE

A CHANCE