Author: Fränz Ney (es16m013)
Supervisor: FH-Prof. Dipl.-Ing. Dr. Martin Horauer

Embedded C++

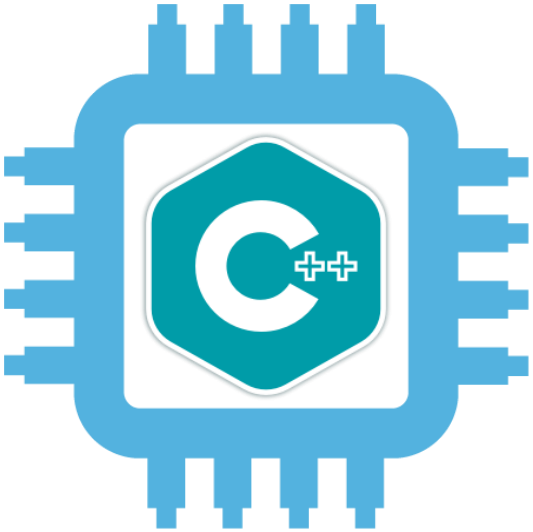
Assessing OO concepts for Embedded Systems

FH University of Applied Sciences TECHNIKUM WIEN

13.11.2018

Overview

- Why using C++ in embedded projects?
- What are the benefits of C++ over C?
- Possible issues of C++.
- Benchmark tests C++ vs C.
- C++ memory overhead.



C++ Concepts

- Function Overloading
  - Using the same function name for two or more functions.
- Default Arguments
  - Default value is used when the caller of the function doesn’t provide an argument.
- Templates
  - Developing functions or classes independent from their data types.
- Pointer vs References
  - Safer alternative for pointers.

C++ Standard Library

- Pro
  - Collection of containers, iterators and algorithms.
  - Tested and documented code can be used.
  - Ease of use.

- Contra
  - Not transparent.
  - Confusing Heap / Stack memory allocation.
  - Can decrease system performance.

C++ Concepts

- Namespaces
  - Increase the modularity of the code.
- Classes
  - Encapsulation of data.
  - Access specifier (private, public, virtual).
- Constructor / Destructor
  - Always work with initialised data and modules.
- Inheritance and Polymorphism
  - Increase the modularity of the code.
  - Using virtual functions or multiple inheritance could influence system performance.

Benefits and Drawbacks

C++ Concept	Pro	Contra
Function Overloading	☺	
Default Arguments	☺	
Function Templates	☺	
Pointer vs References	☺ ☺	
C++ Standard Library	☺	☹ ☹
Namespaces	☺	
Classes	☺ ☺ ☺	
Constructor / Destructor	☺ ☺	☹
Virtual functions and multiple inheritance	☺	☹ ☹ ☹

Popularity of C++

- C is simpler to learn than C++.
- C++ requires more Know-How from the developer.
- Debugging C++ programs could be difficult.
- C Toolchain is available for the most platforms.

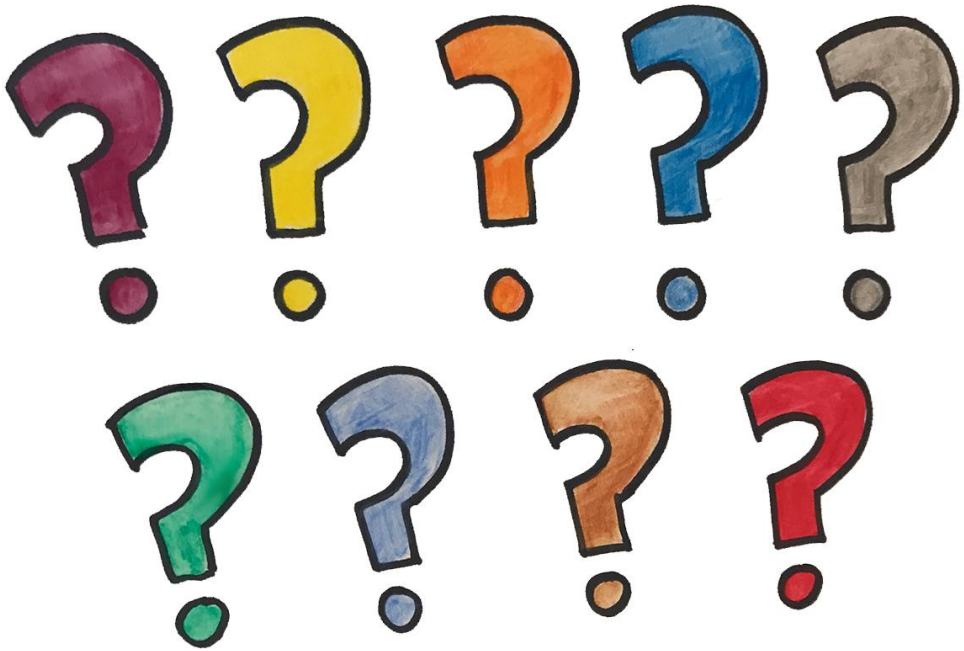
Use cases

- mbed-OS
- distortos
- StratifyOS
- Kvasir
- Modm
- Crect
- EMB²

Outlook

- Compiled language
  - C
  - C++
- Interpreted language
  - Python
  - Embedded Java
  - Script languages
    - Java Script
    - NodeJS

Questions



Function Overloading

- C++

```
int add_i(int a, int b){
  return a + b;
}

double add_d(double a,double b){
  return a + b;
}
```

- C

```
int add(int a, int b){
  return a + b;
}

double add(double a, double b){
  return a + b;
}
```

Default Arguments

- C++

```
int uart_init(int baud=9600,
              int Ndata=8){
  return 0;
}
```

- C

```
int uart_init(int baud,
              int Ndata){

  if(baud < 0)
    baud = 9600;
  if(Ndata < 0)
    Ndata = 8;

  return 0;
}
```

Function Templates

- C++

```
template <typename T> T add(T a, T b)
{
  return a + b;
}
```

- C

```
int add_i(int a, int b)
{
  return a + b;
}

double add_d(double a, double b)
{
  return a + b;
}
```

Pointer vs References

```
int i = 3;

// A pointer to variable i (stores address of i)
int *ptr = &i;

// A reference (or alias) for i.
int &ref = i;
```

C++ Standard Library (heap / stack confusing)

- Code

```
int main() {

    std::string str;

    for(int i=2; i <33; i++){
        str.append("=");
        printf("%02d: %s\n", i,
               str.c_str());
    }

    return 0;
}
```

- Output

```
02: =
03: ==
:::
15: ==============
16: ===============
* Allocate 31 bytes
17: ================
18: =================
:::
30:==============================
31: ===============================
* Allocate 61 bytes
* Deallocate
32: ================================
33: =================================
```

Namespaces (anonym namespaces)

- driver.cpp *(API functions)*

```
namespace driver {

  void serial_init(void){

    is_init = INIT;
    // Do something
  }

  void serial_send(char *data,
                   int len){

    if(is_init == NOT_INIT)
      return;

    for(int i=0; i<len; i++)
      send_byte(data[i]);
  }
} //namespace driver
```

- driver.cpp *(internal usage)*

```
namespace {

  typedef enum {INIT, NOT_INIT} state;
  state is_init = NOT_INIT;

  void send_byte(char byte){
    // Do something
  }
}
```

Classes 1 (Basic Class ‘Com’)

```
class Com {

  public:
  void set_status(int status){
    this->status = status;
  };

  int get_status(void){
    return status;
  };

  virtual void send_byte(char byte){
      printf("Com: %c \n", byte);
  }

  private:
  int status;
};
```

Classes 2 (Derived class ‘Serial’)

- serial.cpp

```
Serial::Serial(int _baud, int _mode)
: baudrate{_baud}, mode{_mode} {

    buffer = new char[100];
}

Serial::~Serial(){

    delete[] buffer;
}

void Serial::set_baudrate(int baudate){

  this->baudrate = baudrate;
}

void Serial::set_mode(int mode){

  this->mode = mode;
}
```

- serial.h

```
class Serial : public Com{
public:
Serial():Serial(9600, 1){};
Serial(int _baud, int _mode);
~Serial();

void set_baudrate(int baudate);
void set_mode(int mode);

private:

int baudrate;
int mode;
char *buffer;
};
```

Classes 3 (Virtual Functions)

- Class ‘Com’

```
class Com {

  public:
  void set_status(int status){
    this->status = status;
  };

  int get_status(void){
    return status;
  };

  virtual void send_byte(char byte){
      printf("Com: %c \n", byte);
  }

  private:
  int status;
};
```

- Class ‘Serial’

```
class Serial : public Com{

public:
Serial():Serial(9600, 1){};

void set_baudrate(int baudate){};
void set_mode(int mode){};
void send_byte(char byte){
    printf("Serial: %c \n", byte);
}

private:

int baudrate;
int mode;
char *buffer;
};
```

Classes 4 (Virtual Function Table)

```
void send(Com& arg) {
  arg.send_byte('A');
}

int main(void){

  Serial ser;

  send(ser);

  return 1;
}
```