41012 Programming for Mechatronic Systems

Week 2

Overview

- Staff
- Link to Industry Needs
- Objectives
- Class Structure
- Assignments
- General Rules
  - Plagiarism
  - Late submissions
  - Peer Review
- In-class exercises

Subject Staff

- Subject Coordinators
  - Alen Alempijevic
- Teaching Staff
  - Alex Virgona

Mechatronic Systems

- 6.4B Devises Interconnected + System on
- Chip Devices Explosion
- Beyond Single Monolithic Code
- OO Paradigm – More Admissible to Systems
- C++ Essential on Many Layers
  - Android Backbone, Libraries, Applications
- Code Reuse / Testing / Documentation
- Robotics Jobs (demand)

Subject objectives

- 1. Design classes that are reusable, reliable and maintainable
- 2. Apply theoretical knowledge of sensors and control to practical programming problems
- 3. Select appropriate class structures and data handling methods for task at hand
- 4. Implement and test object-oriented applications of moderate complexity
- 5. Communicate programming design decisions, dependencies, interconnections, use cases and testing procedures in a written document

Class Structure

- Pre-readings given
- Students view readings, attempt and come to class
- We clarify concepts and push forward with examples
- We build knowledge base (stratify knowledge) towards the assignments; more complex layers of understanding

Assessment

- 1) Assessment I:

  Developing Sensor Class (5%)
- 2) Assessment II:

  Utilising Abstraction for a Range of Sensor Classes (20%)
- 3) Assessment III:

  Threading, Synchronization and Data Integrity (20%)
- 4) Individual Project: (40%)
- 5) Code Review: (9% : 3%+3%+3%)
- 6) Quizes (beginning of class): 6% (6 x 1%)

Practise Quiz

Teaching Delivery

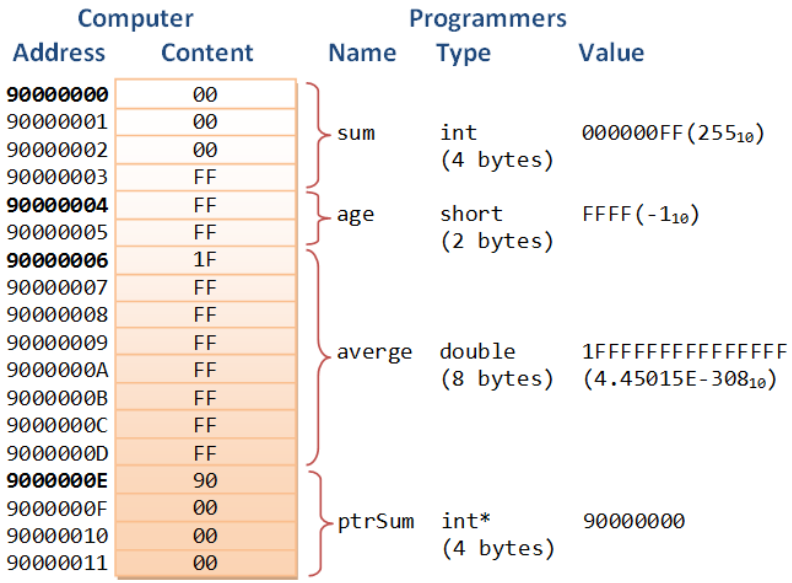
- OS: Ubuntu
- C++11
- Compilation + Tools: CMake
- Documentation: Doxygen
- IDE: QTCreator
- Unit Testing: gtest
- OpenCV (Library)
- ROS (Middleware CBSE)

What you should have done in Week 1

- Download the examples from Lynda.com
- Enable compiling of working.cpp from Chap1
- with CMake
  - Command line
  - QTCreator / Eclipse
- Reviewed C

Pointers & Reference

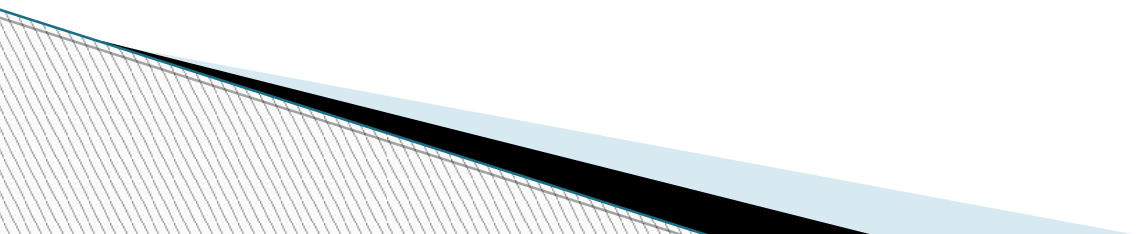
- Pointers are extremely powerful, allow access to memory (memory address) and to manipulate their contents



Pointers & Reference

- The *address-of operator* (&) operates on a variable, and returns the address of the variable

- Pointer:
  - stores a memory address
  - must be declared before they can be used
  - syntax of declaring, place a * in front of the name
  - is associated with a type



Dereference / Indirection

- To get the value pointed to by a pointer (retrieve or modify), you need to use the dereferencing operator *

Pointers & Reference

- int number = 88;
- int * pNumber = &number;
- cout << pNumber<< endl;
- cout << *pNumber << endl;
- *pNumber = 99;
- cout << *pNumber << endl;
- cout << number << endl;

Functions

- Functions allow to structure programs in segments of code to perform individual tasks.
- Generally segmented to be able to be reused

type **name** (parameter1, parameter2, ...)
{ statements }

- What about **void**

Functions

- Create a function that accepts a double value as a parameter and
  - 1. Returns a square value
  - 2. Returns a bool value if the double is greater than zero and the square value instead of initial passed value
  - 3. Returns bool value if the double is greater than zero, the square value, the cube value and the passed value incremented by one

Header / Source File

- Why needed?
  - It allows you to separate *interface* from *implementation*
  - It speeds up compile time

- Generally the interface file is all that is needed
- Allows to have a library with header, inner implementation hidden

- Further Details
- http://www.cplusplus.com/articles/Gw6AC542/

Coding Style Guide

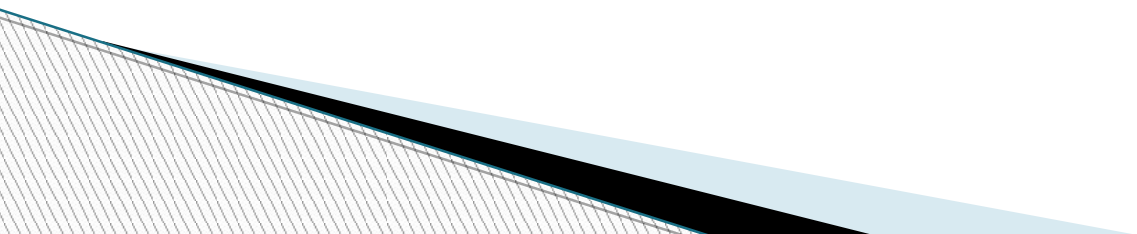
- Code style important for readability of code
- Most important : **CONSISTENCY**
- Common to Adopt Coding Standard
  - **http://wiki.ros.org/CppStyleGuide**
  - **https://google.github.io/styleguide/cppguide.html**

- **We will use ROS C++ Style Guide**

Classes

- *Classes* are an expanded concept of *data structures*:
  - they can contain data members
  - they can also contain functions as members.

- C Struct?
  - Arguably not the same beast!



Classes

```
class Rectangle {
    Rectangle();
    ~Rectangle();
public:
    void setValues (int,int);
    int getArea (void);
    int getPerimeter (void);
private:
    int width_, height_;
};
```

Classes

```
class Rectangle {
public:
void setValues (int,int);
int getArea (void);
int getPerimeter(void);
private:
int width_, height_;
};
```

Questions:

- What are the functions of the class?
- What do they take / return?
- What are the access specifiers?
- Why do we have the setValues method?
- What do the access specifiers guarantee
- Where is the implementation of this class
- What might be the pitfall of this class?
- How can we resolve the pitfall?