Introduction

Overview
In this lesson we will
✓ Begin with a very high level view of the microprocessor and software
✓ Look at solving problems
✓ Introduce a very high level view of a C program
✓ Study basic building blocks of C a program
✓ Identify some metrics for assessing program quality

Getting Started
Goals are to learn
To work with microprocessors
To program in C

What does this really mean
Doesn’t mean we will be writing programs
  Surprised - don't be
What it really means is that we will learn to solve problems
Working with microprocessors is more than
  Buying a microprocessor and connecting a bunch of wires
Learning C more than
  Coding in C language
  Simply writing programs using C language
When we talk about programming
  Will be thinking about things a bit differently

Let's begin by taking quick look at what we will be talking about

Hardware
Includes a lot of different kinds of hardware
✓ Computer
  Microprocessor(s) brains inside of a computer
  CPU – Central Processing Unit
✓ Things we connect to computer
  Keyboard
  Mouse
  Display
  USB devices
  Motors
  Sensors
  Etc.
Software
Includes a lot of different kinds of software
✓ The operating system
   We use and may modify these but typically don’t write
   OSX
   Windows
   Linux
   Unix
✓ The applications
   We write and use these to help solve problems
   Typically called a program
   Writing a program called programming
   Here we really are designing a program
   What distinguishes engineer
   From someone who dabbles with language
✓ The languages
   Applications can be written in variety of different languages
   C
   C++
   Java
   C#
   Among many others
   Different languages have
   Different strengths
   Different weaknesses

Our Jobs
We will
✓ Design some of the hardware
✓ Use other pieces
✓ We will
  • Design and write applications
  • Design and write programs

Solving Problems
For any problem often many different solutions
Solutions can be
✓ Pure hardware
✓ Pure software
✓ Combination of both
Our approach will be to use a combination of both
Let’s look at how we might approach solving a problem

Identify the Problem
An important first step
To solve problem – must first find problem

Some problems are simple others are complex
Consider
• Simple
  Find the area of a triangle
• Complex
  Voice recognition
  Graphics in a video game
    Fill a triangle with a color
    Managing antilock brakes in a car

Gather Information
Once we’ve identified the problem
Next gather information about the problem
   Call such information \textit{data}

Like problem – data can be simple or complex
• Simple
  Height of triangle
  Base of triangle
• Complex
  Video Game
    Texture
    Colors

Identify Relationships Amongst the Data
Relationships often called
• Functions
• Formulas
• Algorithms
Relationships can be simple or complex
  ➢ Simple
    area = \( \frac{1}{2} \) base \( \cdot \) height
  ➢ Complex
    Multiply two matrices
    Transpose matrix
Apply Algorithms to the Data
We observe that an algorithm is simply an ordered set of unambiguous steps
Typically used for solving a problem

Algorithm can be simple or complex
- Simple
  - Single step
- Complex
  - Multiple steps

Hardware and Software Revisited
Have taken high-level view of solving problems
Now look deeper into some tools to aid the process
Start with the hardware – the computer

Computer Hardware
Most contemporary computers
Built around design by ~ 1947
  - Alan Turing
  - Howard Aiken
  - John von Neumann
  - Many others
Developed at
  - Harvard University
    - Called Harvard Architecture
    - Basis for most RISC machines
  - Princeton University
    - Called Princeton von Neumann Architecture
These designs have changed little since then

Components
- CPU
  Where computing done
    - ALU
    - Control
    - General Purpose Registers
- Input
  Keyboard
  Touch Screen
  USB port
  Mouse

- Output
  Display
  Printer
  USB port

- Memory
  Disc ⇒ Registers

The Arduino Microprocessor
Following diagram
High level block diagram of processor on our board – Atmel Atmega 328

Other Stuff…. Built in Peripheral

Input - Output
Now let’s look at the software

**Computer Software**

**Kinds**

- **Application**
  - You write
  - Your program

- **System**
  - Typically you don’t write
  - Operating system
  - Linker - Loader
    - Collects bits of program
    - Organizes into coherent whole
    - Puts into memory to run
  - Input / Output drivers

- **Tools**
  - You may or may not write
  - Editor
  - Compiler - Assembler
  - Debugger

**Combining Pieces**

- Can trace the process
- Translating problem into solution

**Programs and Program Design**

- What is Programming
- A program

Programming

- Let’s start with solving a problem on the software side
  - We call such a process programming
  - We construct a program to solve the problem

Programming is process of

- Selecting an appropriate collection of algorithms
- Identifying the appropriate data
- Put all these into a computer
- Applying the algorithms in the proper way to the data
Achieving the desired result

All sounds simple enough
That third step sounds a bit tricky
How do we do that

Clearly there are some more things that we have to consider
Must have some was to get the algorithms into the computer
Must have some way of representing the data

Another way of looking at programming is to view it as a process of
Translating
  Problem stated in one language
    Natural language
    English
  To another language
    Another natural language
    Mathematical expressions
    Language for computer

Inside computer
  Data represented as collections of 0’s and 1’s
  Algorithms similarly represented

Result of programming is a program

Program
  Program is simply a sequence of instructions
    Direct computer
      ✓ Hardware
      ✓ System Software
    To solve problem

  Program is means by which we apply
    Algorithms to data to get answer

    The steps necessary to solve problem
    How to implement an algorithm

Questions:
  1. Is any set of instructions satisfactory
  2. Are there some instructions that are better than others
     If so
Measuring a Good Program

Computer characteristics
   Rigid
   Dispassionate
   Designed and Built
      To react in precise ways

Program
   Large collection of instructions
   Organized
      Algorithms
      Computational procedures
   To perform desired task

Performance
   Difference between
      Hoped for
      Actual
   Evidence of human failure
      To instruct computer properly

Who gets blamed → Computer
   Errors in program
   Bugs

Goal
   Goal as writers of programs
      Write programs
      Solve problem
      Bug Free

Result
   Both approached asymptotically

More specific goals
   1. Performance
   2. Robustness
   3. Ease of change
   4. Style
Quantify each

- Performance
  - Solve problem
  - Size
  - Speed
  - Ease of use

- Robust
  - Failure conditions
  - Unexpected inputs
  - Side effects
  - Boundary Conditions
  - Passed and returned values

- Change
  - Modularity
  - Reuse and Portability
  - Program or host

- Style
  - Clarity
  - Modularity
  - Readability - Indentation
  - Documentation

Program - Software Development – The Process

Combination of both is probably best

- Top down
- Bottom up

Major pieces

- Specification
- Design
- Code
- Test

Approaches

- Traditional waterfall
  - Complete each phase before starting the next

Rapid Prototype

- Called by various names
- Incremental
- Do a little then repeat
- Need good specification up front
Key
To any approach
Devote most effort
Toward top of the list
Often
People do the reverse
Old saw
Get coding done
Have lots of debugging to do
Specification and design
70-80% of job

Summary and Review of Objectives
✓ Started with high level view of computer hardware and software
✓ Move to solving problems
✓ Examined general computer architecture and software in greater detail
✓ Arduino architecture from high level
✓ Looked at taking problem through process of transforming into something that computer can understand