Cantino Practicing Assignments

Copyright 2017 Jacques de Hooge, GEATEC engineering, www.geatec.com

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

# lesson\_1\_if\_else

## hello

Write a program that displays "Hello, world" on the Arduino IDE Serial Console (AISC).

## name

Write a program that asks for your name and then greets you by name on the AISC.

## age

Write a program that asks for your age and then displays a text on the AISC that depends on the answer.

## age\_chained\_ifs

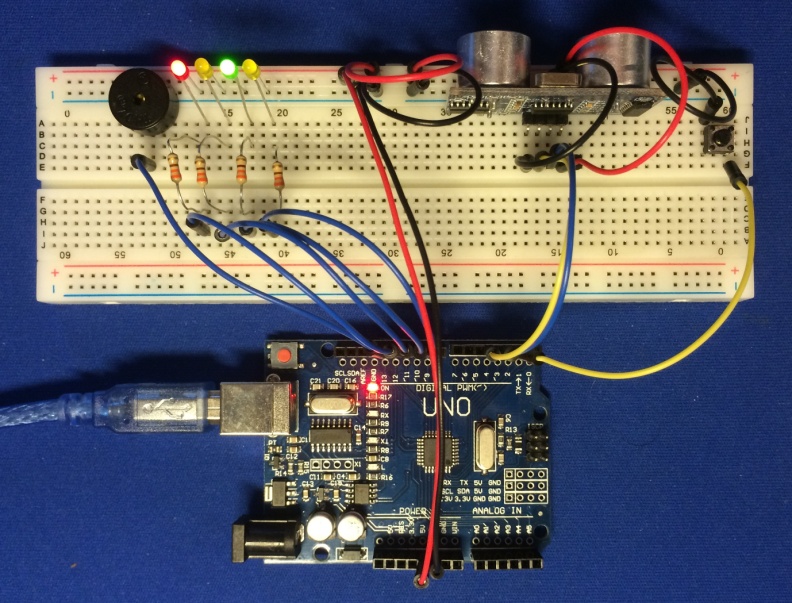
Write a program that asks for your age and gives differen answers at least 4 age categories, using if..elif..elif..else.., using the AISC.

## age\_nested\_ifs

Write a program that asks for your age and gives differen answers at least 4 age categories, using ifs nested in other ifs, elses or elifs, using the AISC.

## switch\_state

Construct the Experimentation Hardware (EH) as shown below (image file hardware.jpg):



Write a progam that uses switches to construct 2 state machines, each switching state at their own pace, but both in a single process. One state machine controls a red/yellow/green traffic light changing state once per 1021 ms, the other controls a yellow blinking light changing state once per 281 ms. Note that these numbers deliberately don't have a simple ratio.

# lesson \_2\_repetition

## table\_of\_three

Make a program that shows the multiplication table of 3 on the AISC, using a for-loop.

## print\_tables

Make a program that prints the multiplication tables from 1 to 10 on the AISC, using nested for-loops.

## ask\_tables

Make a program that asks for the answers of the multiplication tables from 1 to 10, repeating the question until the answer is correct, using a.o. a do..while loop.

## control\_loop

Write a program using a while loop (not a do..while) to have a LED blink and then each time asks (using the AISC) if it should blink again.

# lesson\_3\_functions

## value\_parameters

Make a program defining and calling a function with that makes a LED blink, utilizing the EH.

The function should the current state and the blink time as parameters.

Assignment to the state variable should happen outside the function.

## return\_value

Make a program resembling the first one, but returning the altered state.

The main program assigns this altered state to its local (inside main) state variable.

At the next call this altered state variable is passed in again as a value parameter.

## reference\_parameters

Make a program defining and calling a function with that makes a LED blink, utilizing the EH.

The function should the current state and the blink time as parameters.

Assignment to the state variable should happen inside the function.

# lesson\_4\_arrays

## morse\_encoder

Write a morse encoder using an array of zero terminated strings containing the Morse alphabet in dots and dashes.

Make it accept an arbitrary text via the AISC.

The use the buzzer on the EH to beep the Morse signal accordingly.

## sequencer

Write a 4 channel sequencer with eight timesteps per channel, using nested arrays.

The outer array holds 4 channels, the inner 4 arrays each hold 8 timesteps.

Connect each channel to a LED on the EH.

The user should be able to input an 8 step on-off pattern for each channel using the AISC.

The LEDS on the EH then start blinking according to the patterns entered.

The blinking patterns are repeated indefinitely.

## vector\_arrays

Write a program to add 3D vectors and show them on the AISC.

Each vector is represented by an float array.

# lesson\_5\_modules

## vector\_structs

Write a program to add 3D vectors and show them on the AISC.

Each vector is represented by a struct with an x, y and z field.

## vector\_modules

Put the struct declaration and the method definitions and declarations in a separate module,

that is used by the main module.

All declarations (as opposed to definitions) must be in a header that is included by both.

In general make it a habit to have a separate header for each module.

# lesson\_6\_classes

## dog\_encapsulation

Write a Dog class, with the name of the dog as construction parameter.

Dogs store their name and have a makeSound and a tellName method.

Instantiate some dogs and let makeSound and tellName print their output on the AISC.

(So making a sound is just printing it, although you may additionally use the EH buzzer.)

## dog\_modules

Write a program simular to the first one, but put the Dog class in a separate module.

Be sure to put all declarations in dog.h and all definitions in dog.cpp.

## alternating\_leds

Make a Led class that encapsulates a LED.

The pin index of the LED is supplied at construction.

Objects of this class remember the state of the LED and its pin index.

Instantiate 2 Led objects and use the EH to make 2 LED's blink alternatingly,

i.e. if one of them is off, the other must be on.

## device\_encapsulation

In addition to class Led, write encapsulation classes Switch, UltrasoundSensor and Buzzer. The ultrasound sensor will only be used to switch something on or off.

A buzzer e.g. knows its frequency, an ultrasound sensor will know its switching distance (the maximum distance to the reflector that will switch the state of the class to 'true').

# lesson\_7\_inheritance

## dog\_inheritance

Have class PoliceDog inherit from class Dog.

It has an extra method grabBurglar, that prints "Got you!!" to the AISC.

Instantiate dogs and police dogs and have them do their things by calling their methods.

## device\_inheritance

Have all previously constructed device encapsulation classes inherit from a class Device.

Move common facilities from the individual device encapsulation classes to Device.

Pay attention to the API of Device, that should reflect commonalities between the devices.

# lesson\_8\_polymorphism

## animal\_polymorphism

Make a class hierarchy of several kinds (species, classes) of animals.

Make an array of Animal\* and populate it with pointers to several kinds of animals.

Call a virtual method upon them in a loop, that does different things for different animal types.

This may well be the method makeSound, since each animal kind makes a different sound.

## device\_polymorphism

Give the common Device baseclass a method that will write the state to the AISC.

Put pointers to the Devices into a polymorphic datastructure and have them each report their state to the AISC in ever cylce of the main control loop.

# lesson\_9\_operator\_overloading

## vector\_operators

Make and use a Vector class with overloaded [] and + operators.

## device\_operators

Overload the << operator to enable devices to write their state to the AISC as follows:

cout << \*aLed << \*anotherLed << \*anUltrasoundSensor << \*aBuzzer << \*aSwitch << \*anotherSwitch << endl;

Note that all names behind the \* are pointers. If you find this confusing, postfix them with P, so e.g. aLedP. Don't make a lasting habit of this. Coding datatypes in variable names is to be considered obsolete once you get more experience.

# lesson\_10\_dynamic\_memory

## device\_dynamic

Fill a polymorphic array with pointers to dynamically allocated devices.

Let them operate in cooperation like before.

Make a finite control loop, e.g. executed 10000 times.

At the end of the program, deallocate the devices by calling delete upon them.

N.B. Virtual destructors don't work well with Arduino C++.

# lesson\_11\_templates

## device\_templates

Have classes Sensor and Actuator both inherit from device.

Have the individual device encapsulation classes inherit either from Sensor or from Actuator.

Make a Stack <T> class template.

Instantiate a Stack <Sensor> and a Stack <Actuator>, holding polymorphic pointers to instances of the individual device encapsulation classes.

Let the individual devices cooperate and function as in the examples before.