```
/*
Purix Solar Cooling Control System
An output conditioning function that returns two variables, one for the physical output and one for
printing purposes, maybe a little clumsy but versatile.
Design by cTOPconsult, 2013
*/


    //Defining the output pins
    const int F1_c = 9; //Use this with 0-10V output for external Triac module
    const int F1_r = 6; //Use this with STP Triac modulator (Synchronous Time Proportional)


    //Defining all the variables used
    int Fan = 0; //the "real" output variable used inside main calculations, range 0-1023

    int Fan_c = 0; //actual output sent scaled to the physical outputs, range 0-255
    int Fan_tr = 0; //actual output sent scaled to the physical outputs, range 0-255

    float Fan_p = 0; //output value for various print-out purposes


    int Fan_x = 0; //manual control codes 0, 1, 2 or 3, to over-write outputs
    int setpoint_c=0; //overwrite value for Fan

    int Fan_r = 0;



    void setup()
    {
        //Setting the output pin modes
        pinMode(F1_c, OUTPUT);//Use this with 0-10V output for external Triac module
        pinMode(F1_r, OUTPUT);//Use this with STP Triac modulator


    } //void setup finished




    void loop()
        /*
              .
              .
              .
              .a lot of things going on to calculate the “Fan” value
              .a lot of things going on to calculate the “Fan” value
              .a lot of things going on to calculate the “Fan” value
              .a lot of things going on to calculate the “Fan” value
              .a lot of things going on to calculate the “Fan” value
              .
              .the bypass mode “Fan_x” is controlled via serial commands
              .the bypass value “setpoint_c” is controlled via serial commands
              .
              .
        */

        // We use “outcon” for 0-10V output to external Triac module
        outcon(Fan_c, Fan_p, Fan,1023,0,1023,true,setpoint_c,Fan_x,2,1);

        // We also use “outcon” to convert to an interim variable for the STP modulator
        outcon(Fan_tr, Fan_p, Fan,1023,0,1023,false,setpoint_c,Fan_x,1,1);

        /*
              .
              .
              .STP-modulator receives “Fan_tr” and outputs Fan_r”
              .
        */


        // Finally we write to the physical outputs
        analogWrite(F1_c, Fan_c); //0-10V output for external Triac module
        analogWrite(F1_r, Fan_r); //STP Triac modulator output


    } // void loop finished
```

```
////////////////////////////////////////////////////////////////////////////////////
/////// General purpose function to scale, invert and bypass any output ////////////
////////////////////////////////////////////////////////////////////////////////////
/*
void outcon(int &out_point, int &print_point, int var, long scale, long minimum,
                  long maximum, boolean invert, int man_pct, int bypass, int mode, int p_mode)
   "&out_point" is the global variable receiving the output to the hardware
   "&print_point" is the global variable receiving the output for printing
   "var" is the data input, e.g. the output from a PID-regulator
   "scale" is the expected max value of "var", e.g. 1023 matching the analog input full scale
     NOTE: "scale" >= "var" at any time
   "minimum" and "maximum" define the expected input range within "scale",
     NOTE: "minimum" >= 0 "maximum" <= "scale" and "minimum" < "maximum"
   "invert" TRUE --> inversion of the scaled output
   "man_pct" is an alternative input value in % of scale, usually a manual test overwrite
   "bypass" is a code to determine how output will be bypassed or not
       "0" --> No bypassing. "var" will be used as input
       "1" --> Output is set to zero (or "minimum" in mode "0"), disregarding "var"
       "2" --> Output is set to "scale" (or "maximum" in mode "0"), disregarding "var"
       "3" --> Output is set to the % of "scale" defined in "man_pct"
   "mode" selects the way "outcon" operates with the hardware value
       "0" --> Full functionality: Scale, invert, bypass and fit to 0-255 output for Arduino
       "1" --> Only bypass and invert functions (no 0-255 fit). "minimum" and "maximum" not used
       "2" --> Only bypass and invert functions but with 0-255 fit for direct Arduino output
   "p_mode" selects the way "outcon" operates with the print-value
       "0" --> No action, no conversion
       "1" --> Outputs 0-100% of "scale"
       "2" --> Outputs 100-0% of "scale", inverted
       "3" --> Outputs -100+100% of "scale", with reference to middle of "scale"
       "4" --> Converts TMP36 input to dgrC
       "5" --> Converts TMP35 input to dgrC
       "6" --> Converts TMP37 input to dgrC
   NOTE: Not very many input size error-checks!
   We use floating point inside calculator to reduce error
   Inputs and output are normal integer
   Output is automatically sized to the 8-bit output of an Arduino Mega
   NOTE: The clever person can clean up the excessive use of internal variables :-)
*/
```

```
   void outcon(int &out_point, float &print_point, int var, long scale, long minimum, long maximum,
boolean invert, int man_pct, int bypass, int mode, int p_mode)
    {

      long span, factor, divisor, converted; //used internally in scaling calculations
      float var_x, var_p;

        switch (bypass) {
        case 0:  //no action
          var_x = var;
          break;
        case 1: //force to zero
          var_x = 0;
          break;
        case 2: //force to full
          var_x = scale;
          break;
        case 3: //force to alternative (man_pct) input in % of scale
          var_x = (scale * man_pct) / 100;
          break;
        default: //no action
          var_x = var;
          break;
        }

        switch (p_mode) {
        case 0:  //no action
          var_p = var_x;
          break;
        case 1: // % of scale
          var_p = (var_x * 100) / scale;
          break;
        case 2: // inverted % of scale
          var_p = 100 - (var_x * 100) / scale;
          break;
        case 3: // +/- % of half-scale
          var_p = ((var_x * 200) / scale) - 100;
          break;
        case 4:  //TMP36
          var_p = ((var_x * scale / 1023) - 200) / 4;
          break;
        case 5:  //TMP35
          var_p = (var_x * scale / 1023) / 4;
          break;
        case 6:  //TMP37
          var_p = (var_x * scale / 1023) / 8;
          break;
        default: //no action
          var_p = var_x;
          break;
        }

      if (mode == 0) { //Designed primarily for special cTOPc-TRIAC hardware
        if (maximum > scale) { //in TRIAC application this can occur
            maximum = scale; //set it down to maximum for safety
          }
        span = maximum - minimum; //what is the new span we define
        factor = span / scale; //how big a part of the scale is the new span
        converted = (var_x * factor) + minimum; //the new value including offset
      }
      else {
        converted = var_x;
      }

      if (mode == 2 || mode == 0) { //modes 0 an 2 include scale to 0-255 for arduino
        divisor = (scale + 1) / 256; //how much should we divide with to get a 0-255 output
      }
       else {
        divisor = 1;
       }

      if (invert == true) //Inversion of signal can always be used
      {
        converted = (scale - converted ) / divisor;
      }
      else {

        converted = converted / divisor;
      }

      out_point = converted; //we return the converted value for hardware outpu
      print_point = var_p; //we return the value for the various printout functions
    }
```