```c
#include "contiki.h"
#include <stdio.h> /* For printf() */
#include <random.h>  /* For random_rand() */
#include <string.h> /* For memcpy() */
#define URN  6891988  // specify your URN here

int d1(float f) // Integer part
{
  return((int)f);
}
unsigned int d2(float f) // Fractional part
{
  if (f>0)
    return(100*(f-d1(f)));
  else
    return(100*(d1(f)-f));
}

float *getTempValues(float *floatArray)
{
  int i;
  for (i=0; i<15; i++)
  {
    unsigned short r = random_rand();
    float randomNum = 50 * ((float)r/RANDOM_RAND_MAX) + 10;
    floatArray[i] = randomNum;
  }
  return floatArray;
}

void printArray(float *array)
{
  printf("[");
  int i;
  for (i=0; i<14; i++)
    printf("%d.%d, ", d1(array[i]), d2(array[i]));
  printf("%d.%d]\n", d1(array[i]), d2(array[i]));
}



void bubble_sort(const float *array, float *sorted_array, int length) {
  int i, j;
  memcpy(sorted_array, array, length * sizeof(float));
  for (i = 0; i < length - 1; i++) {
    for (j = 0; j < length - i - 1; j++) {
      if (sorted_array[j] > sorted_array[j + 1]) {
        float temp = sorted_array[j];
        sorted_array[j] = sorted_array[j + 1];
        sorted_array[j + 1] = temp;
      }
    }
  }
}

float calculate_median(float *array, int length) {
    float sorted_array[length];
    bubble_sort(array, sorted_array, length);

    if (length % 2 == 0) {
        return (sorted_array[length/2 - 1] + sorted_array[length/2]) / 2.0;
    } else {
        return sorted_array[length/2];
    }
}
```

```
66
67 static float sqrt_approx(float ssd)
68 {
69     float error = 0.001; // Error tolerance for Babylonian method
70     float x = ssd;       // Initial guess for square root
71     float difference;
72     int i;
73
74     if (ssd == 0)
75     {
76         return 0.0; // No variance
77     }
78
79     for (i = 0; i < 50; i++)
80     {
81         x = 0.5 * (x + ssd / x);
82         difference = x * x - ssd;
83         if (difference < 0)
84         {
85             difference = -difference;
86         }
87         if (difference < error)
88         {
89             break;
90         }
91     }
92     return x;
93 }
94
95
96 float calculate_pcc(float *arrayX, float*arrayY, float averageX, float averageY){
97   float sumX = 0;
98   float sumY = 0;
99   float sumXY = 0;
100  float sumX2 = 0;
101  float sumY2 = 0;
102  int i;
103  for (i=0; i<15; i++)
104  {
105    sumXY = sumXY + (arrayX[i] - averageX) * (arrayY[i] - averageY);
106    sumX2 = sumX2 + (arrayX[i] - averageX) * (arrayX[i] - averageX);
107    sumY2 = sumY2 + (arrayY[i] - averageY) * (arrayY[i] - averageY);
108  }
109  float pcc = (sumXY)/ sqrt_approx(sumX2 * sumY2);
110  return pcc;
111 }
112
113
114 /*---------------------------------------------------------------------------*/
115 PROCESS(sensor_reading_process, "Sensor reading process");
116 AUTOSTART_PROCESSES(&sensor_reading_process);
117 /*---------------------------------------------------------------------------*/
118 PROCESS_THREAD(sensor_reading_process, ev, data)
119 {
120   static struct etimer timer;
121   PROCESS_BEGIN();
122
123   // extract your seed
124   unsigned long a = URN;
125   unsigned int a3 = a/10000;
126   unsigned int a4 = a - a3*10000;
127   printf("Your URN is: %d%d\n", a3, a4);
128   random_init(a4);
129
130   // generate 15 random temperature values
131   float temp15[15];
```

```
  getTempValues(temp15);

  // print array B and the average
  int i;
  float average = 0;
  for (i=0; i<15; i++)
    average = average + temp15[i];
  average = average / 15;
  printf("B = ");
  printArray(temp15);
  printf("Average for B = %d.%d\n", d1(average), d2(average));

  // determine the median (implemente your code here)
  float median;
  median = calculate_median(temp15, 15);

  printf("Median = %d.%d\n", d1(median), d2(median));

  // determine array E (implemente your code here)
  float beta = 0.7;
  float EMA[15];
  EMA[0] = temp15[0];

  for (i=1; i<15; i++)
  {
    EMA[i] = beta * temp15[i] + (1 - beta) * EMA[i-1];
  }

  printf("E = ");
  printArray(EMA);
  float averageE = 0;
  for (i=0; i<15; i++)
    averageE = averageE + EMA[i];
  averageE = averageE / 15;
  // printf("Average for E = %d.%d\n", d1(average), d2(average));
  // determine Pearson Correlation Coefficient (implemente your code here)
  float pearsonCC = calculate_pcc(temp15, EMA, average, averageE);

  printf("Pearson Correlation Coefficient = %d.%d\n", d1(pearsonCC), d2(pearsonCC));


  // do not touch the following
  etimer_set(&timer, CLOCK_CONF_SECOND);   // use this setting for 1 second duration

  while(1)
  {
    PROCESS_WAIT_EVENT_UNTIL(ev=PROCESS_EVENT_TIMER);
    etimer_reset(&timer);
  }
  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```