Code für HC-SR04 Entfernungsmesser (Obst)

```
#include <wiringPi.h>
#include <time.h>
#include <stdio.h>

double timedelay;
clock_t start, end;
double result;

int main(){
  read();
}

int read(){
       while(1){
         pinMode(15,OUTPUT);

         start=clock();
         digitalWrite(15,HIGH);
         pinMode(15,INPUT);
         while(digitalRead(15)!=1);
         end=clock();
         timedelay=((double) (end-start));

         result=17150*timedelay;

         printf("Distance: %f ",result);
       }
}
```

Hier wird zuerst ein Signal vom HC-SR04 ausgegeben (`digitalWrite(15,HIGH);`) und danach wird die Zeit gemessen bis das Signal wieder zurückkommt.

Code für MPU-6050 Gyrosensor (Obst)

```
#include <wiringPiI2C.h>
#include <wiringPi.h>
#include <stdio.h>
#include <math.h>

int fd;
int acclX, acclY, acclZ;
int gyroX, gyroY, gyroZ;
double acclX_scaled, acclY_scaled, acclZ_scaled;

int read_word_2c(int addr)
{
int val;
val = wiringPiI2CReadReg8(fd, addr);
val = val << 8; //bits werden um 8 nach links verschoben
val += wiringPiI2CReadReg8(fd, addr+1);
if (val >= 0x8000)
val = -(65536 - val);

return val;
}

double dist(double a, double b)
{
return sqrt((a*a) + (b*b));
}

double get_y_rotation(double x, double y, double z)
{
double radians;
radians = atan2(x, dist(y, z));
return -(radians * (180.0 / M_PI));
}

double get_x_rotation(double x, double y, double z)
{
double radians;
radians = atan2(y, dist(x, z));
return (radians * (180.0 / M_PI));
}

int main()
{
fd = wiringPiI2CSetup (0x69);
wiringPiI2CWriteReg8 (fd,0x6B,0x00);//disable sleep mode

while(1) {

acclX = read_word_2c(0x3B);
acclY = read_word_2c(0x3D);
acclZ = read_word_2c(0x3F);

acclX_scaled = acclX / 16384.0;
acclY_scaled = acclY / 16384.0;
acclZ_scaled = acclZ / 16384.0;

printf("X rotation: %f\n", get_x_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));
printf("Y rotation: %f\n", get_y_rotation(acclX_scaled, acclY_scaled, acclZ_scaled));


delay(100);
}
return 0;
}
```

Code für BME280 Luftfeuchtigkeitsmesser (Obst)

```
#ifndef __BME280_H__
#define __BME280_H__

#define BME280_ADDRESS                0x77

#define BME280_REGISTER_DIG_T1        0x88
#define BME280_REGISTER_DIG_T2        0x8A
#define BME280_REGISTER_DIG_T3        0x8C
#define BME280_REGISTER_DIG_P1        0x8E
#define BME280_REGISTER_DIG_P2        0x90
#define BME280_REGISTER_DIG_P3        0x92
#define BME280_REGISTER_DIG_P4        0x94
#define BME280_REGISTER_DIG_P5        0x96
#define BME280_REGISTER_DIG_P6        0x98
#define BME280_REGISTER_DIG_P7        0x9A
#define BME280_REGISTER_DIG_P8        0x9C
#define BME280_REGISTER_DIG_P9        0x9E
#define BME280_REGISTER_DIG_H1        0xA1
#define BME280_REGISTER_DIG_H2        0xE1
#define BME280_REGISTER_DIG_H3        0xE3
#define BME280_REGISTER_DIG_H4        0xE4
#define BME280_REGISTER_DIG_H5        0xE5
#define BME280_REGISTER_DIG_H6        0xE7
#define BME280_REGISTER_CHIPID        0xD0
#define BME280_REGISTER_VERSION       0xD1
#define BME280_REGISTER_SOFTRESET     0xE0
#define BME280_RESET                  0xB6
#define BME280_REGISTER_CAL26         0xE1
#define BME280_REGISTER_CONTROLHUMID  0xF2
#define BME280_REGISTER_CONTROL       0xF4
#define BME280_REGISTER_CONFIG        0xF5
#define BME280_REGISTER_PRESSUREDATA  0xF7
#define BME280_REGISTER_TEMPDATA      0xFA
#define BME280_REGISTER_HUMIDDATA     0xFD

#define MEAN_SEA_LEVEL_PRESSURE       1013

/*
* Immutable calibration data read from bme280
*/
typedef struct
{
  uint16_t dig_T1;
  int16_t  dig_T2;
  int16_t  dig_T3;

  uint16_t dig_P1;
  int16_t  dig_P2;
  int16_t  dig_P3;
  int16_t  dig_P4;
  int16_t  dig_P5;
  int16_t  dig_P6;
  int16_t  dig_P7;
  int16_t  dig_P8;
  int16_t  dig_P9;

  uint8_t  dig_H1;
  int16_t  dig_H2;
  uint8_t  dig_H3;
  int16_t  dig_H4;
  int16_t  dig_H5;
  int8_t   dig_H6;
} bme280_calib_data;
```

```c
/* Raw sensor measurement data from bme280
*/
typedef struct
{
  uint8_t pmsb;
  uint8_t plsb;
  uint8_t pxsb;

  uint8_t tmsb;
  uint8_t tlsb;
  uint8_t txsb;

  uint8_t hmsb;
  uint8_t hlsb;

  uint32_t temperature;
  uint32_t pressure;
  uint32_t humidity;

} bme280_raw_data;
```

```c
#include <stdio.h>
#include <errno.h>
#include <stdint.h>
#include <time.h>
#include <math.h>
#include <wiringPiI2C.h>
#include "bme280.h"

int main() {

  int fd = wiringPiI2CSetup(BME280_ADDRESS);
  if(fd < 0) {
    printf("Device not found");
    return -1;
  }

  bme280_calib_data cal;
  readCalibrationData(fd, &cal);

  wiringPiI2CWriteReg8(fd, 0xf2, 0x01);   // humidity oversampling x 1
  wiringPiI2CWriteReg8(fd, 0xf4, 0x25);   // pressure and temperature oversampling x 1, mode normal

  bme280_raw_data raw;
  getRawData(fd, &raw);

  int32_t t_fine = getTemperatureCalibration(&cal, raw.temperature);
  float t = compensateTemperature(t_fine); // C
  float p = compensatePressure(raw.pressure, &cal, t_fine) / 100; // hPa
  float h = compensateHumidity(raw.humidity, &cal, t_fine);       // %
  float a = getAltitude(p);                           // meters

  printf("{\"sensor\":\"bme280\", \"humidity\":%.2f, \"pressure\":%.2f,"
    " \"temperature\":%.2f, \"altitude\":%.2f, \"timestamp\":%d}\n",
    h, p, t, a, (int)time(NULL));

  return 0;
}

int32_t getTemperatureCalibration(bme280_calib_data *cal, int32_t adc_T) {
  int32_t var1  = ((((adc_T>>3) - ((int32_t)cal->dig_T1 <<1))) *
     ((int32_t)cal->dig_T2)) >> 11;

  int32_t var2  = (((((adc_T>>4) - ((int32_t)cal->dig_T1)) *
       ((adc_T>>4) - ((int32_t)cal->dig_T1))) >> 12) *
     ((int32_t)cal->dig_T3)) >> 14;

  return var1 + var2;
}

void readCalibrationData(int fd, bme280_calib_data *data) {
  data->dig_T1 = (uint16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_T1);
  data->dig_T2 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_T2);
  data->dig_T3 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_T3);

  data->dig_P1 = (uint16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P1);
  data->dig_P2 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P2);
  data->dig_P3 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P3);
  data->dig_P4 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P4);
  data->dig_P5 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P5);
  data->dig_P6 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P6);
  data->dig_P7 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P7);
  data->dig_P8 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P8);
  data->dig_P9 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_P9);

  data->dig_H1 = (uint8_t)wiringPiI2CReadReg8(fd, BME280_REGISTER_DIG_H1);
  data->dig_H2 = (int16_t)wiringPiI2CReadReg16(fd, BME280_REGISTER_DIG_H2);
  data->dig_H3 = (uint8_t)wiringPiI2CReadReg8(fd, BME280_REGISTER_DIG_H3);
  data->dig_H4 = (wiringPiI2CReadReg8(fd, BME280_REGISTER_DIG_H4) << 4) | (wiringPiI2CReadReg8(fd,
BME280_REGISTER_DIG_H4+1) & 0xF);
  data->dig_H5 = (wiringPiI2CReadReg8(fd, BME280_REGISTER_DIG_H5+1) << 4) | (wiringPiI2CReadReg8(fd,
BME280_REGISTER_DIG_H5) >> 4);
  data->dig_H6 = (int8_t)wiringPiI2CReadReg8(fd, BME280_REGISTER_DIG_H6);
}
```

```
float compensateTemperature(int32_t t_fine) {
  float T  = (t_fine * 5 + 128) >> 8;
  return T/100;
}

float compensatePressure(int32_t adc_P, bme280_calib_data *cal, int32_t t_fine) {
  int64_t var1, var2, p;

  var1 = ((int64_t)t_fine) - 128000;
  var2 = var1 * var1 * (int64_t)cal->dig_P6;
  var2 = var2 + ((var1*(int64_t)cal->dig_P5)<<17);
  var2 = var2 + (((int64_t)cal->dig_P4)<<35);
  var1 = ((var1 * var1 * (int64_t)cal->dig_P3)>>8) +
    ((var1 * (int64_t)cal->dig_P2)<<12);
  var1 = (((((int64_t)1)<<47)+var1))*((int64_t)cal->dig_P1)>>33;

  if (var1 == 0) {
    return 0;  // avoid exception caused by division by zero
  }
  p = 1048576 - adc_P;
  p = (((p<<31) - var2)*3125) / var1;
  var1 = (((int64_t)cal->dig_P9) * (p>>13) * (p>>13)) >> 25;
  var2 = (((int64_t)cal->dig_P8) * p) >> 19;

  p = ((p + var1 + var2) >> 8) + (((int64_t)cal->dig_P7)<<4);
  return (float)p/256;
}


float compensateHumidity(int32_t adc_H, bme280_calib_data *cal, int32_t t_fine) {
  int32_t v_x1_u32r;

  v_x1_u32r = (t_fine - ((int32_t)76800));

  v_x1_u32r = (((((adc_H << 14) - (((int32_t)cal->dig_H4) << 20) -
      (((int32_t)cal->dig_H5) * v_x1_u32r)) + ((int32_t)16384)) >> 15) *
         (((((((v_x1_u32r * ((int32_t)cal->dig_H6)) >> 10) *
        (((v_x1_u32r * ((int32_t)cal->dig_H3)) >> 11) + ((int32_t)32768))) >> 10) +
      ((int32_t)2097152)) * ((int32_t)cal->dig_H2) + 8192) >> 14));

  v_x1_u32r = (v_x1_u32r - (((((v_x1_u32r >> 15) * (v_x1_u32r >> 15)) >> 7) *
           ((int32_t)cal->dig_H1)) >> 4));

  v_x1_u32r = (v_x1_u32r < 0) ? 0 : v_x1_u32r;
  v_x1_u32r = (v_x1_u32r > 419430400) ? 419430400 : v_x1_u32r;
  float h = (v_x1_u32r>>12);
  return  h / 1024.0;
}
```

```
void getRawData(int fd, bme280_raw_data *raw) {
  wiringPiI2CWrite(fd, 0xf7);

  raw->pmsb = wiringPiI2CRead(fd);
  raw->plsb = wiringPiI2CRead(fd);
  raw->pxsb = wiringPiI2CRead(fd);

  raw->tmsb = wiringPiI2CRead(fd);
  raw->tlsb = wiringPiI2CRead(fd);
  raw->txsb = wiringPiI2CRead(fd);

  raw->hmsb = wiringPiI2CRead(fd);
  raw->hlsb = wiringPiI2CRead(fd);

  raw->temperature = 0;
  raw->temperature = (raw->temperature | raw->tmsb) << 8;
  raw->temperature = (raw->temperature | raw->tlsb) << 8;
  raw->temperature = (raw->temperature | raw->txsb) >> 4;

  raw->pressure = 0;
  raw->pressure = (raw->pressure | raw->pmsb) << 8;
  raw->pressure = (raw->pressure | raw->plsb) << 8;
  raw->pressure = (raw->pressure | raw->pxsb) >> 4;

  raw->humidity = 0;
  raw->humidity = (raw->humidity | raw->hmsb) << 8;
  raw->humidity = (raw->humidity | raw->hlsb);
}

float getAltitude(float pressure) {
  // Equation taken from BMP180 datasheet (page 16):
  //  http://www.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf

  // Note that using the equation from wikipedia can give bad results
  // at high altitude.  See this thread for more information:
  //  http://forums.adafruit.com/viewtopic.php?f=22&t=58064

  return 44330.0 * (1.0 - pow(pressure / MEAN_SEA_LEVEL_PRESSURE, 0.190294957));
}
```

Code für Luxmeter (Stix L.)

```c
#include <wiringPiI2C.h>
#include <stdio.h>
int luxmeter(){
	int handle = wiringPiI2CSetup(0x5C);
	wiringPiI2CWrite(handle,0x01);
	wiringPiI2CWrite(handle,0x21);
	int word = wiringPiI2CReadReg16(handle,0x00);
	int lux=((word & 0xff00)>>8) | ((word & 0x00ff)<<8);
	printf("Aktuelle Beleuchtungsstärke in Lux: %d\n",lux);
	return 0;
}
```

Pinbelegung BH1750:

Betriebsmodi

Steuercode	Anweisung	Kommentar
00000000	Power down	Gerät inaktiv
00000001	Power on	Gerät bereit zur Messung
00000111	Reset	setzt das Datenregister zurück; kein Power-down-Modus
00010000	Continuously H-Resolution Mode	kontinuierliches hochauflösendes Messen (Auflösung 1lx, Dauer 120ms)
00010001	Continuously H-Resolution Mode2	kontinuierliches hochauflösendes Messen (Auflösung 0,5lx, Dauer 120ms)
00010011	Continuously L-Resolution Mode	kontinuierliches niedrigauflösendes Messen (Auflösung 4lx, Dauer 16ms)
00100000	One Time H-Resolution Mode	einmaliges hochauflösendes Messen (Auflösung 1lx, Dauer 120ms)
00100001	One Time H-Resolution Mode2	einmaliges hochauflösendes Messen (Auflösung 0,5lx, Dauer 120ms)
00100011	One Time L-Resolution Mode	einmaliges niedrigauflösendes Messen (Auflösung 4lx, Dauer 16ms)
01000*xxx*	Change Measurement Time (High Bit)	relative Messdauer (Bits 7, 6, 5)
011*xxxxx*	Change Measurement Time (Low Bit)	relative Messdauer (Bits 4, 3, 2, 1, 0)

Code für ADS1115 (Stix L.)

```
#include <stdlib.h>
#include <stdio.h>
int ads1115(){
	FILE* dataFromLDR = popen("gpio -x ads1115:120:0x49 aread 121","r");
	int iLDR = 0;
	fscanf(dataFromLDR, "%d",  &iLDR);
	printf("LDR Output: %d\n",iLDR);

	FILE* dataFromNTC = popen("gpio -x ads1115:120:0x49 aread 122","r");
	int iNTC = 0;
	fscanf(dataFromNTC, "%d", &iNTC);
	float fNTCTemperatur = (float)iNTC/1000;
	printf("Temperatur: %.2f\n",fNTCTemperatur);

	return 0;
}
```

Code für PCF8591 (Stix L.)

```
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <pcf8591.h>

#define PCF 120


int pcf8591(){
	int valueLDR;
	int valueNTC;
	wiringPiSetup();

	pcf8591Setup(PCF, 0x48);

	valueLDR = analogRead(PCF+1);
	valueNTC = analogRead(PCF+2);
	//Werte in Fahrenheit!
	printf("%d\n%d\n",valueLDR,valueNTC);
	analogWrite(PCF+1,valueLDR);  //pin 1: LDR
	analogWrite(PCF+2,valueNTC); //pin 2: NTC


	return 0;
}
```

Code für DS18B20 (Wintersteiger)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BASE_FILE "/sys/bus/w1/devices/28-01144fe3b8aa/w1_slave"

int read_ds18b20()
{
  FILE *fp;
  char con[1000];
  char *pch;
  float temp=0;

  fp = fopen(BASE_FILE, "r");
  if(!fp)
       return 1;

  while(fgets(con, 1000, fp)!=NULL){

              //searches for temperature in document
              pch=strstr(con,"t=");
              if(pch!=NULL){

                     pch+=2;
                     temp = (float) strtod(pch,NULL);
                     //divide by 1000 because temperature is multiplied by this number
                     temp /= 1000;

                     printf("DS18B20 Temperature: %.2f °C\n", temp);
              }
  }

  fclose(fp);
  return 0;

}

int main( void )
{
    read_ds18b20();
    return(0);
}
```

BASE_FILE gibt den Pfad zu dem File, indem schon automatisch durch One Wire die Temperatur hineingeschrieben wird.

Ich suche mir dann die Stelle, an der die Temperatur steht und hole sie mir. Da sie die Temperatur * 1000 anzeigt, muss ich sie wieder durch diesen Wert dividieren. Am Schluss habe ich dann die gewünschte Temperatur.

Code für DHT11 (Wintersteiger)

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define MAX_TIMINGS	85
#define DHT_PIN		11

int data[5] = { 0, 0, 0, 0, 0 };

void read_dht_data()
{
	uint8_t laststate	= HIGH;
	uint8_t counter		= 0;
	uint8_t j		= 0, i;

	data[0] = data[1] = data[2] = data[3] = data[4] = 0;

	/* pull pin down for 18 milliseconds */
	pinMode( DHT_PIN, OUTPUT );
	digitalWrite( DHT_PIN, LOW );
	delay( 18 );

	/* prepare to read the pin */
	pinMode( DHT_PIN, INPUT );

	/* detect change and read data */
	for ( i = 0; i < MAX_TIMINGS; i++ )
	{
		counter = 0;
		while ( digitalRead( DHT_PIN ) == laststate )
		{
			counter++;
			delayMicroseconds( 1 );
			if ( counter == 255 )
			{
				break;
			}
		}
		laststate = digitalRead( DHT_PIN );

		if ( counter == 255 )
			break;

		/* ignore first 3 transitions */
		if ( (i >= 4) && (i % 2 == 0) )
		{
			/* shove each bit into the storage bytes */
			data[j / 8] <<= 1;
			if ( counter > 16 )
				data[j / 8] |= 1;
			j++;
		}
	}
```

```
        /*
         * check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
         * print it out if data is good
         */
        if ( (j >= 40) &&
             (data[4] == ( (data[0] + data[1] + data[2] + data[3]) & 0xFF) ) )
        {
               float h = (float)((data[0] << 8) + data[1]) / 10;
               if ( h > 100 )
               {
                      h = data[0];  // for DHT11
               }
               float c = (float)(((data[2] & 0x7F) << 8) + data[3]) / 10;
               if ( c > 125 )
               {
                      c = data[2];  // for DHT11
               }
               if ( data[2] & 0x80 )
               {
                      c = -c;
               }
               float f = c * 1.8f + 32;
               printf( "Humidity = %.1f %% Temperature = %.1f *C (%.1f *F)\n", h, c, f );
        }else  {
               printf( "Data not good, skip\n" );
        }
}

int main( void )
{
        printf( "Raspberry Pi DHT11/DHT22 temperature/humidity test\n" );

        if ( wiringPiSetup() == -1 )
               exit( 1 );

        while ( 1 )
        {
               read_dht_data();
               delay( 2000 ); /* wait 2 seconds before next read */
        }

        return(0);
}
```

Dieser C-Code holt sich, mehrmals die Daten vom DHT11 und überprüft, ob diese ok sind, falls nicht bekommt man eine spezielle Ausgabe. Es werden bis zu 85 mal neue Werte geholt