



INSTITUTO DE FÍSICA

1º minicurso de Arduino no IFUSP

4 a 22 de maio de 2015
Prof. Alexandre Suaide

Arduino é uma plataforma eletrônica para prototipagem flexível,
de baixo custo, fácil de usar e aberta.

Inscrições abertas até 17 de abril
Mais informações sobre o minicurso e inscrições em

<http://e.usp.br/2ia>



Conteúdo de hoje

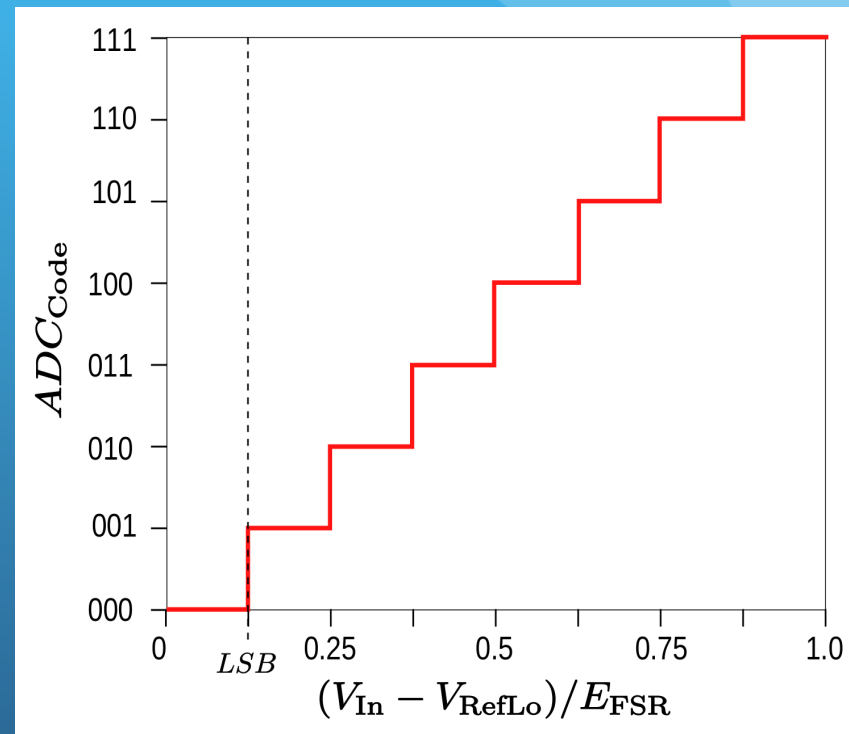
- ADC's
- Sensores, shields, bibliotecas
- Interface RS232 e enviando dados ao computador
 - Vou falar sobre isto somente na oficina mas os slides estão disponíveis.
- A memória do ATMEGA
 - FLASH, EEPROM e RAM

ADC - Analog to Digital Converter

- É um dispositivo que converte uma grandeza analógica (em geral tensão elétrica) em uma grandeza digital
 - DAC - Digital to Analog Converter faz o oposto
- Converter um sinal analógico em digital envolve vários conceitos
 - Resolução
 - Razão sinal para ruído
 - Jitter
 - Taxa de amostragem
 - Linearidade
 - Range dinâmico (número efetivo de bits)
 - Etc.

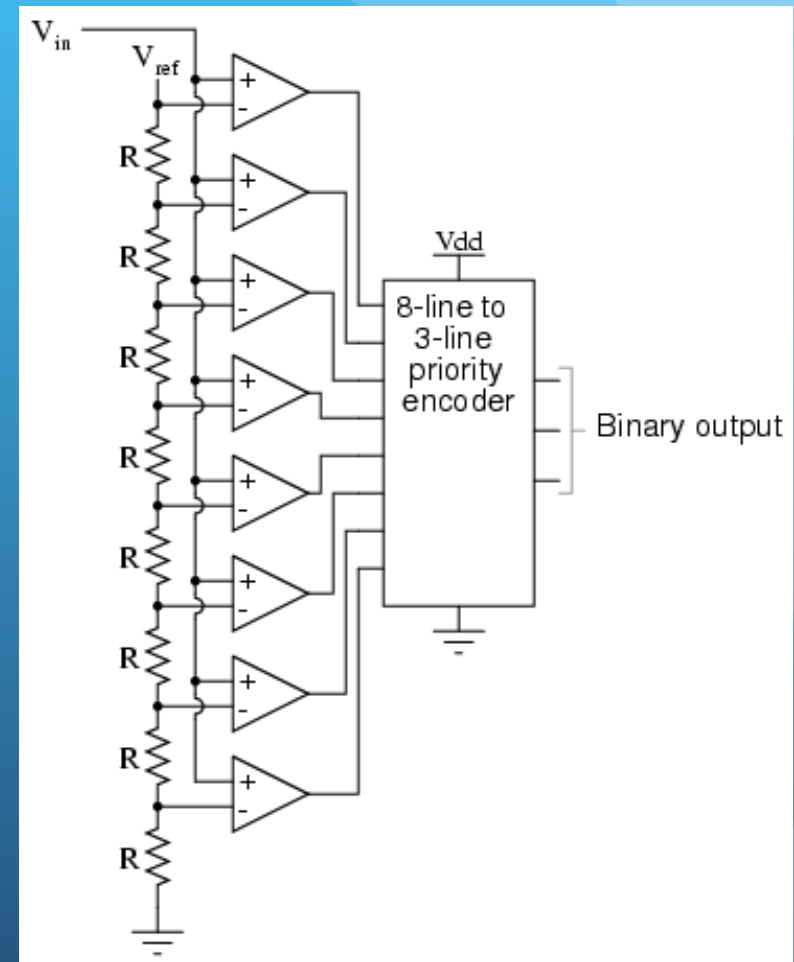
Resolução do ADC

- Quantidade de valores discretos que podem ser gerados pelo ADC
 - Em geral dado em número de bits
 - Ex: 10 bits de resolução podem gerar 1024 valores diferentes para a grandeza analógica
- ADC do Arduino UNO tem 10 bits de resolução.



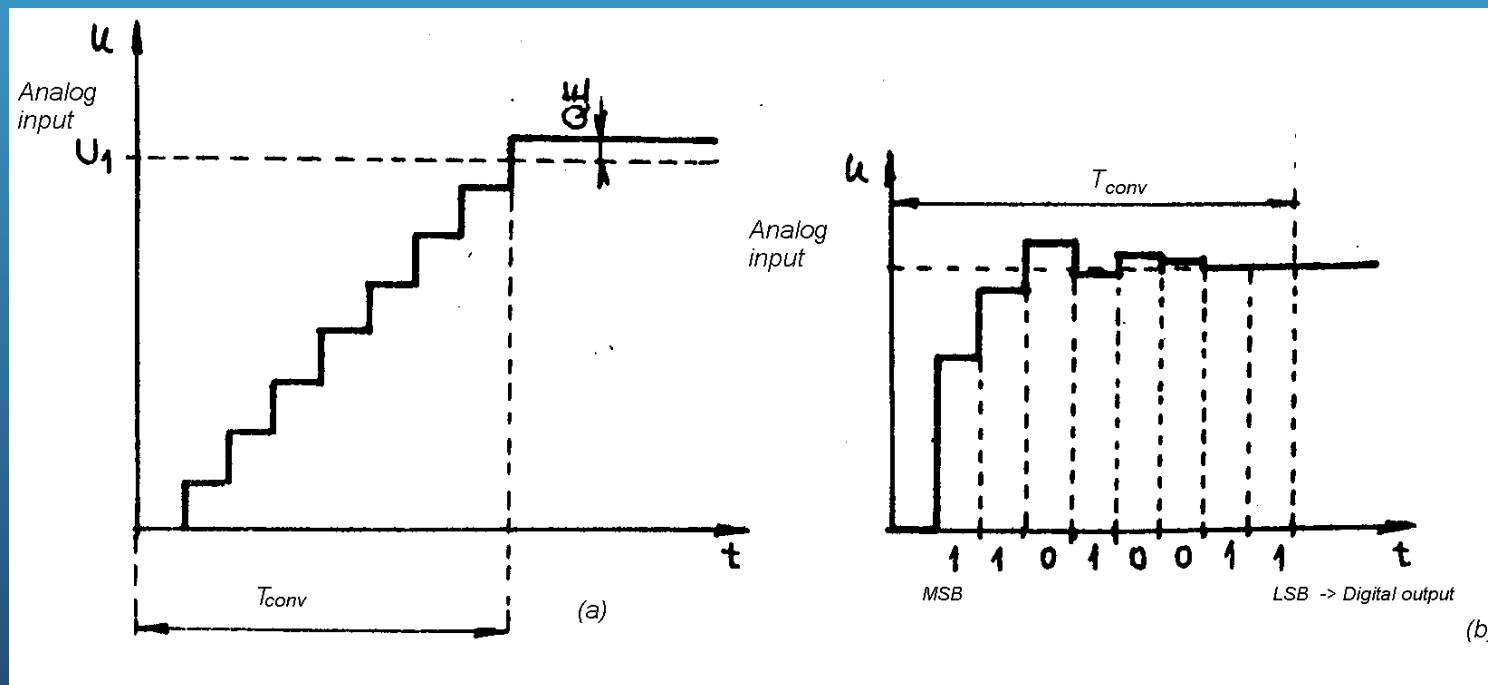
Como um ADC funciona

- Muitas técnicas diferentes
 - FLASH ADC - muito rápido mas muito caro e baixa resolução (muitos componentes)
 - Sigma-Delta - Integra-se o sinal e compara-se com terra
 - Dual-slope - Carga e descarga de um capacitor (mede-se o tempo de descarga)



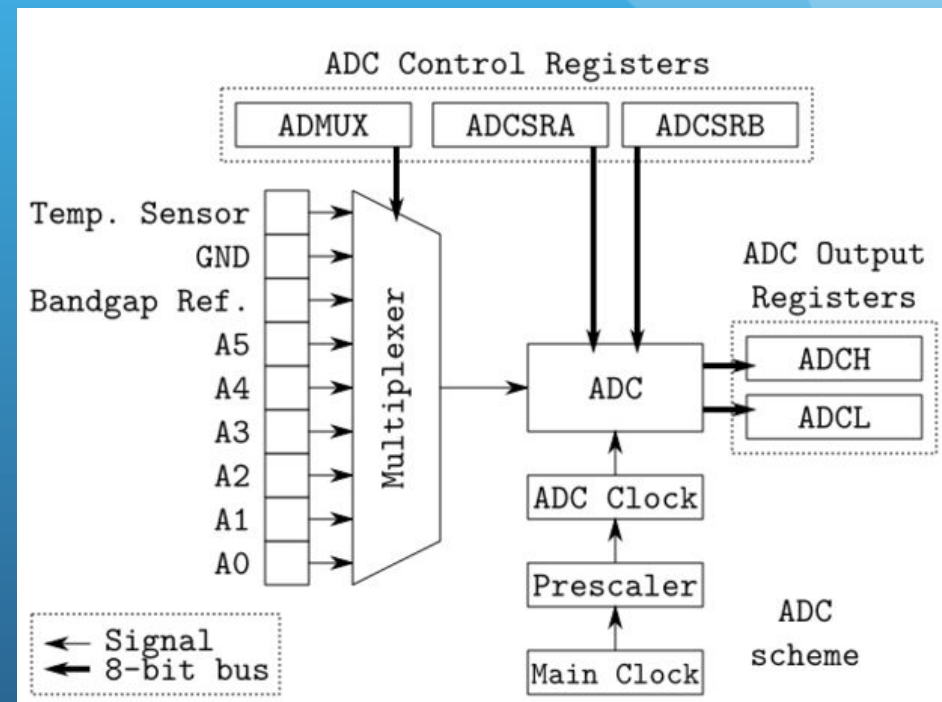
Como um ADC funciona - aproximações sucessivas

- Liga o bit mais significativo e compara ao valor analógico
 - Decide-se se este bit é 0 ou 1
- Vai para o bit seguinte e repete-se a comparação



ADC no Arduino

- Aproximações sucessivas de 10 bits
- Uma conversão de ADC leva cerca de 100 μ s.
- Arduino possui 6 ADCs (A0 - A5)
 - Na verdade possui um só que é multiplexado internamente
 - Não é possível ler simultaneamente 2 canais diferentes

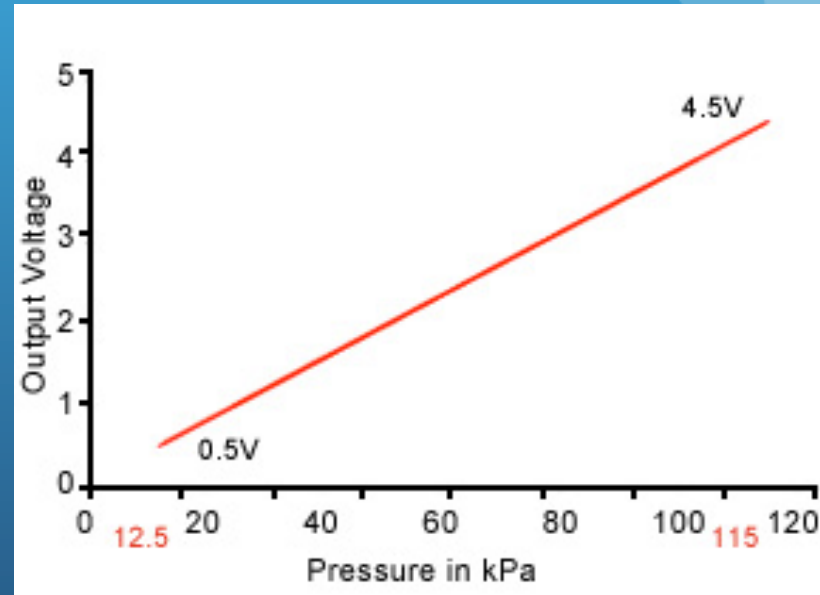


Funções

- `analogRead(ADC)` - Converte e lê o valor analógico no pino ADC
- `analogReference(valor)` - seta o valor de referência para fundo de escala do ADC.
 - `DEFAULT` - Valor padrão da placa de Arduino (5.0 ou 3.3V)
 - `INTERNAL` - Valor interno da ATMEGA (1.1 ou 2.56V)
 - `INTERNAL1V1` - 1.1V (somente Arduino MEGA)
 - `INTERNAL2V56` - 2.56V (somente Arduino MEGA)
 - `EXTENAL` - Valor aplicado no pino AREF ($0 < V < 5V$)

Sensores

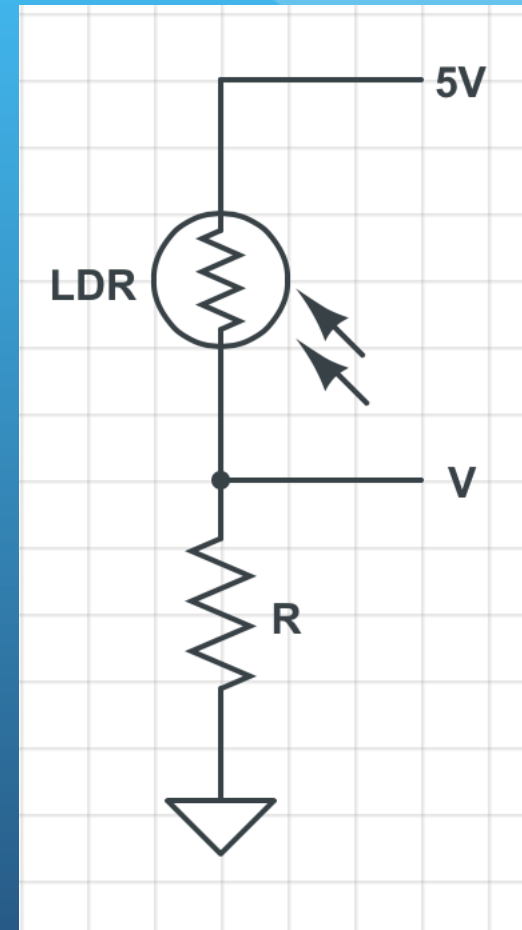
- Um equipamento que transforma uma quantidade física em uma quantidade elétrica (em geral, tensão elétrica)
- Em geral um sensor precisa ser montado em um circuito elétrico para funcionar



Ex: Foto-resistor

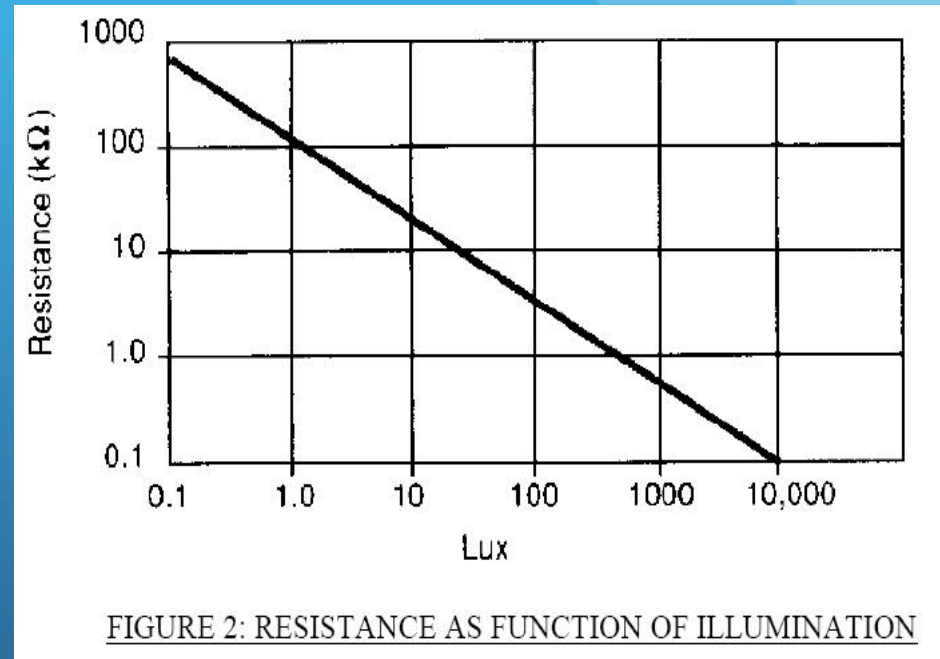
- O LDR (Light Dependent Resistor) é um resistor feito de um material no qual a resistência depende da quantidade de luz incidente
- Pode ser usado como sensor de luz
 - Circuito elétrico simples
 - Medindo V podemos determinar a resistência do LDR

$$V = 5 \frac{R}{R_{LDR} + R}$$



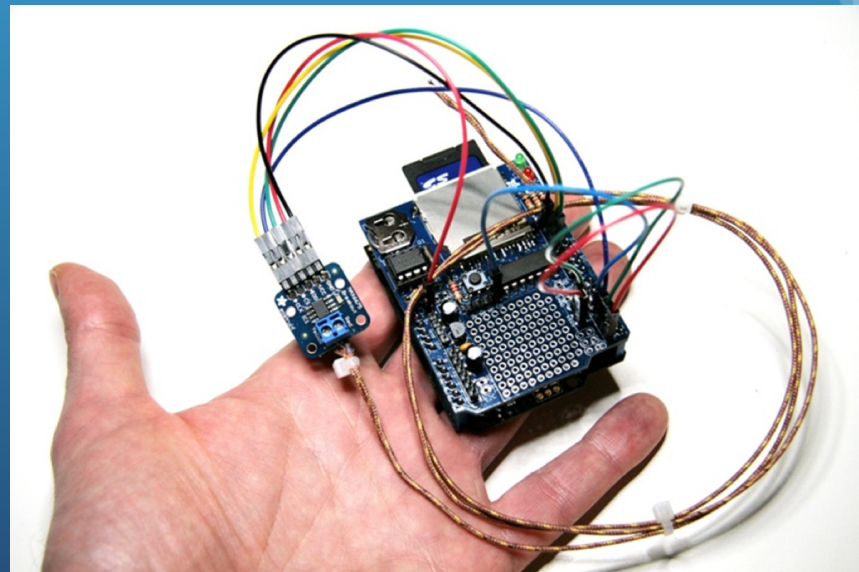
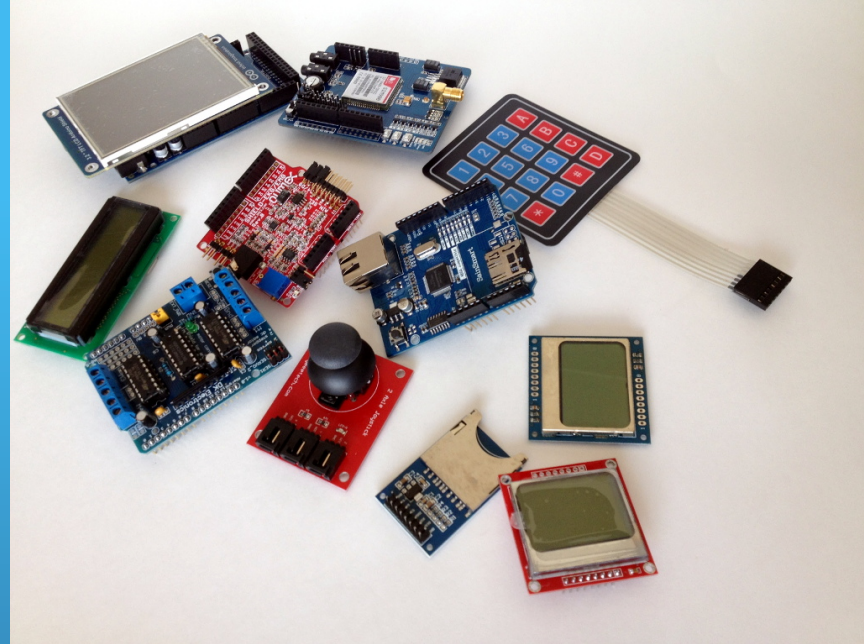
Calibração

- Depois de medir a tensão queremos determinar a quantidade de luz incidente no LDR
- Calibração do sensor
 - Fornecido pelo fabricante (datasheet)
 - Ou nós mesmos fazemos



Shields

- Placas com circuitos elétricos que podem ser conectadas diretamente no Arduino para realizar tarefas específicas
 - Sensores
 - Telas de LCD
 - Memórias
 - Atuadores (relês, botões, etc.)
- <http://playground.arduino.cc/Main/SimilarBoards#goShie>



Bibliotecas

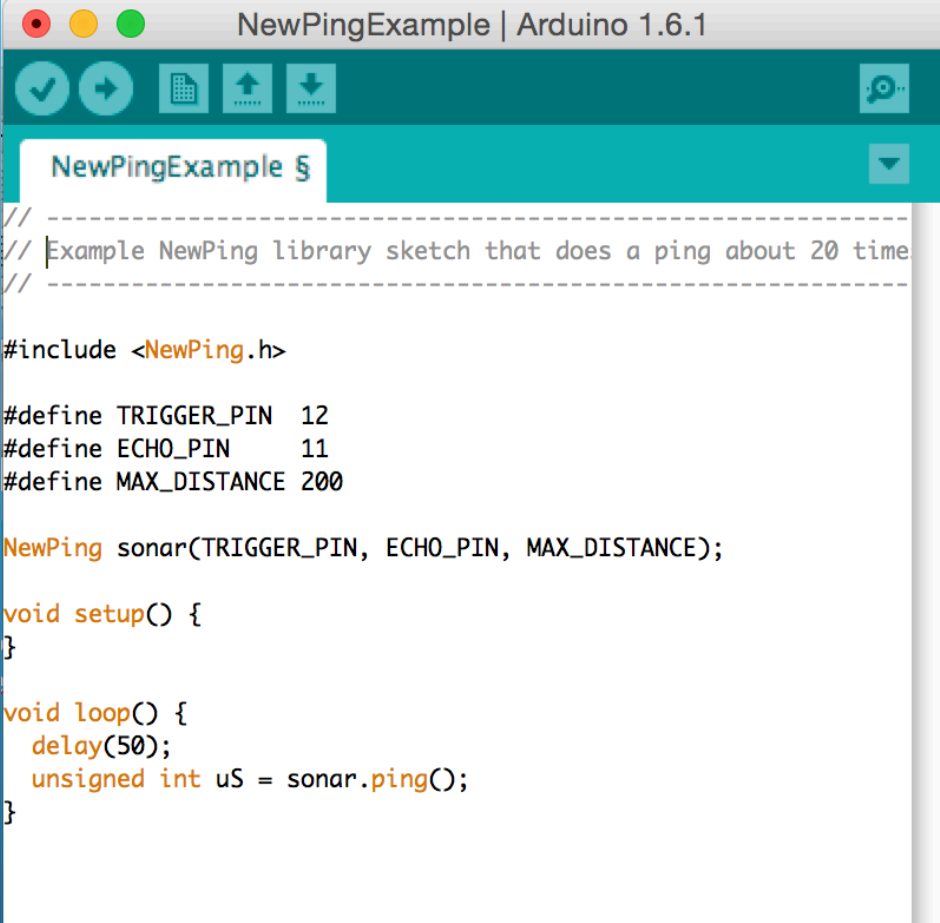
- Pacotes de programas que estendem a capacidade de programação do Arduino
 - Em geral são classes em C++ que desempenham alguma função específica
 - Controlar um “shield”, tornando mais fácil o seu uso
 - Ex: NewPing -- sensor de ultrassom
 - Biblioteca já prepara toda a configuração dos pinos do shield, faz ele funcionar e já retorna a distância do sensor ao objeto
 - Ex: LiquidCrystal - tela de LCD
 - Facilita o envio de mensagens para uma tela de LCD sem precisarmos entender como a tela funciona a nível de circuito.

Escrevendo uma biblioteca

- Escrever uma biblioteca é escrever um código em c++
 - Em geral escreve-se uma classe. Fica mais fácil usar linguagem orientada a objetos
 - Ainda mais porque shields são objetos que você conecta no Arduino.
- Um tutorial, passo a passo, de como escrever uma biblioteca pode ser encontrado em
 - <http://www.arduino.cc/en/Hacking/LibraryTutorial>

Usando uma biblioteca

- Faz-se o include no cabeçalho do programa
- Instancia-se o objeto
 - Em geral faz este objeto global para poder usa-lo em todos os métodos do seu programa
- Usa-se a documentação da biblioteca para saber o que fazer.



```
NewPingExample | Arduino 1.6.1
NewPingExample §
// -----
// Example NewPing library sketch that does a ping about 20 times
// -----
#include <NewPing.h>

#define TRIGGER_PIN 12
#define ECHO_PIN 11
#define MAX_DISTANCE 200

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
}

void loop() {
  delay(50);
  unsigned int uS = sonar.ping();
}
```

Estas coisas consomem memória

- A criação de objetos, uso de variáveis, tamanho do programa, etc. consomem memória.
- Microcontroladores são muito limitados em termos de memória
 - Mais ou menos o que encontrávamos em computadores nos anos 70 e 80.
- No caso do ATMEGA328 do Arduino
 - Memória FLASH - 32 kB
 - Memória EEPROM - 1 kB
 - Memória RAM - 2 kB
- O que são estas memórias, como usa-las?

Memória FLASH

- Utilizada para armazenar o programa
- Não volátil - você pode desligar o microcontrolador que a informação não é perdida
- Leitura extremamente rápida
- Escrita muito complicada
 - Em geral a escrita é feita em grandes blocos (páginas) e precisa-se apagar primeiro a página a ser escrita
 - Inviável durante execução do programa
 - Contudo podemos gravar volumes de dados durante upload do programa para ler durante execução (PROGMEM)

Usando o PROGMEM

```
#include <avr/pgmspace.h>
```

- Forma de armazenar grandes tabelas de dados na memória FLASH

- `const dataType variableName[] PROGMEM = {};`
- `const PROGMEM dataType variableName[] = {};`

- Exemplo

- `const PROGMEM float dados[] = {1.3, 4.2, 5.9, 0.6, 10.1};`

- Para ler a memória use as funções da biblioteca

- `float valor = pgm_read_float_near(dados + 3);`
- Retorna o terceiro valor da tabela

- Outras funções em

- http://www.nongnu.org/avr-libc/user-manual/group_avr_pgmspace.html

Memória EEPROM

- Não volátil - você pode desligar o microcontrolador que a informação não é perdida
- Leitura e escrita fáceis durante execução do programa
- Pode ser entendida como um pequeno hard drive para armazenar informações ao longo da execução do programa
 - EEPROM Library
 - <http://www.arduino.cc/en/Reference/EEPROM>
- Apenas 1kB disponível.

Biblioteca EEPROM

```
#include <EEPROM.h>
```

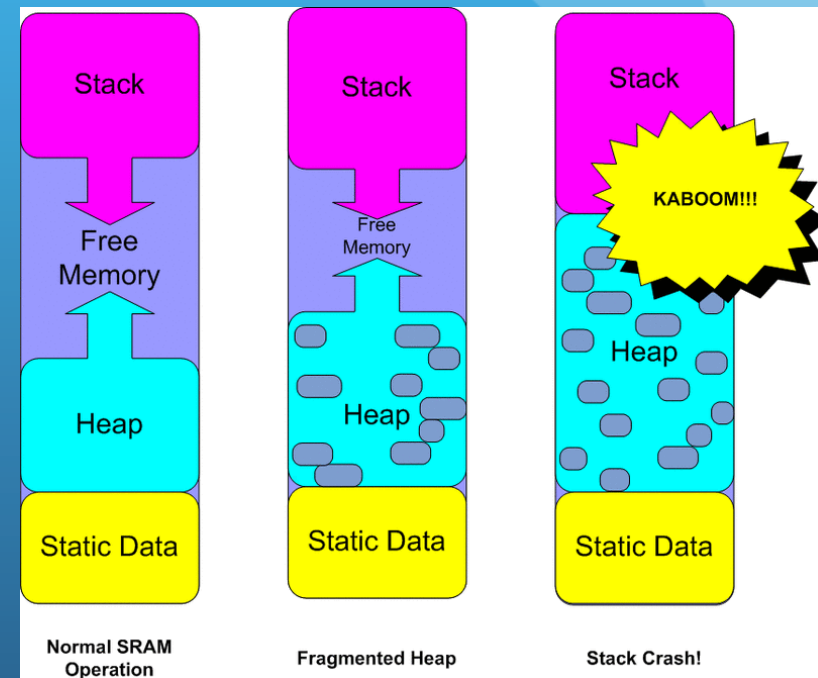
- `read(add)` - lê um byte do endereço `add` na EEPROM
- `write(add, valor)` ou `update(add, valor)` - grava um byte (`valor`) no endereço `add`
- `get(add, objeto)` - restaura para objeto o conteúdo no endereço `add`
- `put(add, objeto)` - grava no endereço `add` o conteúdo de objeto.
- `EEPROM[add]` - Acesso à memória EEPROM como se fosse uma array

Memória RAM

- Memória volátil - RESET ou power OFF apaga conteúdo
- Utilizada para armazenar variáveis, objetos, etc.
- Utilizada como pilha
- Muito pequena
 - 2kB apenas
- Falta de memória é difícil de diagnosticar. As vezes fazemos o upload do programa sem problema mas ele roda de forma estranha.

Como a RAM é organizada

- Variáveis globais e estáticas são carregadas na parte baixa da memória
- A pilha está na parte alta
- Variáveis locais são armazenadas em uma pilha local
 - É totalmente recuperada depois que saímos da função.
- Alocações dinâmicas fazem a heap crescer
 - Mesmo liberando memória, nem sempre a heap diminui, podendo colidir com a pilha



Como otimizar o uso da RAM

- Se possível use `#define` ou invés de variáveis globais para constantes
- Grandes volumes de dados IMUTÁVEIS podem ser gravados na memória FLASH com a diretriz PROGMEM
- Reduza tamanho de buffers em algumas bibliotecas. Não carregue bibliotecas desnecessárias.
- Evite alocações dinâmicas de memória
 - Evite comandos como o `new`.
- Prefira variáveis locais já que a pilha local é destruída depois que a função termina
 - Nem sempre possível.



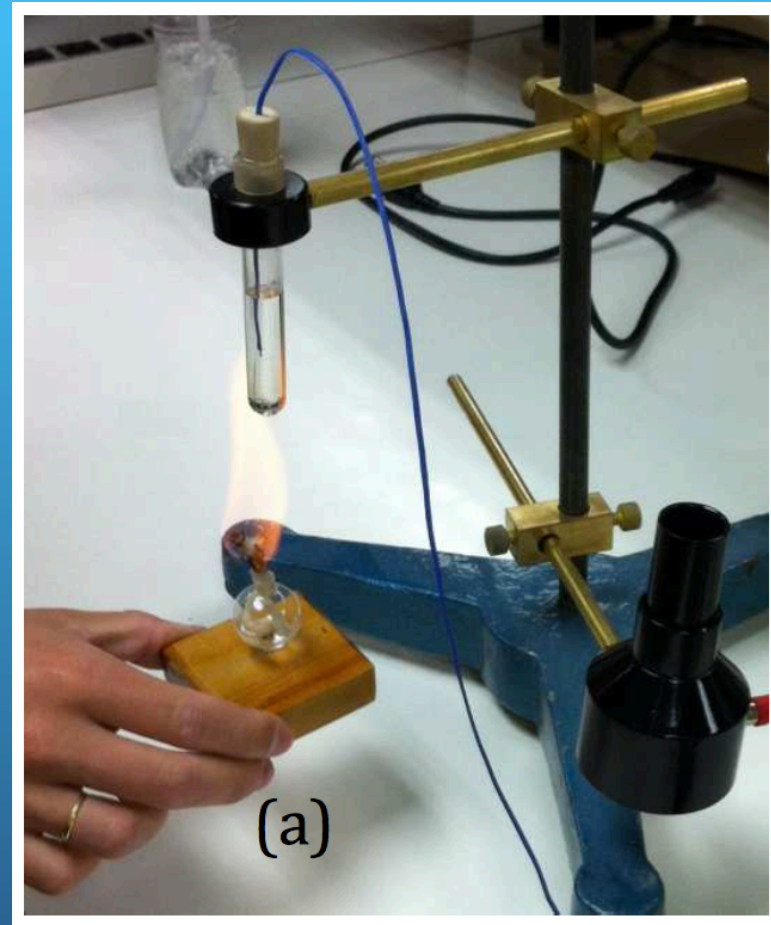
Oficinas

- Usando o ADC do Arduino para medir a temperatura
 - Criando uma biblioteca para um termômetro
- Usando uma tela de LCD de 16x2
 - Biblioteca LiquidCrystal
- Automatizando um experimento em física
- Usando a porta serial para enviar dados para o computador
- Um pequeno interpretador de comandos na porta serial
- Usando a EEPROM para gravar dados

Motivação

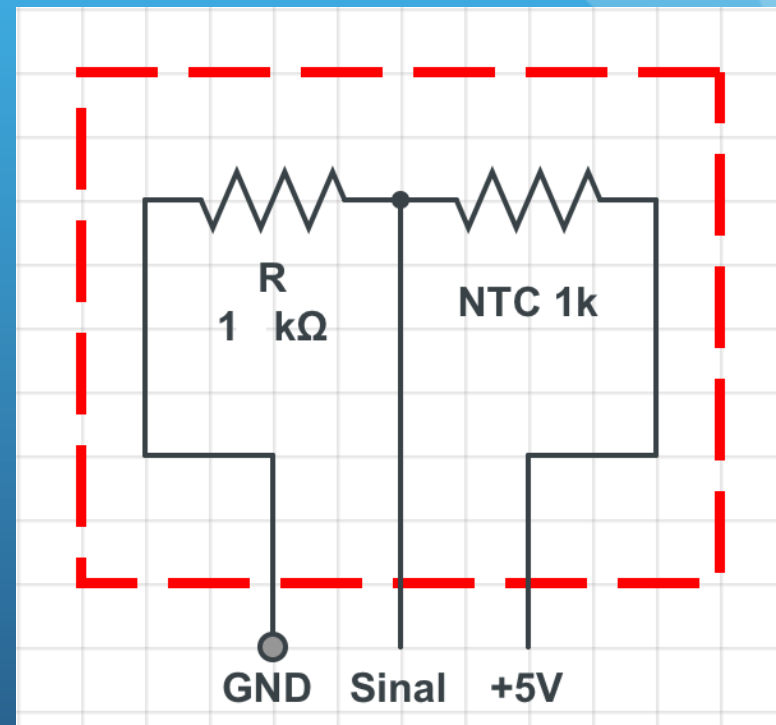
Experimento do resfriamento de um líquido

- Criar um shield para ler a temperatura do líquido.
- Criar uma biblioteca para usar este shield
- Mostrar dados em tempo com um display de LCD
- Gravar os dados em memória
- Gerenciar o processo de um computador



Preparando o shield

- Material:
 - Termistor NTC 1k
 - Resistor sensível à temperatura
 - Resistor conhecido
 - Usar um resistor de ~ 10k
- Montagem
 - Aplicar tensão 5V no conjunto R + NTC
 - Ler voltagem em R
 - Shield tem 3 pinos
 - Use o bradboard para montar o shield



Preparando a biblioteca

- Subdiretório na pasta de bibliotecas do Arduino
 - Documentos → Arduino → libraries
 - Arquivos .h e .cpp
- Construtor
 - Inicializa o objeto
- Métodos públicos ou privados
- Variáveis públicas ou privadas
- Medindo a temperatura
 - Lê o valor de ADC conforme configurado no construtor do objeto
 - Calcula o valor da resistência do NTC
 - Converte para temperatura
 - Retorna o valor da temperatura

Arquivos .h e .cpp

disponíveis para download

sensorTemperatura.h

```
////////////////////////////////////
//
// Sensor de temperatura RTC
//
////////////////////////////////////

#ifndef sensorTemperatura_h
#define sensorTemperatura_h

#include <Arduino.h>
#include <avr/io.h>
#include <avr/interrupt.h>

class sensorTemperatura
{
public:
    sensorTemperatura(int);
    float tempK();
    float tempC();
    void calibra(float, float, float, float);

private:
    int _pino;
    float _REXT;
    float _R0;
    float _B;
    float _T0;
};

#endif
```

sensorTemperatura.cpp

```
#include "sensorTemperatura.h"
#define MAXADC 1024
#define VMAX 5
#define ZEROC 273

sensorTemperatura::sensorTemperatura(int pinoADC)
{
    _pino = pinoADC;

    // para RTC 1K 102
    calibra(1000, 1000, 3636, 298);
}

float sensorTemperatura::tempK()
{
    float adc = analogRead(_pino);
    float V = (VMAX*adc/MAXADC);
    float R = (VMAX-V)*_REXT/V;
    float T = _B/log(R/(_R0*exp(-_B/_T0)));
    return T;
}

float sensorTemperatura::tempC()
{
    return tempK()-ZEROC;
}

void sensorTemperatura::calibra(float REXT, float R0,
float B, float T0)
{
    _REXT = REXT;
    _R0 = R0;
    _B = B;
    _T0 = T0;
}
```

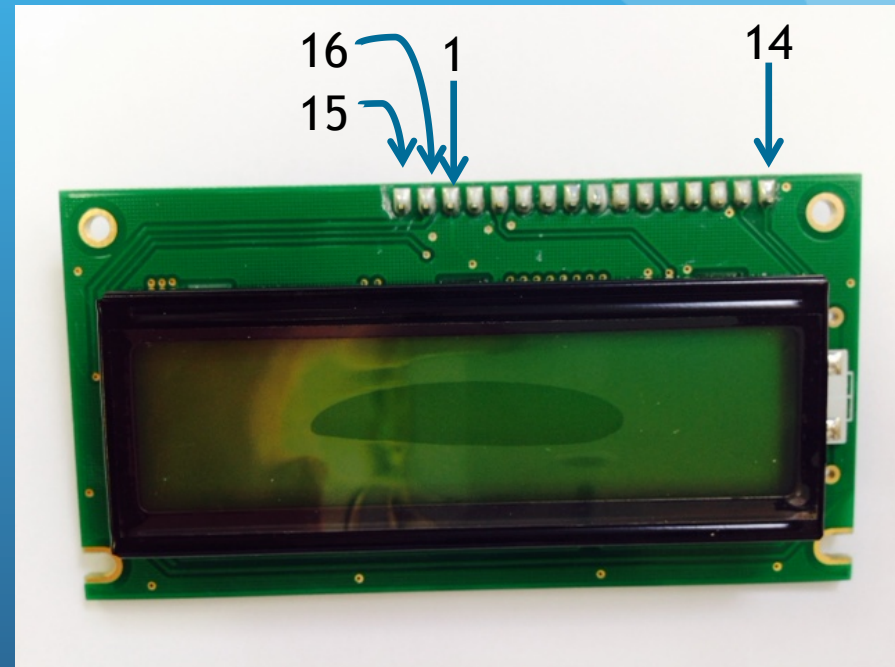
Preparando a aplicação

código disponível para download

- Três partes importantes
 - Incluir a biblioteca `sensorTemperatura`
 - Instanciar um objeto
 - Global, para ser acessível de qualquer parte do código
- Loop()
 - Lê a temperatura
 - Espera um tempo para nova leitura
- Como eu monitoro a temperatura lida no sensor?
- Do que adianta medir se eu não tenho como acessar esta informação do jeito que o programa está?
- 3 métodos
 - Mostrar em um LCD
 - Enviar para um computador
 - Gravar os dados na EEPROM para ler mais adiante

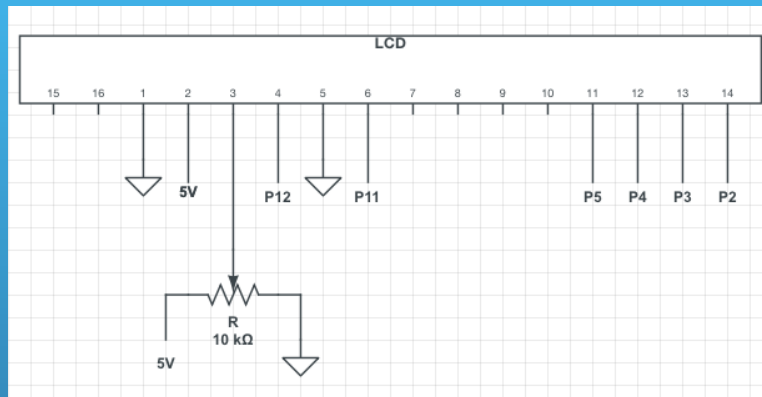
LCD e biblioteca LiquidCrystal

- A biblioteca LiquidCrystal serve para facilitar o uso de telas de cristal líquido compatíveis com Hitachi HD44780
 - Tela comum de 16 x 2 com custo ~ R\$10,00.
 - Simples de utilizar
- CUIDADO!
 - A tela tem 16 pinos de interface, normalmente numeradas de 1 a 16
 - A que compramos, a numeração é um pouco diferente. Fiquem atentos para não ligarem errado.

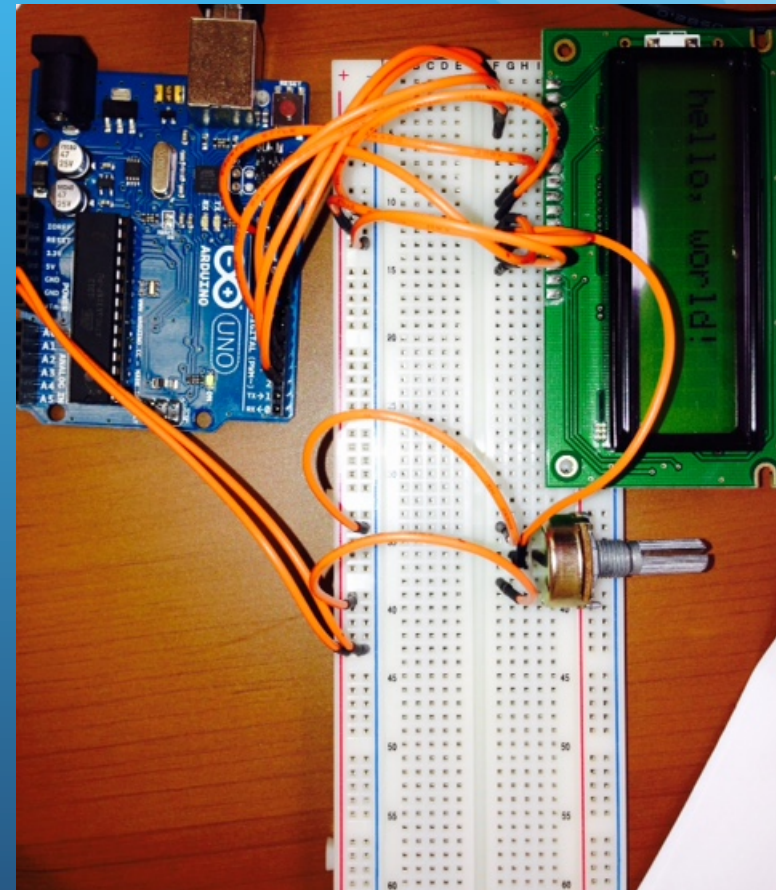


A tela que temos na oficina não tem a pinagem “normal”

Esquema de montagem



- 1 e 5 em terra
- 2 em +5V
- 3 regula o contraste (tensão entre 0 e 5 V)
- 4, 6 e 11-14 - pinos de I/O digitais



Uso do LCD no arduino

<http://www.arduino.cc/en/Reference/LiquidCrystal>

- Biblioteca LiquidCrystal
 - begin() - inicializa e estabelece tamanho da tela
 - clear() - limpa a tela
 - print() - imprime
 - setCursor() - posição de escrita
 - E outras funções. Ver o link acima para mais detalhes
- Código ao lado no site para download

```
// inclui as bibliotecas
#include <LiquidCrystal.h>
#include "sensorTemperatura.h"

// cria os objetos que iremos utilizar no programa
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
sensorTemperatura sensor(0);

void setup()
{
  lcd.begin(16, 2);
  // coloquei a funcao calibra so como exemplo, ja que
  // eh a mesma utilizada no construtor (RTC 1K 102)
  sensor.calibra(1000,1000,3636,298);
}

void loop() {

  float T = sensor.tempK();
  float TC = sensor.tempC();

  lcd.setCursor(0,0);
  lcd.print("T (K) = ");
  lcd.print(T);

  lcd.setCursor(0,1);
  lcd.print("T (C) = ");
  lcd.print(TC);

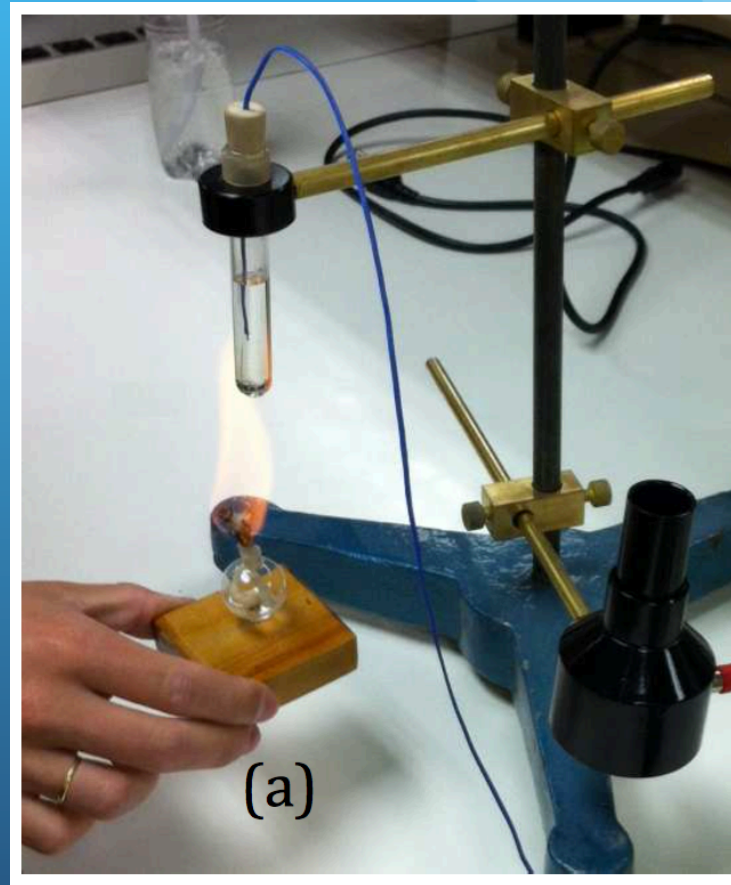
  delay(1000);
}
```


Atividades

- Monte o “shield” para o sensor de temperatura
- Baixe, instale e modifique a biblioteca sensorTemperatura
 - Ex: crie uma função que retorna a temperatura em Fahrenheit
- Baixe, instale e modifique o programa do Arduino
 - Oficina2_temperatura_1.ino
 - Brinque com a tela de LCD, modifique o programa
- Estimativa de tempo: 30-40 minutos

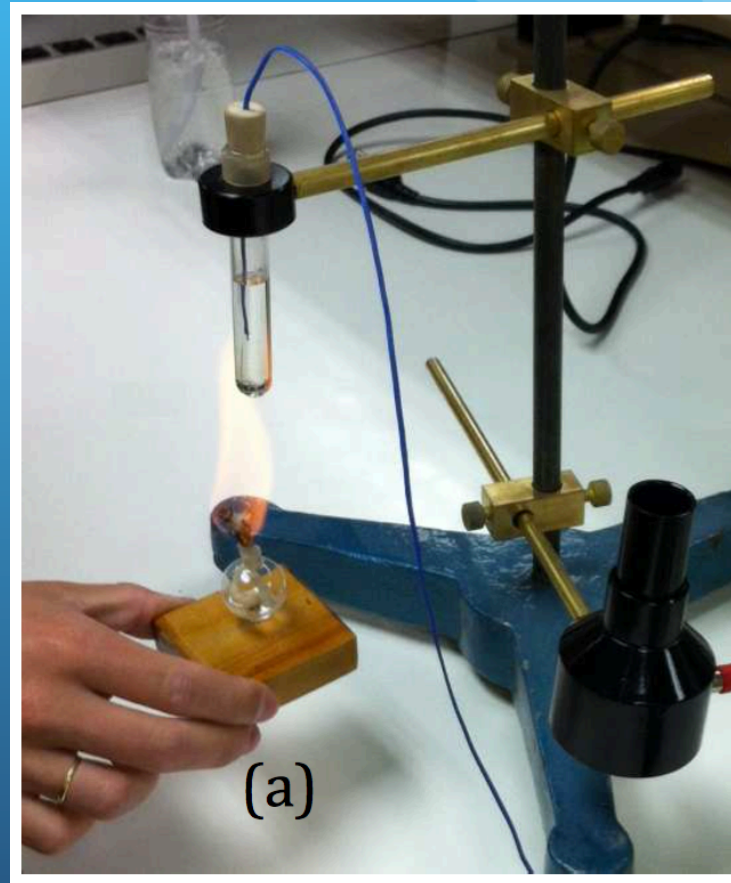
Automatizando o experimento

- Medir temperatura em função do tempo
- No experimento original mede o tempo para cada variação de 0.5°C na temperatura
- Modo mais simples:
 - Botão de reset do Arduino inicia a medida
- Programa para baixar no site



Faça o experimento

- Baixe o programa e faça o upload no Arduino
- Coloque o sensor de temperatura imerso na glicerina
- Aqueça (CUIDADO) até chegar a uns 100°C
 - Use o LCD para monitorar
 - Pressione RESET
 - Vá anotando os valores a cada medida por uns 5 minutos
- Depois vamos discutir: o que poderíamos fazer para melhorar o experimento?



Faça o experimento

- Baixe o programa e faça o upload no Arduino
 - Oficina_2_temperatura_2.ino
- Coloque o sensor de temperatura imerso na glicerina
- Aqueça (CUIDADO) até chegar a uns 100°C
 - Use o LCD para monitorar
 - Pressione RESET
 - Vá anotando os valores a cada medida por uns 5 minutos
- Depois vamos discutir: o que poderíamos fazer para melhorar o experimento?

```
// inclui as bibliotecas
#include <LiquidCrystal.h>
#include "sensorTemperatura.h"

// cria os objetos que iremos utilizar no programa
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
sensorTemperatura sensor(0);

float T0 = 1000;
unsigned long tempo0;

void setup()
{
  lcd.begin(16, 2);
  // coloquei a funcao calibra so como exemplo, ja que
  // eh a mesma utilizada no construtor (RTC 1K 102)
  sensor.calibra(1000,1000,3636,298);
  tempo0 = millis();
}

void loop()
{
  float T = sensor.tempK();
  if(fabs(T-T0)> 0.5)
  {
    unsigned long tempo = millis();
    lcd.setCursor(0,0);
    lcd.print("t (s) = ");
    float dt = (float)(tempo-tempo0)/1000;
    lcd.print(dt);
    lcd.print(" ");
    lcd.setCursor(0,1);
    lcd.print("T (K) = ");
    lcd.print(T);
    lcd.print(" ");
    T0 = T;
  }
}
```

Alguns pontos para melhorar

- Definir melhor o início e fim da tomada de dados
- Transferir os dados automaticamente para o computador
- Armazenar os dados para ver mais tarde

- Como fazer isto com o Arduino?
 - Muitas soluções possíveis
 - Vou usar duas abordagens
 - Uso da porta serial para envio de dados ao computador e controle do experimento
 - Uso da EEPROM para guardar os dados

Usando a porta serial

<http://www.arduino.cc/en/Reference/Serial>

- Inicializar a conexão
 - Em geral na função `setup()`
 - `Serial.begin(velocidade)`
 - $300 \leq \text{velocidade (bps)} \leq 115200$
- Enviar dados arduino \rightarrow outro device (pc)
 - `Serial.print(...)`, `println(...)`, `write(...)`, etc.
 - Formatação da saída é bem limitada
- Receber dados outro device (pc) \rightarrow arduino
 - `Serial.read(...)`, `readString(...)`, `parseFloat(...)`, etc.

Código para porta serial - somente escrita

- Baixe o código e faça o upload
 - Oficina_2_temperatura_3.ino
 - Modifique-o à vontade
- Note que precisamos dar um print para cada parte da mensagem na porta serial.
- Depois do upload, vá em
 - Tools → Serial Monitor
 - Verifique se a velocidade está em 9600
 - Verifique se NEWLINE está selecionado (para marcação de fim de linha)
- Tempo ~ 10 minutos

```
// inclui as bibliotecas
#include <LiquidCrystal.h>
#include "sensorTemperatura.h"

// cria os objetos que iremos utilizar no programa
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
sensorTemperatura sensor(0);

float T0 = 1000;
unsigned long tempo0;

void setup()
{
  Serial.begin(9600);
  lcd.begin(16, 2);
  // coloquei a funcao calibra so como exemplo, ja que
  // eh a mesma utilizada no construtor (RTC 1K 102)
  sensor.calibra(1000,1000,3636,298);
  tempo0 = millis();
}
void loop() |
{
  float T = sensor.tempK();
  if(fabs(T-T0)> 0.5)
  {
    unsigned long tempo = millis();
    lcd.setCursor(0,0);
    lcd.print("t (s) = ");
    float dt = (float)(tempo-tempo0)/1000;
    lcd.print(dt);
    lcd.print(" ");
    lcd.setCursor(0,1);
    lcd.print("T (K) = ");
    lcd.print(T);
    lcd.print(" ");
    T0 = T;
    Serial.print(dt);
    Serial.print(" ");
    Serial.println(T);
  }
}
```

Usando a porta serial para entrada de dados

Interpretador de comandos


- Lógica do programa
 - Observa se tem dados chegando → Arduino na porta serial
 - Se tiver, vai concatenando até chegar um ENTER
 - Interpreta a string e executa o comando apropriado
- Quais comandos
 - start - inicia tomada de dados
 - stop - para tomada de dados
 - print - imprime tabela de dados
- `inp` é uma variável global do tipo `string`
- `toma_dados` é uma variável global booleana que começa com o valor `false`

```
void loop()
{
  if(Serial.available())
  {
    while (Serial.available())
    {
      char inChar = (char)Serial.read();
      inp += inChar;
      if (inChar == '\n') processa();
    }
  }

  if(!toma_dados) return;

  float T = sensor.tempK();

  if(fabs(T-T0)> 0.5)
  {
    unsigned long tempo = millis();
    lcd.setCursor(0,0);
    lcd.print("t (s) = ");
    float dt = (float)(tempo-tempo0)/1000;
    lcd.print(dt);
    lcd.print(" ");
    lcd.setCursor(0,1);
    lcd.print("T (K) = ");
    lcd.print(T);
    lcd.print(" ");
    T0 = T;
    Serial.print(dt);
    Serial.print(" ");
    Serial.println(T);
    saveEEPROM(dt,T);
  }
}
```



Processando eventos

- Define um structure com nomes dos eventos e ponteiros para as funções de cada evento
- Na função processa()
 - Verifica se a string começa com um dos comandos listados
 - Executa a função atribuída.

```
typedef struct cmd
{
    char cmd[10];
    void (*func) (void);
};

#define NCOMANDOS 3

cmd comandos[] = {
    {"start",    cmd_start},
    {"stop",     cmd_stop},
    {"print",    cmd_print}
};

void processa()
{
    inp.toLowerCase();
    inp.trim();
    String I = inp;
    inp = "";
    for(int i = 0; i<NCOMANDOS; i++)
    {
        if(I.startsWith(comandos[i].cmd))
        {
            Serial.println("Executando comando");
            comandos[i].func();
            return;
        }
    }
    Serial.println("0 comando nao foi reconhecido.");
}
```

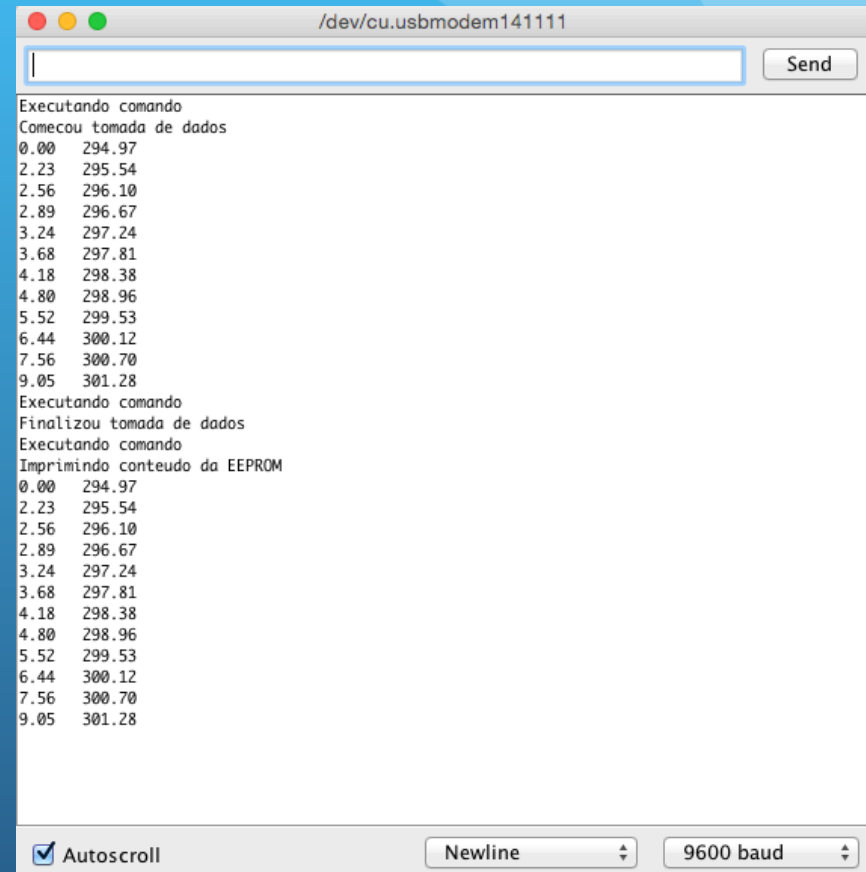
Define as funções

- Escreve o código para cada função atribuída a um comando
- Há uma função para gravar os dados tomados na EEPROM
 - Precisa também incluir no código
 - `#include <EEPROM.h>`
- Código disponível completo para download

```
void cmd_start()
{
  toma_dados = true;
  tempo0 = millis();
  EEPROM.write(0,0);
  Serial.println("Começou tomada de dados");
}
void cmd_stop()
{
  toma_dados = false;
  lcd.clear();
  Serial.println("Finalizou tomada de dados");
}
void cmd_print()
{
  Serial.println("Imprimindo conteudo da EEPROM na ultima tomada de dados");
  byte n = EEPROM.read(0);
  for(int i = 0; i<n; i++)
  {
    int pos = i*(sizeof(float)*2)+1;
    float dt,T;
    EEPROM.get(pos,dt);
    EEPROM.get(pos+sizeof(float),T);
    Serial.print(dt);
    Serial.print(" ");
    Serial.println(T);
  }
}
void saveEEPROM(float dt, float T)
{
  byte n = EEPROM.read(0);
  int pos = (int)n*(sizeof(float)*2)+1;
  if(pos>1020)
  {
    Serial.println("Memoria cheia. Nao posso gravar mais");
    return;
  }
  EEPROM.put(pos,dt);
  EEPROM.put(pos+sizeof(float),T);
  EEPROM.write(0,n+1);
}
```

Atividades

- Baixe o código
 - Oficina_2_temperatura_4.ino
- Execute-o e se familiarize
- Inclua alguns comandos (ex:)
 - zero - limpa a EEPROM
 - media - mostra a temperatura média
- Note que o LCD só fica ligado quando estamos fazendo a tomada de dados
 - Modifique o código para o LCD mostrar continuamente a temperatura e o tempo, independentemente se estamos ou não tomando os dados
- Tempo estimado ~ 40 minutos.



```
/dev/cu.usbmodem141111
| Send
Executando comando
Começou tomada de dados
0.00 294.97
2.23 295.54
2.56 296.10
2.89 296.67
3.24 297.24
3.68 297.81
4.18 298.38
4.80 298.96
5.52 299.53
6.44 300.12
7.56 300.70
9.05 301.28
Executando comando
Finalizou tomada de dados
Executando comando
Imprimindo conteudo da EEPROM
0.00 294.97
2.23 295.54
2.56 296.10
2.89 296.67
3.24 297.24
3.68 297.81
4.18 298.38
4.80 298.96
5.52 299.53
6.44 300.12
7.56 300.70
9.05 301.28
 Autocroll
Newline
9600 baud
```

Outras atividades

- Usem o tempo restante da oficina para explorar outras ideias.
- Estou à disposição para discuti-las.