UNIDADE 1: PRIMEIROS PROGRAMAS

OBJETIVOS

Criar alguns programas iniciais que agilizem o processo de trabalho com sistemas de entrada (*input*) e de saída (*output*) (I/O) digitais.

Vamos experimentar programas criados em Arduino para que perceba como se estruturam, para, de seguida desenvolver um você mesmo. Arduino é uma linguagem de programação sofisticada, com regras, instruções e sintaxe próprias. Vamos começar por abordar as instruções base, onde será necessário fazer uso de *input* e *output* digitais:

SECÇÃO TEÓRICA

- EM QUE CONSISTE UM PROGRAMA
  - Secção de comentários
  - Secção de variáveis e funções
  - Secção de configuração de tarefas
  - Secção do corpo central do programa
  - Secção de mensagens
- INSTRUÇÕES DO PROGRAMA
  - Função set up()
  - Função loop()
  - Função pinModel()
  - Função digitalRead()
  - Função digitalWrite()
- OPERADORES LÓGICOS
  - Operador NOT
  - Operador AND
  - Operador OR
  - Combinar operadores

SECÇÃO DE PRÁTICA

- EXEMPLO 1: Acender LEDs de 1 volt

- EXEMPLO 2: Acender LEDs de 2 volts
- EXEMPLO 3: Acender LEDs de 3 volts
- EXEMPLO 4: Iluminação reversa de LEDs
- EXEMPLO 5: Controlo de dois comandos
- EXEMPLO 6: Trabalho individual

MATERIAIS PRÁTICOS

- Computador portátil ou de secretária

- Ambiente de trabalho Arduino IDE; deve incluir o material suplementar já instalado e configurado

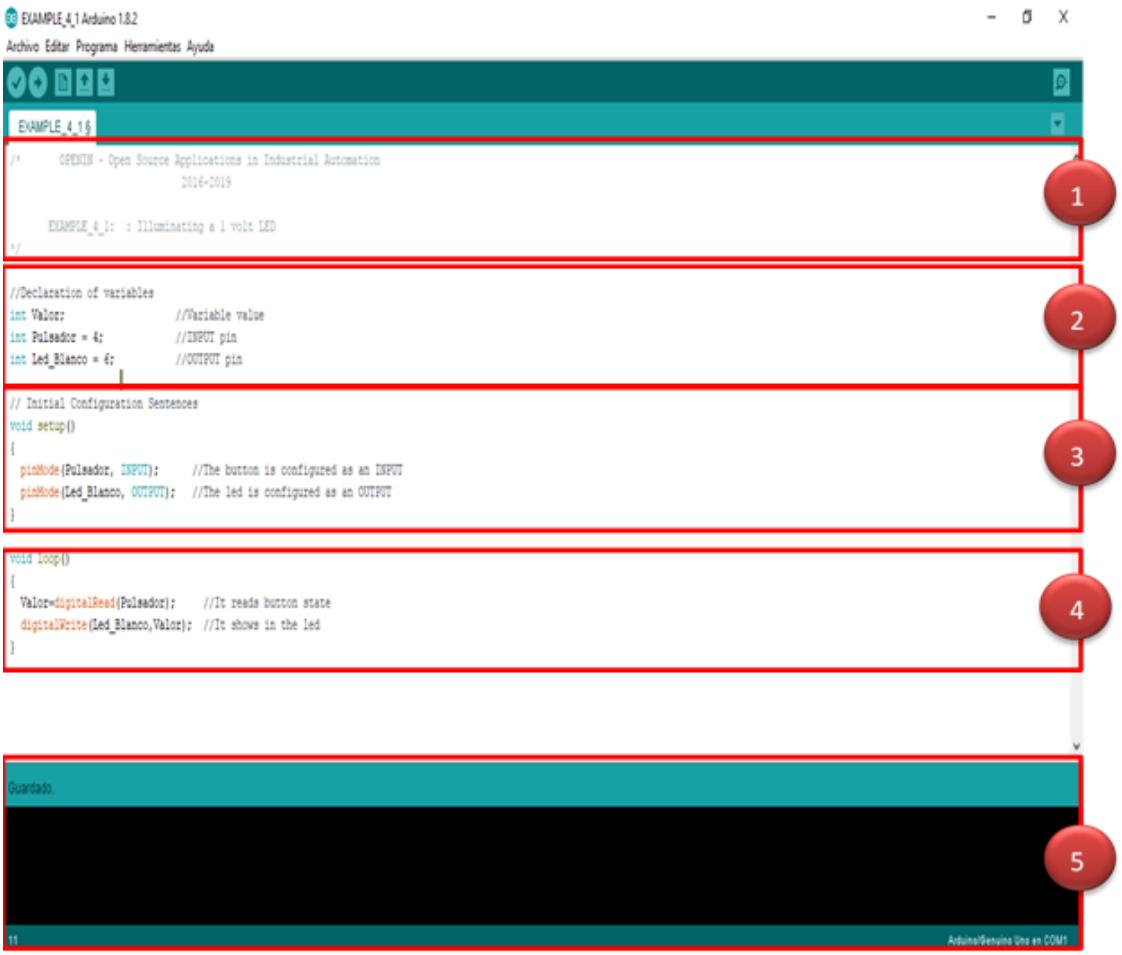
- Placa de controlo Arduino UNO

- Cabo USB

ÍNDICE

SECÇÃO TEÓRICA

1. EM QUE CONSISTE O PROGRAMA



A. SECÇÃO DE COMENTÁRIOS

Todos os programas devem começar por fornecer um conjunto de informações. Neste caso, por exemplo, será o título do curso, a data, o autor, a empresa, etc.(Figura 1). Além disso é interessante que inclua uma explicação sobre o programa e o que permite fazer.

Esta informação denomina-se “Header Comments” (“Comentários de Cabeçalho”). Podem ser incluídos todos os comentários que considerar pertinentes, onde quiser, desde que surjam entre os símbolos “/*” e “*/”. Observe atentamente o exemplo na Figura 1.

Também podem ser incluidos comentários simples de uma linha. Neste caso, antes de inserir os comentários utilize apenas o símbolo "//"; a utilização destes comentários é bastante comum.

Figura 1

Seja proativo! Habitue-se a escrever comentários sobre o seu programa. Pode precisar deles posteriormente para se recordar do que fez, como fez ou porque o fez.

B. SECÇÃO SOBRE A DECLARAÇÃO DE VARIÁVEIS E FUNÇÕES

Recomendamos que utilize esta secção para declarar as variáveis e funções. Ainda não sabe o que são, mas não se preocupe, vai aprender em que consistem um pouco mais à frente. Imagine uma variável como uma espécie de caixa ou recetáculo a que atribui um nome e, possivelmente, também um valor. Guarde-a na memória RAM e poderá utilizá-la mais tarde, o número de vezes que quiser.

Uma função, por outro lado, abrange outras instruções, regras e funções. Dê também um nome a este grupo e poderá utilizá-lo sempre que necessitar. A Figura 2 exemplifica a criação de uma declaração de variáveis.

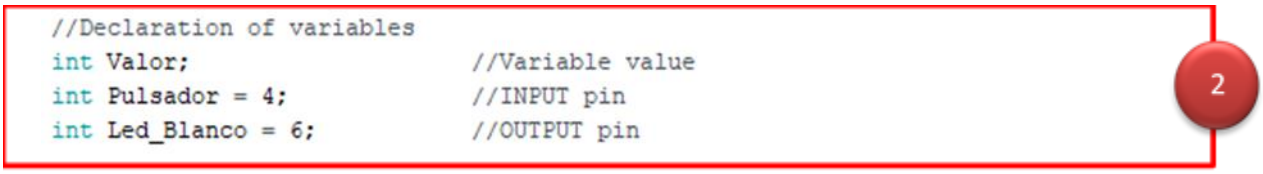


Figura 2

Foram criadas três variáveis no exemplo representado na figura acima: "Valor", "Pulsador" e "Led_Blanco". O valor do *input* digital será guardado no primeiro. À segunda variável "Pulsador", é atribuído o valor 4 que corresponde ao número do *pin* de *input* a que o interruptor está associado; temos que ler o estado do interruptor. Foi-nos fornecido o valor 6 para a variável "Led_Blanco", correspondendo este valor ao pin de *output* a que o LED fora associado. O LED branco é o que consta da placa de experiências BASIC I/O.

De uma forma geral, quer as variáveis quer as funções têm de ser declaradas ANTES de serem utilizadas no programa.

C. SECÇÃO DE CONFIGURAÇÃO DE TAREFAS

Normalmente, os programas escritos em Arduino começam por executar algumas instruções e funções. Estas determinam que *pins* devem ser usados como *input* e quais devem ser utilizados como *output*. Observe a Figura 3 abaixo.

```
// Initial Configuration Sentences
void setup()
{
  pinMode(Pulsador, INPUT);      //The button is configured as an INPUT
  pinMode(Led_Blanco, OUTPUT);   //The led is configured as an OUTPUT
}
```

3

Figura 3

O pin da variável "Pulsador" representado na Figura 2, a que foi previamente atribuído o valor 4, configura-se como *INPUT* (Figura 3). O pin da variável "Led_Blanco", ou pin 6, configura-se como *OUTPUT.*

De uma forma geral, as instruções de configuração executam-se apenas quando faz RESET do sistema ou quando o liga a uma fonte de energia.

D. SECÇÃO DO CORPO CENTRAL DO PROGRAMA

Todas as regras, instruções e funções que constituem o seu programa têm de ser escritas nesta secção. No exemplo que estamos a utilizar de momento existem apenas duas funções, mas não se esqueça que um programa pode conter centenas ou mesmo milhares de funções. Observe o exemplo na Figura 4.

```
void loop()
{
  Valor=digitalRead(Pulsador);     //It reads button state
  digitalWrite(Led_Blanco,Valor);  //It shows in the led
}
```

4

Figura 4

A primeira função, "Valor=digitalRead(Pulsador);", lê o estado digital do interruptor ligado ao pin 4, o que anteriormente havia sido configurado como *input*. O estado do interruptor é guardado na variável "Valor" na memória RAM.

A segunda função, "digitalWrite(Led_Blanco,Valor);", escreve o conteúdo da variável "Valor" no pin 6; este aspeto foi lido na função anterior. Lembre-se que o pin número 6 havia sido previamente configurado como *output*.

O controlador executa o mais rapidamente possível todas as funções que formam o corpo central do programa. Executa-as constantemente e indefinidamente desde a primeira à última.

Importante: ao declarar as variáveis, configuração e principais funções do programa, certifique-se que terminam SEMPRE com o símbolo: “;”.

E. SECÇÃO DE MENSAGENS

Nesta secção final, o ambiente de trabalho do Arduino IDE apresenta uma série de mensagens, como a apresentada na Figura 5.

Nesta secção, é avisado está a guardar, a compilar ou a gravar um programa na memória do controlador e, também, se há erros de compilação, o tipo de erro e onde ocorreu.

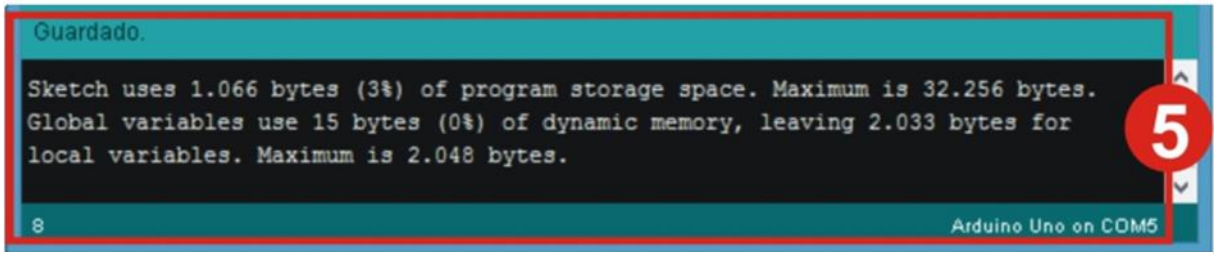


Figura 5

Este exercício foi um sucesso. Sabemos que o programa ocupa um total de 1,066 dos 32,256 bytes de memória FLASH disponíveis. 15 dos 2,048 bytes de memória RAM disponíveis foram utilizados para dados (devido às variáveis que criámos). Sendo assim, sobram 2,033 bytes.

Lembre-se: “Compilar” um programa significa traduzir para código binário (também conhecido como linguagem máquina) o que escreveu na linguagem complexa Arduino; isto é o que fica realmente gravado na memória FLASH do controlador. Relativamente ao que exige de si, como programador, este é um processo simples e automático.

2. INSTRUÇÕES DO PROGRAMA

Agora já percebe em que consiste um programa – pelo menos em teoria. Sabe que é constituído por um número de instruções, também conhecidas como “FUNÇÕES”; um programa pode conter milhares de funções.

Cada linguagem de programação tem as suas funções específicas, a sua própria sintaxe e regras de programação. Vamos focar-nos na linguagem de programação Arduino. Já utilizámos cinco instruções ou funções diferentes, agora vamos analisá-las com mais detalhe.

A. A FUNÇÃO SETUP()

Esta função e todas as que contém são executadas quando o sistema é reiniciado. Esta situação ocorre sempre que pressiona o botão REINICIAR (RESET) ou sempre que liga o sistema.

Normalmente, existem outras funções integradas nesta: configuração do *pin* de *input* e *output*, configurações de certas variáveis, bibliotecas, etc.. De uma forma geral, pode incluir todas as instruções e funções que quiser na função setup(), desde que se encontrem entre chavetas: "{...}". Tem sempre de criar uma função setup(), mesmo que não contenha nada: deixe simplesmente as chavetas vazias.

O aspeto essencial a reter é que todas as funções da função setup() são executadas em simultâneo: quando reinicia o sistema.

Sintaxe:

Void setup()

{

.....}

Exemplo:

Void setup()

{

pinMode(INPUT, Button); *//The button pin is configured as an input*

pinMode(White_LED,OUTPUT); *//The button pin is configured as an output*

}

B. A FUNÇÃO LOOP()

Vai inserir todas as suas instruções e funções nesta função, entre chavetas "{...}". Aqui constrói o "corpo principal" do programa.

Sintaxe:

Void loop()

{

.....}

Exemplo:

Void loop()

{

Value=digitalRead(Button); *//Lê o estado do botão.*

digitalWrite(White_LED,VALUE); *//Surge no LED*

}

A principal função loop() de um programa pode incluir centenas ou milhares de outras funções. O exemplo acima inclui apenas duas. A função loop() também pode estar vazia, sem nenhuma função entre chavetas { }. De qualquer forma, este aspeto não altera nada: tem sempre de haver uma função loop().

Todas as outras funções entre as chavetas da função loop() são executadas constantemente, da primeira à última, até o sistema ser desligado.

C. A FUNÇÃO PINMODE()

Configure um dos pins controladores do Arduino como *input* ou *output*. Normalmente, aparece no início do programa e incluiu-se na função setup().

Sintaxe:

pinMode;

pin: Este é o número do pin que vamos configurar; pode ser entre 0 e 13 no Arduino UNO.

Mode: Estabelece se o pin atua como *INPUT* ou *OUTPUT*

Exemplos:

int Button=4;

int White_LED=6;

pinMode(White_LED, OUTPUT); *//O pin 6 é output*

pinMode(9 OUTPUT); *//O pin 9 é output*

pinMode(Button, INPUT); *// O pin 4 é input*

pinMode(12, INPUT); *// O pin 12 é input*

Todos os pins digitais são automaticamente configurados como *inputs* quando faz reset do sistema.

D. A FUNÇÃO DIGITALREAD()

Esta função lê e retorna o estado lógico binário (“1” ou “0”, “HIGH” ou “LOW”) de qualquer pin controlador do Arduino.

Sintaxe:

digitalRead(pin);

pin: Apresenta o número do pin que vamos ler; pode ser entre 0 e 13 no Arduino UNO.

Exemplos:

int Button=4;

Value=DigitalRead(Button); *//Lê o estado do pin 4 e guarda-o em "Value""*

X=digitalRead(8); *//Lê o estado do pin 8 e guarda-o em "X"*

E. A FUNÇÃO DIGITALWRITE()

Escreve e estipula o valor binário ("1" ou "0", "HIGH" ou "LOW") através de um pin de *output*.

Sintaxe:

digitalWrite(pin, value);

pin: Dá-nos o número do pin a partir do qual vamos estipular o valor; pode estipular-se entre 0 e 13 no Arduino UNO.

Value: Indica o valor a ser estipulado ("1" ou "0", "HIGH" ou "LOW").

Exemplos:

int White_LED=6;

DigitalWrite(White_LED,Value); *//Estipula o conteúdo de "Value" ("1" ou "0" através do pin 6)*

DigitalWrite(11, LOW); *//Estipula o nível "0" através do pin de output 11*

DigitalWrite(7,1); *//Estipula o nível "1" através do pin de output 7*

3. OPERADORES LÓGICOS

Não podemos avançar sem abordar três tipos de operadores que são capazes de relacionar diferentes tipos de funções, operações, expressões, etc.. Estes operadores são conhecidos como operadores lógicos, e são muito utilizados em sistemas digitais. Vamos explicá-los se uma forma simples, sem recorrer a demasiados termos técnicos.

A. O OPERDOR NOT

Este operador exprime negação (NOT). Representa-se por um ponto de exclamação (!). É mais fácil de compreender com um exemplo. Repare na função que se segue, vai de certo reconhecê-la:

Value=digitalRead(Button);

A variável “Value” equivale ao nível “1” quando o “Button” também é “1”. Se não for este o caso, “Value” equivale ao nível “0”. Agora, atente na seguinte função:

Value = ! digitalRead(Button);

A variável “Value” equivale ao nível “1” quando o “Button” não se encontra em **NOT** no nível “1”, ou, por outras palavras, não está no nível “0”. Por outro lado, se o botão estiver no nível “1”, a variável “Value” não será “0”. Atente no exemplo abaixo e responda:

Value = ! digitalRead(12);

“*Value* equivale ao nível “1” ou “*true*” se: _____

“*Value*” equivale ao nível “0” ou “*false*” se: _____

B. O OPERADOR AND

Esta operação gera um nível “1”, também conhecido como “*true*”, quando TODOS os elementos que interrelaciona também pertencem a um nível “1” ou são “true” (verdadeiros). Representa-se com os símbolos: “&&”. Observe o exemplo:

Value =digitalRead(4) && digitalRead(12);

A variável “Value” equivale ao nível “1”, “*true*”, quando os *inputs* 4 e (AND) 12 são lidos e ambos se encontram também no nível “1”. Se algum dos *inputs* estiver no nível “0”, o resultado para a variável “Value” também será nível “0”, ou “*false*”. Observe o exemplo abaixo e responda:

Value =digitalRead(4) && digitalRead(8) && digitalRead(12);

“*Value*” equivale a nível “1” ou “*true*” se: _____

“*Value*” equivale a nível “0” ou “*false*” se: _____

C. O OPERADOR OR

Esta operação gera um nível “1”, também conhecido como “true”, quando QUALQUER UM dos elementos que inter-relaciona também se encontra no nível “1” ou é verdadeiro (“true”). Representa-se com os símbolos “||”. Observe o exemplo:

Value =digitalRead(4) || digitalRead(12);

A variável “Value” equivale ao nível “1”, “*true*”, quando ambos os *input* 4 OU (OR) 12 OU (OR) se encontram no nível “1”. Se todos os *inputs* estiverem no nível “0” o resultado da variável “Value” também será nível “0” ou “*false*. Veja o exemplo que se segue e responda:

Value =digitalRead(4) || digitalRead(8) || digitalRead(12);

"*Value*" equivale a nível "1" ou "*true*" se: _____

"*Value*" equivale a nível "0" ou "*false*" se: _____

D. COMBINAR OPERADORES

É possível combinarem-se vários operadores lógicos na mesma função. Tem de usar parênteses (…) para estabelecer a ordem em que devem ser acedidos e calculados. Observe no exemplo seguinte:

Value = ! digitalRead(4) && **digitalRead(7);**

A variável "Value" equivale ao nível "1" ou "*true*" quando o *input* 4 não está (is **NOT)** no nível "1", mas **(BUT)** o *input* 7 se encontra no nível 1. Observe o exemplo e responda:

Value = (digitalRead(8) && **! digitalRead(12)) || digitalRead(4));**

"*Value*" equivale a nível "1" ou "*true*" se: _____

"*Value*" equivale a nível "0" ou "*false*" se: _____

SECÇÃO DE PRÁTICA

4. EXEMPLO 1: Acender um LED de 1 volt

Este é o mesmo exemplo que estudou na Secção Teórica, por isso, não vamos repetir a explicação que incluída anteriormente. Queríamos apenas recordar que este exemplo mostra como se lê o estado do botão ligado ao pin 4 e como é registado no LED branco, no quadro de experiências BASIC I/O ligado ao pin 6 do controlador.

- **Compilar e Gravar**

Como pode observar na Figura 6, tem apenas que pressionar o botão marcado com 1. O ambiente Arduino IDE compila o seu programa traduzindo-o para código máquina ou binário; todo o processo é relativamente rápido e completamente automático. Não é muito relevante saber os detalhes exatos deste processo.

Se o programa estiver bem escrito, sem erros, vai ver mensagens como as incluídas na janela 2 (Figura 6). Já as viu anteriormente: informam-no sobre quanto espaço o programa irá ocupar no controlador da memória FLASH e sobre quanta memória RAM irá ser consumida pelas variáveis que utiliza. Informa ainda, caso o programa tenha sido bem escrito, que este ficará imediatamente gravado na memória FLASH do controlador.

Se houver algum erro, essa informação aparece na mesma janela: os tipos de erros que fez e onde se encontram no programa. Corrija-os e continue a compilar o programa.

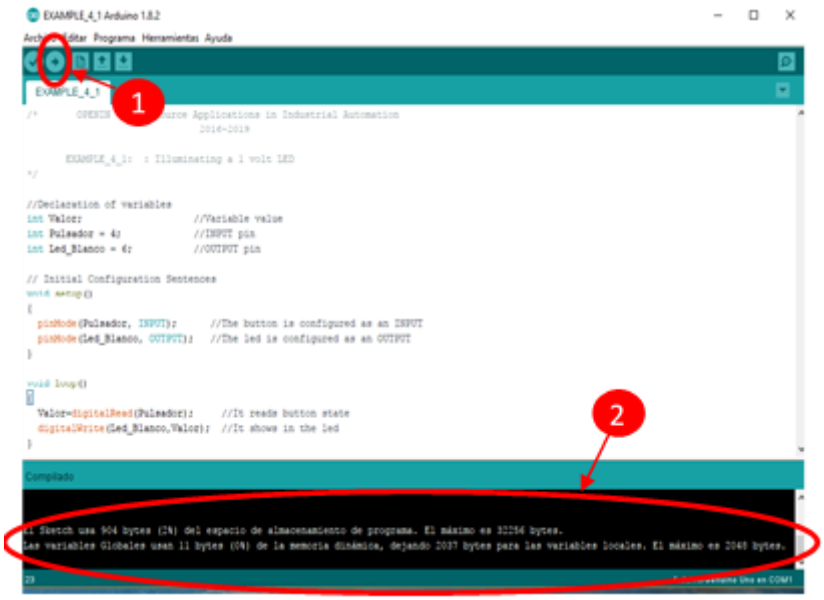


Figura 6

- **Checking**

Mesmo que o ambiente do Arduino IDE não o informe sobre nenhum erro quando termina a compilação, não significa necessariamente que o programa vá funcionar corretamente. O ambiente deteta apenas erros sintáticos, isto é, uma função que não tenha sido escrita corretamente ou que não exista, variáveis que não tenham sido declaradas de forma adequada, dados que não existam, etc.

O verdadeiro teste acontece quando se verifica se o controlador faz exatamente aquilo para que o programou. Este exemplo é muito simples: partindo do princípio que o quadro de experiências BASIC I/O se encontra no painel de controlo do Arduino UNO de forma correta, tudo o que precisa de fazer é pressionar o botão D4: se o LED branco associado ao pin D6 acender, está tudo bem; se não acender, o programa desliga-se.

Uma última verificação: quando grava um programa numa memória FLASH, o programa fica aí indefinidamente, mesmo que desligue o sistema. Aqui fica outro detalhe: assim que um programa é gravado, o painel Arduino já não depende, em nenhum momento, do PC. Uma forma muito simples de o testar é retirar o cabo USB do painel Arduino UNO e ligá-lo a uma fonte de alimentação comum, ou a uma bateria. Vai rapidamente perceber se está tudo a funcionar normalmente.

Talvez neste momento lhe pareça que este aspeto não é uma mais-valia e que não precisa de um controlador para acender um LED com um botão. Engana-se. Este é o primeiro passo: se compreendeu o conceito de um programa e as suas funções, estará apto para levar a cabo tarefas muito mais complexas. Tem apenas de avançar passo a passo.

5. EXEMPLO 2: Acender um LED de 2 volts

Aqui está outra versão do programa anterior. Controla o painel BASIC I/O do LED branco ligado ao pin 6, com o botão ligado ao pin 4 no mesmo painel. A única diferença é não terem sido utilizadas variáveis para definir que pins estariam ligados ao botão e ao LED branco.

Os números de referência dos pin 4 e 6 aparecem nas próprias funções: pinMode(), digitalRead(), digitalWrite(). Neste caso, os números dos pins apresentam-se como “*constantes*”, não como “*variáveis*”.

Tal como fez com o exemplo anterior, compile e grave o programa. De seguida, certifique-se que funciona corretamente.

- **Conclusões**

Utilizando as variáveis ou constantes para definir um pin é basicamente uma decisão sua. Por exemplo, se o botão vai estar permanentemente associado ao pin 4, utilize este número como uma constante, tal como fizemos no exemplo – será guardado na memória RAM.

Por outro lado, se houver alguma possibilidade de este botão, ou qualquer outro periférico, sofrer alterações nas ligações do controlador de pin, é melhor definir esta ligação como uma variável: por exemplo, “intButton=4;”. Desta forma, se futuramente o botão for ligado ao pin 12, a única alteração que terá de fazer será a declaração da variável: “intButton=12;”. Não tem de modificar as funções do programa para utilizar este botão.

6. EXEMPLO 3: Acender um LED de 3 volts

Aqui está ainda outra versão final do programa para controlar LED com um botão.

A primeira ação a executar é compilar o programa e gravá-lo no controlador; certifique-se que funciona adequadamente.

- **Conclusões**

Se estudar o programa atentamente, vai verificar as diferenças entre este exemplo, o EXEMPLO 1 e o EXEMPLO 2. Podemos concluir o seguinte:

- ✓ Não foi utilizada nenhuma variável; isto significa poupança na memória RAM. Em muitos casos, utilizar variáveis e constantes depende do que realmente estiver interessado em fazer.
- ✓ Os pins a serem utilizados como *input* digitais não precisam de ser configurados com o pinMode(). Todos os pins são configurados automaticamente como *input*.
- ✓ O resultado de uma função pode ser utilizado como *input* para outra função sem passar pela variável intermédia "Value", como nos exemplos anteriores. Observe a expressão seguinte:

digitalWrite(6,digitalRead(4));

Há uma função integrada noutra. A função mais interna é que é executada em primeiro lugar: digitalRead(4). O resultado avança para a função seguinte digitalWrite(...);.

- ✓ Um programa pode ser escrito de diversas formas. Neste momento, preocupe-se apenas com o facto de o programa funcionar corretamente; este é o aspeto mais importante. Com a prática, será capaz de reduzir a quantidade de memória FLASH do programa, assim como os dados de memória RAM utilizados, o que tornará o programa mais "EFICIENTE".

Pode fazer uma verificação muito simples. Compile e grave o EXEMPLO 1, EXEMPLO 2 e o EXEMPLO 3. Verifique quanta memória FLASH do programa e que variável de memória RAM utiliza. Obtém esta informação no ambiente do Arduino IDE sempre que concluir a gravação de um programa – aparece em baixo. Complete a tabela seguinte:

EXEMPLO	Memória FLASH do Programa	Dados da Memória RAM
EXEMPLO 1		
EXEMPLO 2		
EXEMPLO 3		

7. EXEMPLO 4: Iluminação reversível de um led

A ideia será utilizar o operador lógico NOT e criar um programa que acenda o LED branco ligado ao pin 6, sempre que o botão ligado ao pin 4 NÃO (NOT) estiver ligado.

A parte principal do programa está inserida na função loop(), realçada a amarelo. É muito semelhante ao exemplo anterior. A única diferença é que a função NO, representada pelo símbolo "I", inverte a leitura do botão ligado ao pin 4.

O exemplo apresentado na Figura 7 pode simular uma máquina com um motor que está sempre a trabalhar a não ser que seja acionado um botão de emergência.

Figura 7

8. EXEMPLO 5: Controlador de dois botões

Imagine que pretende controlar um alarme com dois botões bastante separados. O alarme deve desligar-se quando seleciona OR ou outro (ou ambos). O alarme é simulado pelo LED vermelho no painel experimental BASIC I/O ligado ao pin 11. Os dois botões estão ligados aos pins 4 e 12.

Aqui está o programa final (Figura 8). Observe-o com atenção.

Figura 8

9. EXEMPLO 6: Trabalho individual

Utilize o EXEMPLO 6 como um programa modelo ou “template”. Faça as alterações necessárias ao modelo para resolver os exercícios propostos. Esperamos que utilize todo o conhecimento que adquiriu com os conteúdos apresentados até ao momento e que aprenda ainda mais. Acima de tudo, divirta-se! À medida que vai aprendendo mais sobre programação, vai aperceber-se que o controlador é capaz de muito mais do que acender um LED simples... está apenas a começar!

Exercício	Descrição
1	O LED âmbar deve acender-se ao pressionar os botões 7 e 12.
2	O LED verde deve acender-se ao pressionar os botões 12 e 8, mas não o botão 4.
3	O LED branco acende-se quando se pressiona o botão 4, não o 7. O LED vermelho acende-se quando se pressionam os botões 8 ou 12.
4	O LED verde acende-se quando se pressiona o botão 4, mas não o botão 7, e O quando se pressiona o botão 8, mas não o botão 12.
5	O LED vermelho acende-se quando os botões 4, 7 e 8 são pressionados ou, quando é pressionado o botão 12.

Recorde o seguinte: Tem de registar cada uma das modificações que faz no ambiente do Arduino IDE, de as gravar no controlador e assegurar-se que funciona de acordo com as especificações de cada exercício.

Complete a tabela que se segue com as instruções que criou para o seu programa.

Exercício	Função	Instruções do seu programa
1	setup()	
	loop()	
2	setup()	
	loop()	
3	setup()	
	loop()	
4	setup()	
	loop()	
5	setup()	
	loop()	

REFERÊNCIAS

LIVROS

[1]. **EXPLORING ARDUINO**, Jeremy Blum, Ed.Willey
[2]. Practical ARDUINO, Jonathan Oxer & Hugh Blemings, Ed.Technology in action
[3]. Programming Arduino, Next Steps, Simon Monk
[4]. Sensores y actuadores, Aplicaciones con ARDUINO, Leonel G.Corona Ramirez, Griselda S. Abarca Jiménez, Jesús Mares Carreño, Ed.Patria
[5]. ARDUINO: Curso práctico de formación, Oscar Torrente Artero, Ed.RC libros
[6]. 30 ARDUINO PROJECTS FOR THE EVIL GENIUS, Simon Monk, Ed. TAB
[7]. Beginning C for ARDUINO, Jack Purdum, Ph.D., Ed.Technology in action
[8]. ARDUINO programming notebook, Brian W.Evans

SÍTIOS WEB

[1]. https://www.arduino.cc/
[2]. https://www.prometec.net/
[3]. http://blog.bricogeek.com/
[4]. https://aprendiendoarduino.wordpress.com/
[5]. https://www.sparkfun.com