DEV 2BE

Apostila 2020

LINGUAGEM C E ARDUINO

APLICADOS AO DESENVOLVIMENTO DE GAMES

Alexandre Zanlorenzi | Bruno Segato
Thainá Weingartner | Vinícius Jacik

UniAmérica
Centro Universitário

SUMÁRIO

PROGRAMAS NECESSÁRIOS

Tinkercad:

- Plataforma de simulação para circuitos e Arduino: https://www.tinkercad.com/

CodeBlocks com compilador MinGw:

- IDE para criação de algoritmos, disponível em: http://www.codeblocks.org/downloads/26
- Obs: selecionar a versão com compilador MinGW

Ou outro IDE de sua escolha:

- Visual Studio: https://visualstudio.microsoft.com/pt-br/downloads/
- Ao fazer o download certifique-se de que a IDE tenha um compilador compatível com a linguagem C.

Indicações Bibliográficas:

- C Completo e Total, Herbert Schildt;
- Linguagem C, Luís Damas;

Indicações de Canais do Youtube:

- Brincando com ideais;
- WR kits;
- Boson Treinamentos;
- CFBCursos;

DICIONÁRIO DO PROGRAMADOR

Como em todas as áreas, Engenharia de Softwares também tem seus termos para evitar ambiguidades quando se quer dizer algo. Durante essa apostila, toda vez que algo novo for apresentado, teremos uma sessão *"Dicionário do Programador"* para ajudar-te a gravar e conhecer os nomes e termos utilizados. Você também pode pesquisar sobre esses nomes para obter mais informações.

De início, temos alguns nomes que precisa saber:

- **Algoritmo:** é uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema;
- **Sintaxe:** é como se escreve algo;
- **Semântica:** é o significado daquilo que você escreveu;
- **Biblioteca (library):** é um conjunto de funções predefinidas pelo sistema que serão carregadas pelo compilador;
- **Função:** é um conjunto de sub-rotinas, comandos e outras funções que realizam uma tarefa no seu algoritmo;
- **Variáveis:** identifica um valor guardado na memória e que pode ser alterado.

INTRODUÇÃO A PROGRAMAÇÃO

O que é um programa?

De maneira geral um programa pode ser definido como um conjunto de instruções, escritas de forma estruturada que buscam resolver um problema. A partir de agora iremos abordar estes aspectos de modo a entender o que e como são definidas estas instruções e como elas são estruturadas.

Começaremos com o mais simples dos programas: Diga “Oi” para o mundo.

De acordo com a IDE que estiver utilizando, crie um arquivo .C para criar seu primeiro algoritmo. Feito isso, vamos aprender como organizar as coisas para montar nosso programa.

Agora que você já criou seu arquivo vamos entender dois conceitos muito simples que precisaremos utilizar sempre que desejarmos escrever nossos programas em C, para que estes funcionem de maneira correta.

Primeiro de tudo, vamos adicionar a biblioteca que vamos usar, a biblioteca *standard input and output*, para adicioná-la devemos digitar o seguinte comando **#include** e o nome de seu arquivo **stdio.h**.

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

Agora você deve estar se perguntando o que este comando faz? Bem como dito no parágrafo anterior ele adiciona uma biblioteca ao nosso programa, mas para que precisamos disto?

De maneira simples, uma biblioteca adiciona ao nosso programa, instruções que já foram escritas por outros programadores, fazemos isto para tornar a nossa vida mais simples, uma vez que não precisamos nos preocupar com instruções complexas para as quais já temos uma solução pronta.

Se tomarmos como exemplo a nossa biblioteca **stdio.h**, ela nos permite utilizar instruções de acesso às entradas e saídas de dados, ou seja ela nos permite utilizar nosso terminal para escrevermos na tela, e também nos permite ler através do programa o que digitamos. Ao decorrer deste curso falaremos mais sobre bibliotecas e o poder que elas têm.

Depois de incluirmos nossa biblioteca precisamos nos preocupar com algo muito importante na linguagem C, que é a sua função principal.

Quando tentamos compilar um programa em C, o compilador vai lendo linha por linha do nosso programa até encontrar esta função, pois ela é responsável por contar ao compilador onde o nosso programa de fato inicia sua execução.

Em C, essa sempre será a função **main**. As funções têm seu tipo e seus parâmetros de acordo com o que precisam para funcionar e o que vão realizar, aprofundaremos isso mais tarde. Por hora, você pode escrever:

```
void main () { }
```

Agora que já inserimos no nosso programa alguns comandos básicos para que ele entenda o que precisa fazer, vamos de fato dizer olá para o mundo da programação.

Para fazer isto vamos utilizar a função **printf** para imprimir no monitor as informações que queremos. Vamos escrever a seguinte linha de código dentro das chaves da nossa função principal.

```
printf("Ola, Mundo");
```

O resultado deve ser algo parecido com a imagem a seguir:

```c
#include <stdio.h>
int main(){
    printf("Ola, Mundo!");
    return 0;
}
```

Presta atenção, cabeção!

As linhas de comando em C são finalizadas por um ponto e vírgula "**;**" para que o compilador saiba onde acaba a linha;

A linguagem C é ***case sensitive***, ou seja, ela difere letras maiúsculas e minúsculas nas palavras, então tome cuidado ao escrever.

Antes que eu me esqueça..... O que ia falar mesmo?

A lembrei, algo muito importante para conhecermos neste primeiro momento são comentários, eles são muito importante para documentar nossos programas, podendo ser usados também para que quando outra pessoa abra o nosso código consiga entender o que determinado trecho que que esteja comentado.

E neste início da nossa jornada através da programação, se você for esquecido como um dos autores desta apostila, os comentários podem ser úteis para lhe ajudar a lembrar de forma rápida o que determinado bloco de código faz.

Então vamos verificar como fazer um comentário no nosso código?

```c
/*
    Isto é um comentário
    de várias linhas.
*/
#include <stdio.h>
int main(){
    printf("Ola, Mundo!"); // Isto é um comentário de uma linha
    return 0;
}
//Será que você sabe qual dos autores é o mais esquecido?
```

Agora que já escrevemos o nosso primeiro programa, estão prontos para os próximos passos, ou preferem uma xícara de café para refrescar as idéias?

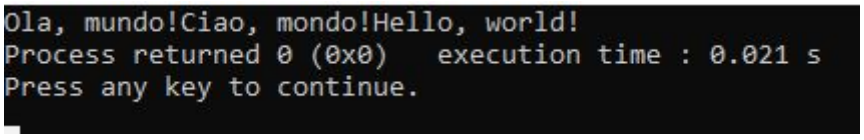
Já pegou seu café? Sim?

Então vamos lá!

Bora programar?!

Abra sua IDE, crie um novo documento, insira a biblioteca básica utilizada em C e crie um programa que imprima "Olá, mundo!" em pelo menos três idiomas diferentes, dentro do programa comente ao lado qual o idioma daquela saudação.

O resultado deve ser algo parecido com a imagem abaixo.



OBS: Tente não utilizar o ctrl+C, ctrl+V ao longo dos exercícios, é muito importante digitar todos os comandos para fixar o conteúdo estudado.

VARIÁVEIS

O que são variáveis?

Variáveis nada mais são do que recipientes, assim como um pote de nescau que serve para guardar o chocolate em pó. Dessa forma, assim como temos diferentes tipos de recipientes para diferentes produtos, também temos diferentes tipos de variáveis para diferentes tipos de dados. E da mesma forma que podemos trocar o nescau por café podemos também trocar os valores de uma variável, contudo devemos sempre nos atentar ao tipo de dado que cada variável pode armazenar.

Cada variável precisa de um **nome** para identificá-la e de um **tipo**. Os tipos de dados mais utilizados em C são:

int - valores inteiros;
- exemplos `-2, 10, 20000;`
- para declarar `int nome_da_variavel;`
- para ler e imprimir `"%d"`

float - valores com virgula, chamados de ponto flutuante;
- exemplos `-6.5, 9.64, 5.0;`
- para declarar `float nome_da_variavel;`
- para ler e imprimir `"%f"` ou `"%.nf"` sendo "`n`" a quantidade de números que desejar utilizar após a vírgula

bool - valor booleano, 1 ou 0;
- exemplos `1, 0` ou `true, false;`
- para declarar `bool nome_da_variavel;`
- para ler e imprimir `"%d` ou `%s"`

Pesquise sobre "variáveis em C" e "vetores em C" para encontrar o valor máximo que cada variável pode armazenar e outros tipos de variáveis.

char - caractere;
- exemplos `"a", "?", "*", "0";`
- para declarar `char nome_da_variavel;`
- para ler e imprimir `"%c"`

string - vetor de caracteres;
- exemplos `"palavra", "uma frase", "eu adoro cafe";`
- para declarar `char nome_da_variavel[n];` sendo "`n`" a quantidade de caracteres da string.
- para ler e imprimir `"%s"`, para uma palavra somente ou `"%[^\n]"` quando desejamos toda a linha até pressionar enter;

Para atribuir um valor ou expressão a uma variável, usa-se o sinal de **=**, como a seguir:

```
variável = expressão ou valor;
idade = (ano atual – ano de nascimento);
```

Presta atenção, cabeção!

Para nomear as variáveis existem algumas regras: deve-se iniciar com letra, não podem conter espaços ou caracteres especiais, não pode ser uma das palavras que são reservadas para o uso do sistema.

Agora que já vimos como escrever no console usando a função a **printf** e também aprendemos o que são **variáveis** que tal fazermos um uso prático delas? Vamos pedir para que o nosso programa apresente os cumprimentos aos nossos participantes do Dev2Be de 2020?

O primeiro passo então é perguntarmos qual o nome do nosso participante certo? Para fazer isso basta utilizarmos a função **printf**, a estrutura do nosso programa neste ponto deve ser muito semelhante à que usamos para dizer Olá ao mundo da programação, confira na figura abaixo.

```c
#include <stdio.h>
int main(){
    printf("Caro participante, qual eh o seu nome?");
    return 0;
}
```

Neste ponto perguntamos ao participante como é seu nome, mas como vamos guardar este nome? Bem para isso vamos utilizar as variáveis, mas especificamente um vetor de **char**, ou seja uma **string,** que como aprendemos nesta seção é o tipo ideal para armazenarmos um nome.

Vamos então declarar esta variável, para que possamos utilizá-la ao longo do nosso programa? Para fazermos isso basta que declaremos a nossa variável de acordo com o exemplo que vimos nesta seção, o resultado deve ser algo como o código abaixo.

```c
#include <stdio.h>
int main(){
    char nome[10]; // Variável do tipo String
    printf("Caro participante, qual eh o seu nome?");
    return 0;
}
```

Agora só nos falta ler através do programa o nome digitado pelo usuário, e para isso isso nós iremos utilizar a função **scanf** que lê o que foi digitado no console.

O resultado deve estar parecido com o código abaixo, este código deve perguntar ao participante qual o seu nome, e armazenar este nome dentro de uma variável.

Depois utilizamos a variável que contém o valor do nome e pedimos que o nosso programa de boas vindas para o usuário.

```c
#include <stdio.h>
int main(){
    char nome[10]; // Variável do tipo String
    printf("Caro participante, qual eh o seu nome?\n");
    scanf("%s", nome);// Função Scanf
    printf("Bem vindo ao Dev2Be, %s!", nome);
    return 0;
}
```

Algo importante para ressaltar neste ponto é, como você pode perceber até aqui nós utilizamos alguns caracteres como **`\n`** e **`%s`**, e você deve estar se perguntando o porque disto, certo?

A linguagem C possui em alguns caracteres especiais como:

- **`\n`** - Quebra de linha;
- **`\t`** - Tabulação vertical;
- **`\a`** - Sinal sonoro do computador;
- **`\0`** - Caracter nulo

E os caracteres de formatação, que servem para formatar a saída ou entrada dos dados dependo do seu tipo como por exemplo:

- **`%c`** - Caracter
- **`%s`** - String;
- **`%d`** - Inteiro;
- **`%f`** - Float;

E além destes temos muitos outros, mas não se preocupe com isto agora, ao decorrer do curso vamos aprender melhor sobre eles e como usá-los;

Presta antenção, cabeção!

Quando usamos a função **scanf** devemos nos atentar a um pequeno detalhe, se formos ler um dado digitado pelo usuário, para inserir este valor na variável devemos utilizar o caracter especial **&**, confira no código abaixo, como fazemos isto.

```c
#include <stdio.h>
int main(){
    int numero; // Variável do tipo int
    printf("Digite um numero inteiro:\n");
    scanf("%d", &numero);// Caractere "&"
    printf("O numero digitado eh %d!", numero);
    return 0;
}
```

E assim concluímos a segunda etapa da nossa apostila. E aí já acabou o café? Quer um tempinho para pegar outra xícara?

Bora programar?!

1. Crie um programa que peça o nome, e-mail e uma senha de 6 dígitos e imprima o seguinte texto:

nome, seja bem vindo ao Clube de Programadores Dev2Be. Seu cadastro foi efetuado com sucesso com o e-mail email@email.com e sua senha é XXXXXX.

OPERADORES

Lembra que no capítulo passado utilizamos o sinal de **=** para **atribuir um valor** a uma variável? Então... o **=** é um **operador de atribuição**, e na linguagem C, temos vários outros operadores que são utilizados para realizar operações matemáticas ou verificar condições. Por isso, muita atenção à utilização de cada um deles:

Operador	Ação
Aritiméticos	
+	somar
–	subtrair
*	multiplicar
/	dividir
%	resto da divisão (Inteiros)
Lógicos	
!	nega uma operação
&&	operador AND
||	operador OR
Relacionais	
>	maior que
<	menor que
=>	maior ou igual que
=<	menor ou igual que
==	verificar uma igualdade
Operadores de incremento e decremento	
++	incremento unitário
––	decremento unitário

ATENÇÃO
- O operador de divisão fornece resultado inteiro apenas quando ambos os operandos são inteiros. Ex:
7 / 2 == 3;
7.0 / 2 == 3.5;

- O operador de resto somente pode ser utilizado com operandos inteiros. Ex:
7 % 2 == 1;
7.0 % 2 == erro;

Ao desenvolver um programa é muito comum termos que utilizar expressões como:

```
variável = variável operador expressão;
ciclo = ciclo + 5;
```

Para simplificar esse tipo de expressão é possível colocar o operador da expressão precedendo o operador de atribuição e retirar a variável repetida, da seguinte forma:

```
variável operador= expressão;
```

```
ciclo += 5;
```

Presta atenção, cabeção!

Assim como na matemática, na programação os operadores possuem ordem de precedência, isso significa que em uma expressão a execução de suas operações não ocorrerá da esquerda pra direita simplesmente. Por isso, é utilizado parênteses para priorizar as operações que o programa deve executar primeiro. Veja a tabela abaixo para entender a ordem de precedência dos operadores.

Precedência dos operadores em C					
Maior	( )	[ ]			
	++	--			
	*	/	%		
	+	-			
	<	<=	>	>=	
	==	!=			
	!	&&	||		
Menor	=	+=	-=	*=	/=

Bora programar?!

1. Ler um número inteiro do teclado e imprimir seu sucessor e seu antecessor.
2. Fazer um programa em C que pergunta um valor em metros e imprime o correspondente em decímetros, centímetros e milímetros
3. Faça um algoritmo que leia três notas de um aluno, calcule e escreva a média final deste aluno. Considerar que a média é ponderada e que o peso das notas é 2, 3 e 5.

ESTRUTURAS CONDICIONAIS

Em nosso dia-a-dia, quase tudo que fazemos são tomadas de decisões baseadas em condições. Por exemplo, ao sair de casa de manhã para o trabalho é comum avaliarmos o clima, se o tempo estiver fechado com probabilidade de chuva, provavelmente pegamos um guarda-chuva e um casaco, se o tempo estiver claro e calor, vestimos uma roupa mais fresca e não esquecemos o óculos de sol. Da mesma forma, um programa é desenvolvido com diversas interações e tomadas de decisão que são criadas através de estruturas condicionais.

Para verificar essas condições usamos as estruturas **if** e **else**, que significam em inglês **se** e **senão**. A estrutura da função é montada como a seguir:

```
if(condição) {
      Bloco de instruções;
} else {
      Bloco de instruções;
}
```

A função **if** verifica se a condição é verdadeira, caso for, executa seu bloco de instruções, caso não seja, o **else** é executado. **O uso do else é opcional**;

É comum termos uma série de condições que devem ser levadas em consideração na tomada de uma decisão. Por exemplo, se temos um programa que verifica a idade de uma pessoa e imprime se essa pessoa é uma criança, adulto ou idoso. Precisamos verificar a idade em 3 condições diferentes, para isso, ao invés de utilizarmos apenas o if seguido de else, usaremos o **else if** seguido da nova condição, veja a seguir:

```
if(idade >= 0 && idade <= 17) {
      printf("voce eh uma crianca");
} else if (idade > 17 && idade <= 59){
      printf("voce eh um adulto");
} else if (idade > 59){
      printf("voce eh um idoso");
} else {
      printf("Opcao invalida!");
}
```

Além das situações em que são utilizadas mais de uma condição na tomada de decisão, também é muito comum termos que realizar um novo teste dentro de uma condição. Para exemplificar isso, vamos supor um programa que deseja separar pessoas por gênero e idade. Para isso, utilizamos um primeiro if para verificar a condição de gênero e dentro dessa estrutura outros ifs verificam a idade. Veja o código a seguir para compreender melhor:

```
if (genero == "mulher") {
    if (idade <= 17) {
        printf("voce eh uma mulher, crianca");
    } else if (idade > 17 && idade <= 59){
        printf("voce eh uma mulher, adulta");
    } else if (idade > 59){
        printf("voce eh uma mulher, idosa");
    }
} else if (genero == "homem"){
    if (idade <= 17) {
        printf("voce eh um homem, crianca");
    } else if (idade > 17 && idade <= 59){
        printf("voce eh um homem, adulto");
    } else if (idade > 59){
        printf("voce eh um homem, idoso");
    }
 }
```

Em C, existe um comando para tomada de **decisão múltipla**, o **switch**, muito utilizado em menus. O intuito desse comando é escolher uma entre várias alternativas definidas, a partir da leitura de uma expressão. Vamos conferir sua estrutura.

```
switch (opção){
case opção:
      instrução;
      Break;
default:
      break;
}
```

Para utilizarmos o **switch** basta que seja definido uma variável como por exemplo um número inteiro, e então a utilizamos como controle, caso seja 1 execute determinada sequência de código, caso 2, execute outra ação e assim por diante, e caso não seja nenhuma entrada esperada podemos ter também um caso **default**.

Outro ponto importante que devemos ressaltar, é o uso do comando **break**, este comando serve para dizer ao programa que pare a execução de determinado bloco de código. Como no **switch** acima, se a expressão selecionada for o caso 1, pedimos para executar a ação que queremos e usamos o **break**, para sair do **switch** depois de realizar esta ação. Sem o uso do **break**, o programa irá seguir para a próxima opção.

Bora programar?!

1. Crie um algoritmo que peça o ano de nascimento do usuário e informe se ele pode fazer a carteira de motorista;

2. Ler dois valores e imprimir uma das três mensagens a seguir: ‘Números iguais’, caso os números sejam iguais ‘Primeiro é maior’, caso o primeiro seja maior que o segundo e ‘Segundo maior’, caso o segundo seja maior que o primeiro.

3. Crie uma calculadora, que realize as quatro operações fundamentais, para 2 valores, dica: você pode usar um char para identificar o sinal da conta

4. Utilizando o comando switch, desenvolva um programa que pergunte ao usuário qual a sua bebida preferida (café, chá ou chocolate), e retorne uma resposta dizendo. “`Aqui está sua xícara de (%s)`”.

LAÇOS DE REPETIÇÃO

Em programação um dos comandos fundamentais são os laços de repetição, que permitem que um conjunto de instruções seja executado até que ocorra um certa condição. Vamos supor que você deseja criar um programa que imprima os números de 0 a 100 na tela, para isso seria possível criar uma variável inicializando-a como 0 e escrever 200 linhas de código no seguinte formato:

```
int numero = 0;

printf("%d\n", numero); // numero == 0
numero++;
printf("%d\n", numero); // numero == 1
numero++;
printf("%d\n", numero); // numero == 2
numero++;
printf("%d\n", numero); // numero == 3
numero++;
(...)
```

Agora imagina se um programa simples como esse tomasse 200 linhas no minimo, quantas linhas seriam necessárias para desenvolver softwares mais complexos. Loucura né? Por isso utilizamos o comando **for**. Veja a seguir o mesmo programa, só que dessa vez otimizado pelo uso de laço de repetição.

```
int numero;

for (numero = 0; numero <= 100; numero++) {
   printf("%d\n", numero);
}
```

Muito mais simples né? Digita os dois códigos na sua IDE e veja como o resultado é o mesmo. Agora vamos entender como funciona essa estrutura do **for**.

```
for (inicialização; condição ; incremento/decremento)
    {
        comando;
    }
```

Entretanto, o **for** não é o único comando de repetição, também existe o laço **while** que possui o formato da seguinte maneira:

```
while (condição) {
        comando;
    }
```

Diferentemente do for, o while não exige uma condição inicial nem um incremento ou decremento. O while executa o comando enquanto a condição for verdadeira.

O seguinte exemplo mostra um programa que imprime uma mensagem enquanto a tecla 1 não for pressionada:

```c
#include <stdio.h>

int main () {
int sair = 0;
    while (sair != 1) {
        printf("Seja bem-vindo!\n");
        printf("Pressione 1 para sair\n");
        fflush(stdin); //comando para limpar o buffer do teclado
        scanf("%d", &sair);
        system("cls"); //comando para limpar tela
    }
    printf("Saindo do programa.\n");
    return 0;
}
```

Você deve ter percebido que os dois laços de repetição que apresentamos (**for** e **while**) testam a condição no começo, certo? Como alternativa para isso, existe o **do-while** que verifica a condição ao final do laço. Veja como é utilizado o **do-while**:

```c
do {
    comando;
} while(condição);
```

A diferença do **while** para o **do-while** é que ele irá executar o comando ao menos uma vez.

No capítulo anterior nós aprendemos o uso do comando **break**, que também é utilizado dentro de laços de repetição. Outro comando utilizado em C é o **continue**, que trabalha de forma parecida com a do **break**, mas em vez de forçar a saída do laço, ele força que ocorra a próxima iteração, pulando o que estiver no comando.

Pesquise sobre os comandos *exit* e *goto*, quais suas funcionalidades e onde eles podem ser utilizados.

Bora programar?!

1. Crie um programa que imprima toda a tabuada de 1 ao 9.
2. Desenvolva um programa que escreva de 80 a 30 decrescendo de 5 em 5.

3. Faça um programa em que o usuário digite 2 valores e se a soma deles forma maior que 15 o programa encerra, caso contrário, repete.

4. Desenvolva um programa que leia vários conjuntos de três valores reais e mostre para cada conjunto: sua soma, seu produto e sua média. O programa para quando um conjunto não entrar com seus valores em ordem crescente.

5. Escreva um programa em que o usuário digita um número e o programa retorna se o número digitado é um número primo ou não é um número primo.

6. Solicitar a idade de várias pessoas e imprimir: Total de pessoas com menos de 21 anos. Total de pessoas com idade entre 21 e 50 anos. Total de pessoas com mais de 50 anos. O programa termina quando idade for = 150.

INDENTAÇÃO

Se liga! Um dos pré requisitos para programar em grupo com sucesso é um código bem indentado. É importante mantermos um código organizado para que todos compreendam de maneira clara o que está sendo escrito. Um código bem indentado facilita a entender o código, corrigir os erros e corrigir a lógica do programa.

Indentação é o espaço entre o começo da linha e o início do bloco de código, esse espaço pode ser feito utilizando a tecla tab ou uma sequência de espaços. Cada vez que se inicia um bloco de instrução deverá aumentar o espaço entre o texto e à margem esquerda. Ao fechar esse bloco, retoma-se a distância utilizada no procedimento anterior.

Abaixo vemos um código bem indentado:

```
int main (){
    int i, j;
    for (i = 0; i <= 10; i++){
        if(i==4 || i==9){
            for (j = 0; j <= 10; j++){
                printf("%d x %d = %d\n", i, j, i * j);
            }
        }else{
            printf("Numero nao %d valido\n", i);
        }
    }
    return 0;
}
```

Perceba que o alinhamento das variáveis e o primeiro laço for estão a um tab da margem esquerda, as estruturas if/else que estão encadeadas dentro do for estão a dois tabs, e por sua vez os trechos de código dentro dessas a três tabs e assim por diante.

Código sem indentação:

```
int main (){
int i, j;
for (i = 0; i <= 10; i++){
if(i==4 || i==9){
for (j = 0; j <= 10; j++){
printf("%d x %d = %d\n", i, j, i * j);}}
else{
printf("Numero nao %d valido\n", i);}}
```

```
return 0;}
```

Código mal indentado:

```
#include <stdio.h>

int main (){

          int i, j;
    for (i = 0; i <= 10; i++){
if(i==4 || i==9){
                    for (j = 0; j <= 10; j++){
                printf("%d x %d = %d\n", i, j, i * j);
            }
         }
   else{
        printf("Numero nao %d valido\n", i);
        }
     }
     return 0;
}
```

Manter um código bem identado é algo bastante simples, e atualmente as IDEs nos ajudam a realizar esta tarefa, sendo que a maioria delas já possuem funções para auto indentar nossos códigos, no entanto, principalmente neste ínicio da sua jornada, o ideal é nos acostumarmos a já escrevê-los de forma correta.

E aí, conseguiu entender a diferença entre um código bem identado, e um código mal identado? Sim? Muito simples esse tópico não, nem deu tempo de terminar o café certo?

Então vamos aos próximos passos!

VETORES

Um vetor é um conjunto de variáveis de um mesmo tipo, que compartilham o mesmo nome. Para entendermos melhor isso pense na seguinte situação. Imagine que você queira armazenar todos os meses do ano, com o que aprendemos até agora nós já sabemos como fazer isso certo?

Sabemos que para isso utilizarímos variáveis, se quisermos armazenar em forma de números fariamos algo como:

```c
#include <stdio.h>
int main (){
    int jan = 1;
    int fev = 2;
    int mar = 3;
    int abr = 4;
    //E assim por diante
    return 0;
}
```

Ou para armazenarmos as primeiras três letras de cada mês:

```c
#include <stdio.h>

int main (){
    char mes_1[4] = "Jan";
    char mes_2[4] = "Fev";
    char mes_3[4] = "Mar";
    char mes_4[4] = "Abr";
    //E assim por diante
    return 0;
}
```

Mas ao fazer isto, note que precisamos declarar cada mês em sua respectiva variável, e você deve estar se perguntando, e se eu tiver 1000 valores para armazenar, como por exemplo o quadro de funcionários de uma empresa? Seria algo extremamente trabalhoso não?

Para isso podemos então aplicar vetores, e como fazemos isso? Bem um vetor como já dito é nada mais do que uma lista de valores que compartilham o mesmo nome, sendo assim para declarar um vetor é muito semelhante a declarar uma variável simples, ele possui a seguinte estrutura:

```c
int meses_do_ano[12];
```

Onde:

- int – É o tipo de variável que o vetor vai armazenar;
- meses_do_ano - É o nome do nosso vetor; e

- [12] - É a quantidade de valores que o nosso vetor pode armazenar, ou seja quantas posições ele tem;

Agora temos um vetor com 12 posições, ideal para o nosso caso em que desejamos armazenar os doze meses do ano.

Mas no planeta onde eu moro tem 25 meses e agora? Caso o planeta em que você habita tenha mais ou menos meses você pode ajustar esse valor. tudo bem? Mas só por curiosidade, diz aí qual é o seu planeta?

Seguindo aqui na terra, para atribuirmos ou acessarmos uma das posições de um vetor é necessário que façamos uma referência a essa posição no nosso código, vejamos como fazer isso.

```c
#include <stdio.h>

int main (){
    int meses_do_ano[12];// Declaração do vetor
    meses_do_ano[0] = 1; // Atribuindo o mês 1 na primeira posição
    meses_do_ano[1] = 2; // Atribuindo o mês 2 na segunda posição

    //Acessando a posição do vetor

    printf("%d", meses_do_ano[0]);
    return 0;
}
```

Presta atenção, Cabeção!

Os vetores começam com o índice **0** ou seja a primeira posição do vetor possui o índice **0**, e o último índice possui o valor de **n-1**, onde **n** é o número de posições que o vetor possui. Assim o último índice de um vetor de 12 posições é:

$$Índice = 12 - 1 = 11$$

Observando o código acima podemos ver como atribuir um valor a determinada posição do vetor, basta indicar qual o índice desta posição e usar o operador de atribuição. E para acessar este valor chamamos a variável (vetor) e passamos qual posição queremos acessar.

Mas da forma como atribuímos, fomos colocando posição por posição e isso não é o ideal para o nosso caso certo? Para isto temos a **carga inicial automática de vetores**. Ela funciona da seguinte maneira, logo quando declaramos um vetor nós podemos inicializar ele já com os valores que queremos em cada posição isso se faz da seguinte forma:

```
int meses_do_ano[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```

Basta que declaremos o vetor e usando o operador de atribuição **=** e entre **{}** (chaves, esse não é o amigo do Kiko) inserimos os valores de cada posição em ordem e separados por **‘,’**.

Além disso podemos automatizar a inserção ou leitura de valores dentro de um vetor através de estruturas de repetição como o **for** e **while**. Vamos fazer isto?

```c
#include <stdio.h>

int main (){
    int meses_do_ano[12];// Declaração do vetor
    int i;
    // Atribuindo os valores as posições do vetor
    for(i=0; i<12;i++){
        meses_do_ano[i] = i+1;
    }
    // Printando todas as posições do vetor
    for(i=0; i<12;i++){
        printf("%d \n", meses_do_ano[i]);
    }
   return 0;
}
```

Olhando para os vetores você deve estar achando familiar, pois é vimos ele na seção de variáveis onde dissemos que uma variável do tipo **String** é um vetor de **Char**. E de fato eles são mesma coisa, contudo devemos ter alguns cuidados ao trabalhar com vetores Strings.

Precisamos ter em mente que toda String possui um caracter nulo representado por **\0**, o qual representa o fim da String indicando ao compilador que ela chegou ao fim. Portanto sempre que declaramos um vetor para armazenar uma palavra ou frase, devemos sempre deixar uma posição livre para que o compilador insira o **\0**. Desta forma se vamos declarar um nome com 5 letras, devemos iniciar um vetor com seis posições.

Ex:

Posição	0	1	2	3	4	5
nome[6]	B	R	U	N	O	**\0**

O nosso vetor tem seis posições mas nós só utilizamos cinco delas, e o compilador insere o **\0** por entender que esta string acabou.

Bora programar?!

1. Declare um vetor com os dias da semana e depois escreva esse dias no console. Dica: Para resolver esse exercício será necessário utilizar um vetor de duas dimensões, pesquise sobre Matrizes, e também peça ajuda aos professores).

2. Crie um vetor para armazenar as quatro notas de um aluno, que devem ser inseridas pelo usuário, calcule a média destas notas, e imprima no console o nome do aluno suas notas e média final, e caso a média seja igual ou maior que 70 o aluno foi aprovado, caso contrário ele foi reprovado. O resultado deve ser algo como:

```
Aluno: Bruno Segato
Nota 1 = 85
Nota 2 = 75
Nota 3 = 90
Nota 4 = 80
Media Final = 82,5
Situação = Aprovado!
```

3. Fazer um programa em "C" que lê uma string qualquer de no máximo 80 caracteres e imprime quantos caracteres tem o string.

4. Fazer um programa em "C" que lê uma string contendo palavras separadas por um espaço em branco cada e as imprime uma abaixo das outras.

5. Escreva um programa em que troque todas as ocorrências de uma letra L1 pela letra L2 em uma string. A string e as letras devem ser fornecidas pelo usuário.

6. Converta um número de 1 a 12 ao seu respectivo mês do ano: “janeiro”, “fevereiro”, ”março”...

FUNÇÕES

Até agora nós escrevemos nossos programas todos dentro da função **main** e como eles eram pequenos não tínhamos muitos problemas com isso, no entanto à medida que eles começam a crescer ficam cada vez mais complexos, e escrevê-los desta forma começa a trazer alguns problemas, como, otimização, dificuldade de interpretação e alguns outros.

Para nossa sorte ao usarmos funções nós podemos contornar vários destes problemas deixando nosso trabalho muito mais fácil. Mas você deve estar se perguntando, afinal o que são funções?

Funções são trechos de códigos que quando executados realizam um procedimento específico, dessa forma podemos dizer que elas são especialistas em realizar determinada operação.

Ao longo deste curso nós já utilizamos algumas funções, como por exemplo as funções **printf** e **scanf**, estas são funções que estão inclusas dentro da biblioteca **stdio.h** que incluímos no início dos nossos programas.

Essas funções que foram escritas por outros programadores, possuem todos os comandos necessários para que o nosso programa entenda que queremos escrever ou ler um valor do console, desta forma para utilizarmos nós apenas chamamos elas dentro do nosso código e passamos alguma informação para que ela execute esses procedimentos.

Vamos então entender como podemos escrever nossas próprias funções? Vamos começar pela sua estrutura que será algo como o seguinte exemplo:

```
tipo nome (parametros){
      instruções;
 return retorno da função ;
}
```

Onde:

- Tipo- É o tipo de retorno da função, este pode ser float, char, bool ou pode também ser do tipo void quando a função não possui retorno;
- Nome - É o nome que damos para a função;
- Parâmetros - São os parâmetros que a função espera receber, podemos também ter funções que não recebem nenhum parâmetro;
- Instruções - São os procedimentos que a função deve executar;

- Return - Para as funções que devem retornar algum valor, utilizamos o comando **return** passando para ele o que deve ser retornado, de acordo com o tipo de retorno da função;

Ainda está um pouco confuso? Vamos então ver na prática como usar uma função. Para isso suponha que queiramos escrever o sumário desta apostila separando os tópicos por "***". Poderíamos fazer algo como o programa abaixo.

```
#include <stdio.h>
int main () {
   printf("**********************\n");
   printf(" Bem vindo ao Dev2Be \n");
   printf("**********************\n");
   printf("Aula 1: Introducao   \n");
   printf("**********************\n");
   printf("Aula 2: Variaveis    \n");
   return 0;
}
```

Mas note que as linhas com asterisco sempre são repetidas, e cá entre nós esse processo é bastante chato não? Desta forma, como nós queremos sempre realizar o mesmo procedimento, nós poderíamos então recorrer a uma função que fizesse isso para nós. Vamos escrever nossa primeira função?

```
#include <stdio.h>
void escreve_asterisco(int numero_de_asteriscos){
    int i;
    for (i=0; i<= numero_de_asteriscos;i++){
        printf("*");
    }
    printf("\n");
}
int main () {
   escreve_asterisco(23);
   printf(" Bem vindo ao Dev2Be \n");
   escreve_asterisco(23);
   printf("Aula 1: Introducao   \n");
   escreve_asterisco(23);
   printf("Aula 2: Variaveis    \n");
   return 0;
}
```

Já que estamos no ritmo, vamos então escrever mais uma função para compreendermos melhor? Para isto vamos fazer uma função que retorna a soma de dois números inteiros.

Mas desta vez repare também que nós escrevemos o protótipo da função antes da função **main** e implementamos ela depois, isso é necessário pois se escrevermos a nossa função depois da main e não declararmos seu protótipo antes, o compilador nos retornará um erro. Isso acontece porque ele percorre linha a linha e inicia quando encontra a main, e caso não tenhamos declarado nossa função antes ele não saberá que a função `soma_inteiros` existe.

Diante disto você deve se perguntar, e porque não escrevemos todas nossas funções antes da main como no exemplo anterior? Bem, na prática poderíamos fazer isto, mas além de dificultar a leitura do nosso código isso pode ser considerado uma má prática.

```c
#include <stdio.h>


int soma_inteiros(int num1, int num2);// Protótipo da função

int main () {
  int s;
  //Aqui chamamos a função passando os valores que ela
  //deve somar, e atribuímos o seu retorno a variável S;
  s = soma_inteiros(10,10);
  printf("%d", s);
  return 0;
}
// Implementação da função
int soma_inteiros(int num1, int num2){
  int soma;
  soma = num1 + num2;
  return soma;
}
```

Funções são recursos muito importantes para o desenvolvimento dos nossos programas, portanto é importante que você compreenda bem este tópico, para isso sugiro que pesquise um pouco mais funções.

Presta atenção, cabeção!

Um conceito muito importante que devemos conhecer, especialmente quando estamos trabalhando com funções é sobre **escopo de variáveis**. De forma simples podemos ter dois tipos de escopo, local e global.

Quando declaramos uma variável fora do bloco de qualquer função, por exemplo logo abaixo das bibliotecas, elas assumem o escopo global, isto permite que qualquer função do nosso programa acesse e faça modificações nesta variável.

Já quando estas são declaradas dentro de uma função quer seja a main, ou uma função desenvolvida por nós, ela está limitada ao escopo local, ou seja somente aquela função pode acessar e modificar esta variável.

Contudo existem alguns recursos para que possamos modificar os tipos de acesso de uma variável entre funções. Pesquise sobre escopo de variáveis para se aprofundar mais neste tema.

Bora programar?!

1. Desenvolva um programa que peça dois números ao usuário, e utilizando funções realize as operações de soma, subtração, divisão e multiplicação.
2. Faça um programa que pergunte ao usuário, primeiro seu nome, depois seu sobrenome, e implemente uma função que faça a junção das duas Strings digitadas pelo usuário.

INTRODUÇÃO AO ARDUINO

A placa Arduino é uma plataforma de desenvolvimento de protótipos de hardware e software livre, ou seja, você pode fazer o que quiser com o equipamento, desde pequenos sensores de temperatura e umidade, até grandes robôs ou o que sua imaginação permitir.

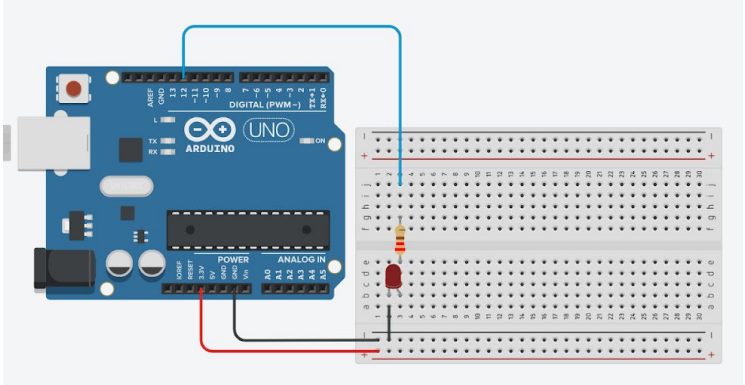
E como elas funcionam? De maneira simples estas placas nos fornecem portas para ligarmos outros dispositivos, os quais podem ser um led, um botão, um sensor dentre outros. Uma vez ligado estes dispositivos, nós podemos programar suas ações através dos códigos que desenvolvemos, podemos por exemplo fazer um programa que acenda um led quando pressionamos um botão, ou quando ficar escuro.

Mas, chega de falar. Que tal implementar o nosso primeiro programa no Arduino?

Para isso, neste curso nós iremos utilizar a plataforma Tinkercad, onde podemos simular o uso de um Arduino, para desenvolvermos e testarmos nossos projetos.

Vamos lá então?

O nosso primeiro programa, como sempre será o mais simples de todos, vamos apenas ligar um led, e fazê-lo piscar. Para isto precisamos primeiro realizar as conexões deste led com o Arduino, está ligação deve estar parecida com a imagem abaixo.



Vamos agora entender estas conexões.

Como já dissemos o Arduino nos fornece **portas** para conectarmos nossos dispositivos, o que fizemos aqui foi simplesmente ligar o nosso led a uma dessas portas para podermos controlar suas ações.

Mas note que para isso precisamos também alimentar nosso dispositivos com energia para que eles funcionem, assim nós ligamos o led ao **GND** (Negativo), e depois ligamos ele a porta, só que para isso utilizamos também um resistor, para garantir que a corrente elétrica que chega até o led, seja o suficiente para acendê-lo, mas não o queime.

Agora que já temos nossos dispositivos conectados vamos controlá-los através do nosso código.

Para isso precisamos entender a estrutura que os nossos códigos para o Arduino devem ter. De maneira simples devemos nos atentar as duas funções básicas a **Setup** e a **Loop**.

A *setup* é executada apenas uma vez, ela inicializa o Arduino e faz configurações iniciais no microcontrolador, como definir as portas de entrada ou saída, inicializar funções específicas como módulos ou sensores.

ATENÇÃO
Para entender melhor pesquise sobre alguns componentes eletrônicos básicos como resistores, transistores e outros.

A função *loop* se repete infinitamente e é responsável por executar as rotinas que aconteceram no Arduino como processamento de dados, obter leituras de sensores, piscar LEDs, transmitir dados, etc.

Outro ponto importante a ser destacado são os comandos que utilizaremos para controlar nossos dispositivos, pois são através deles que dizemos ao Arduino o que ele deve fazer.

De maneira geral existem diversos comandos, e dependo do dispositivo que estivermos utilizando, este pode ter alguns comandos específicos, mas não se preocupe com isto, a medida que você for desenvolvendo seus projetos você irá compreendê-los melhor.

Neste primeiro programa nós vamos apenas olhar para três deles que serão necessários para fazer o nosso led piscar. São eles:

- pinMode() - Declarado dentro do setup, ele serve para dizer ao Arduino a porta que vamos utilizar e também se utilizaremos ela como entrada (INPUT) ou saída (OUTPUT);
- digitalWrite() - Usado para controlar uma porta digital, com ele informamos ao Arduino se queremos que esta porta esteja ligada (HIGH) ou desligada (LOW);
- delay() - Serve para fazer com que o programa espere um tempo (definido em milissegundos), até que execute outra ação;

Vamos então ver como fica o nosso código para piscar um led?

```
void setup(){
  pinMode(12, OUTPUT);
}

void loop(){
  digitalWrite(12, HIGH);
  delay(1000);
  digitalWrite(12, LOW);
  delay(1000);
}
```

Primeiro dentro da função setup, dizemos ao Arduino que o nosso led está conectado na porta 12 e informamos que ela será uma saída. Depois dentro da função loop nós pedimos ao arduino que acenda o led através do comando `digitalWrite(HIGH)`, pedimos para que ele espere 1 segundo e depois apague o led com comando `digitalWrite(LOW)`.

Presta atenção, cabeção!

As placas de Arduino possuem portas digitais e analógicas, o modelo Arduino Uno, que vamos utilizar no nosso curso, possui 14 portas digitais(0-13) e 6 portas analógicas(A0-A5). As portas digitais fornecem valores entre 0 (LOW) e 1(HIGH), ou seja elas alteram seu estado entre ligado e desligado. Já as portas analógicas trabalham com uma variação mais sensível de tensão, sendo assim elas permitem ler valores em intervalos.

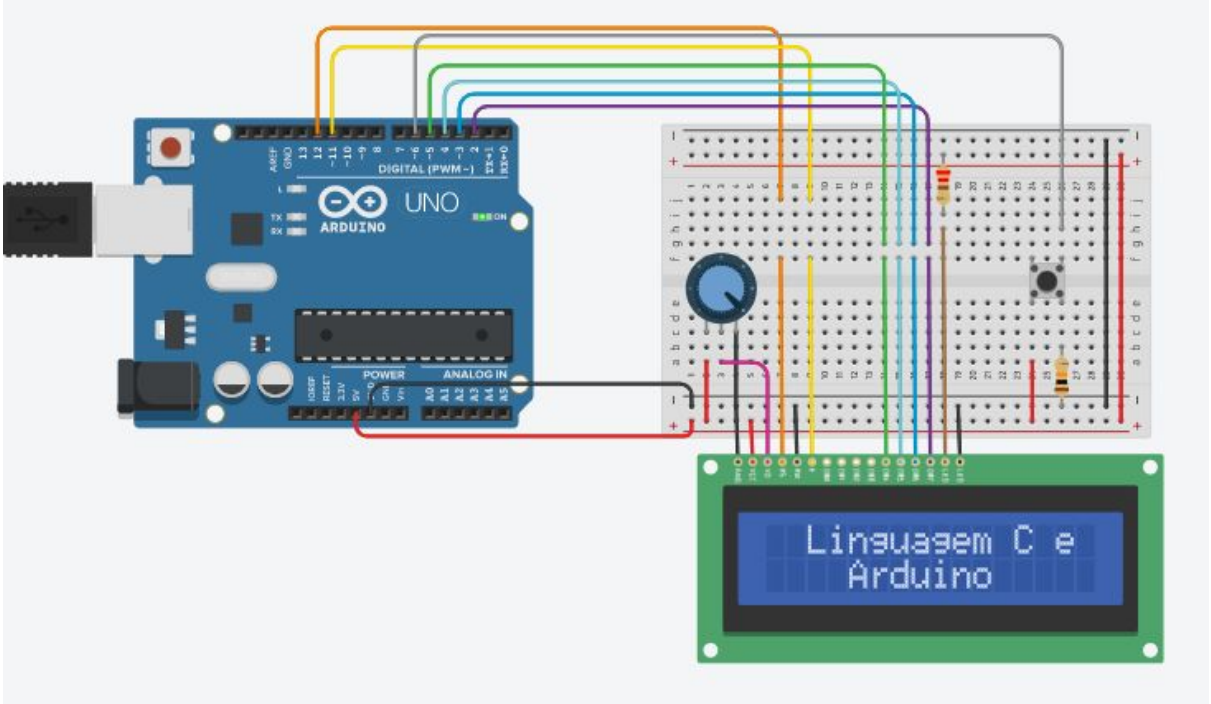
Para esclarecer pense na seguinte situação, se queremos verificar o estado de um botão, nós apenas precisamos saber se ele está pressionado ou não certo? Por isso fazemos uma leitura digital, passando 1 se ele estiver pressionado e 0 caso não esteja. Mas se queremos ler uma temperatura geralmente nós queremos saber qual a temperatura atual, por exemplo 36,5; 36,6; 34,2 e assim por diante, portanto nós precisamos dividir essa leitura em intervalos menores, que vão variando entre um mínimo até o máximo.

Bora programar?!

1. Desenvolva um semáforo utilizando os conhecimentos que aprendemos durante esta seção.

2. Utilizando um sensor fotoelétrico, desenvolva um programa que acenda um led quando a luminosidade for igual ou menor que 0;

PRIMEIROS PASSOS COM O LCD



Agora que já vimos alguns conceitos básicos do Arduino, vamos aprender um pouco sobre o dispositivo lcd.

Este dispositivo é utilizado como um monitor, assim como nas seções anteriores nós usávamos o console para exibir as informações dos nossos programas, no Arduino podemos utilizar um display lcd para apresentarmos estas informações de forma visual.

Para isso precisamos primeiro ligar nosso lcd ao Arduino. Este processo pode ser um pouco complicado no início, mas ” PARA NOSSA ALEGRIAAAA”, o Tinkercad já fornece alguns modelos prontos, basta apenas selecionarmos e implementarmos o restante dos nosso código.

Entretanto se você pretende desenvolver seus projetos com lcd é extremamente importante que você compreenda como são feitas estas conexões e para isso, você pode pesquisar na internet por vídeos e tutoriais. Mas recomendamos fortemente que você acesse a documentação oficial do Arduino, pois ela possui explicações sobre uso geral do Arduino, além de explicação sobre várias bibliotecas e dispositivos.

Para isso você pode acessá-la através do link:

https://www.arduino.cc/reference/pt/

Vamos agora conhecer alguns comandos do lcd, para que possamos desenvolver nossos programas.

- `lcd.begin()` – Inicializa o lcd, para que possamos utilizá-lo;
- `lcd.print()` – Utilizado para escrever no lcd, similar a função printf;
- `lcd.setCursor()` – Comando para posicionar o cursor na linha e coluna que desejamos escrever;
- `lcd.clear()` – Este comando apaga tudo que estava sendo mostrado no lcd antes de sua execução;
- `lcd.write()` – Similar ao lcd.print, mas utilizado para escrever dados, como por exemplo os personagens que utilizaremos para desenvolver nosso jogo.

Existem muitos outros comandos que você pode conferir no link acima, por hora vamos aplicar o que aprendemos para escrever um "Oi, mundo" no lcd do Arduino?

Para isso nós vamos procurar por um modelo já montado de lcd no Tinkercad, tendo nosso lcd configurado vamos então ao código, por se tratar de um simples "Oi, mundo", ele não possui muita complexidade e deve ser algo como o código abaixo:

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  lcd.begin(16, 2);
}
void loop() {
  lcd.setCursor(0,0);
  lcd.print("Oi, mundo!");
}
```

Você deve estar pensando, hum acho que consigo entender este código. Isto mesmo já vimos estes conceitos, lendo ele de cima para baixo nós fizemos os seguintes passos, incluímos uma biblioteca para trabalhar com o lcd, indicamos para o Arduino quais portas o lcd irá utilizar, dentro do setup iniciamos o lcd, e depois dentro do loop posicionamos o cursor e pedimos para escrever "Oi, mundo".

Bora programar?!

1. Desenvolva um programa que fique deslizando o seu nome na tela do lcd, ele deve começar a escrever em um lado da tela e sumir no outro continuamente.
2. Pesquise como utilizar botões no Arduino, e use estes conhecimentos para desenvolver um programa com dois botões, que escreva seu nome em uma das linhas e quando pressionado o primeiro botão ele troque seu nome de linha, e quando pressionado o segundo botão ele retorne para linha anterior.

DESENVOLVIMENTO DO JOGO

Até aqui nós estudamos conceitos básico da linguagem e também vimos uma breve introdução ao Arduino, estes conhecimentos serão a base necessária para que possamos desenvolver o nosso jogo. Mas eu sei que você deve estar impaciente se perguntando, e quando eu vou desenvolver meu jogo?

Bem vamos começar a parte mais divertida então?

Mas antes vamos definir algumas coisas, como os conceitos que aprendemos até aqui, o nosso jogo também será algo simples, não vamos desenvolver o próximo Valorant, mas ainda assim, antes de começar a escrever o código do nosso jogo, nós precisamos ter em mente, que tipo de jogo ele será, como ele vai funcionar, ou seja quais as mecânicas de jogo, qual a história por trás do nosso game, etc.

> Existe uma gama enorme de conhecimentos sobre desenvolvimento de jogos que envolvem vários processos, um bom ponto de partida se você deseja saber mais sobre esta área é pesquisar sobre Game Design Document (GDD).

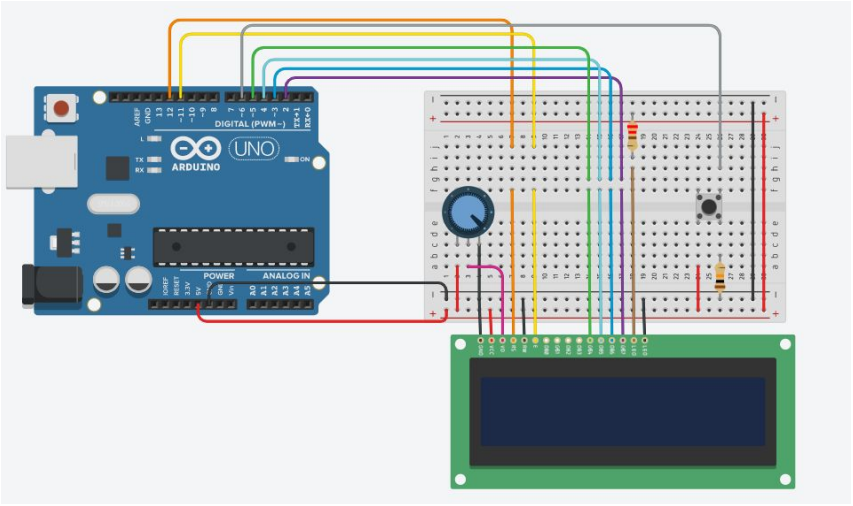
Portanto as definições do nosso jogo são as seguintes:

- Tipo de jogo - Ação e aventura, run and jump (estilo dinosaur game do google);
- Mecânicas - Pulo, pontuação e colisão;
- História - Será criada em sala, através de uma dinâmica;

Com isto já podemos ter uma ideia de como, nosso jogo será correto? Então vamos começar a desenvolvê-lo.

PASSO 1 - MONTANDO E CONFIGURANDO O HARDWARE

A partir do modelo de conexão pronto do lcd no tinkercad, adicione um botão.

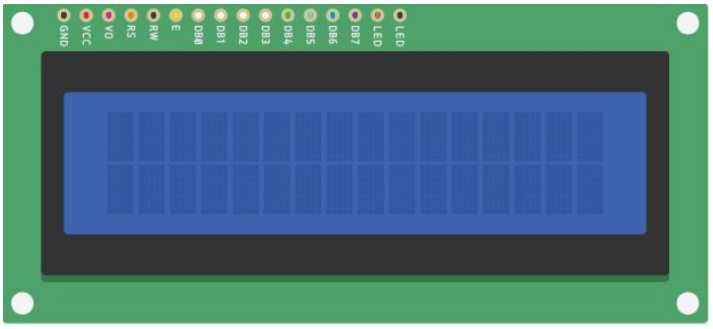


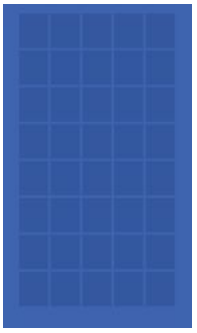
No código certifique-se de incluir a biblioteca LiquidCrystal, setar suas entradas, configurar a entrada do botão no setup e de inicialização do lcd.

```
#include <LiquidCrystal.h>
#define botao 6

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
      pinMode(botao, INPUT);
      lcd.begin(16, 2);
}
```

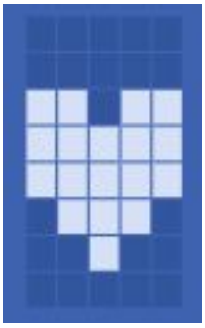
PASSO 2 - CRIANDO PERSONAGENS

Para criar os personagens precisamos entender que o lcd disponibilizado no Tinkercad possui 16 x 2 caracteres(espaços), como é possível ver na imagem abaixo. Fazendo um recorte de um desses caracteres, é possível verificar que eles são compostos por 5 x 8 pixels, ou seja 5 colunas e 8 linhas, formando 40 quadradinhos que podem estar acesos ou não.





Entendendo isso, podemos criar caracteres personalizados, o arduino suporta até 8 desses caracteres. Para personalizá-los cada pixel aceso recebe o valor de 1 e cada pixel apagado recebe o valor 0. Veja um exemplo de caracteres personalizados.



```
byte heart[8] = {
      B00000,
      B00000,
      B11011,
      B11111,
      B11111,
      B01110,
      B00100,
      B00000
};
```

O site https://maxpromer.github.io/LCD-Character-Creator/ facilitará a nossa vida. Basta acender ou apagar os pixels, dar um nome ao nosso desenho e copiar para nosso código em arduíno.

Precisamos agora inserir nosso caractere personalizado no código, para isso, declaramos a variável do tipo byte, em seguida inserimos a função `lcd.createChar(posição,nome)` no setup para criar o caractere. Por fim, para imprimir na tela utilizamos a função `lcd.write(byte(posição))`. Veja o exemplo abaixo:

```
byte heart[8] = {
     B00000,
     B00000,
     B11011,
     B11111,
     B11111,
     B01110,
     B00100,
     B00000
};

void setup(){
    lcd.begin(16,2);
    lcd.createChar(0,heart);

    lcd.write(byte(0));
}
```

PASSO 3 - MOVIMENTANDO PERSONAGENS

Com nosso personagem criado e posicionado no lcd, chegou a hora de criarmos a lógica de movimentação. Para mover os objetos pela tela de lcd é necessário mudar as posições **x**(coluna) e **y**(linha) do caractere a cada iteração do loop ou a cada ação do botão. Por exemplo, se iniciamos o personagem na posição (0,1), ou seja coluna 0 e linha 1, para realizar o movimento de pulo, queremos que o personagem seja impresso na posição (0,0). Mas não podemos esquecer de limpar a posição anterior ao movimento, caso contrário, o objeto aparece duplicado nos dois espaços. Veja o código a seguir:

```
void pulo(){
    pypersonagem=0;
    lcd.setCursor(0,1);
    lcd.print(" ");
    lcd.setCursor(0,pypersonagem);
    lcd.write(byte(0));
}
```

Me diz uma coisa, isso tudo está muito fácil né? Então agora nós vamos deixar com vocês a parte de implementar a movimentação com o acionamento do botão e também criar e fazer a movimentação do obstáculo. Utilize dos conhecimentos de programação em C, pesquise e peça ajuda aos professores se necessário.

Presta atenção cabeção!

Evite o uso da função `lcd.clear()` para apagar a posição anterior do objeto. Apague apenas o caractere movimentado utilizando `lcd.print(" ")`. O excesso de lcd.clear() tornará o jogo lento.

PASSO 4 - COLISÃO

Já temos todos os nossos objetos do jogo e suas movimentações. Então vamos pensar em como desenvolver a lógica de colisão? A colisão ocorre quando dois ou mais objetos ocupam a mesma posição do lcd. Por isso, para verificar se os objetos colidiram vamos trabalhar com as variáveis x e y de ambos caracteres. Veja o exemplo a seguir:

```
if(pxpersonagem == pxobjeto && pypersonagem == pyobjeto);
```

Diante dessa informação é possível tomar diversas decisões. O jogador pode acumular pontos, o personagem pode recuperar energia, a dificuldade do jogo pode aumentar, o personagem pode criar super-poderes e uma colisão pode ocasionar o game-over também.

A criatividade é toda sua!

PASSO 5 - PONTUAÇÃO

Qual o sentido de um jogo se ele não possuir um objetivo? Precisamos adicionar alguns critérios de vitória ou derrota ao nosso jogo. Uma das formas básicas de implementar um objetivo é criar um sistema de pontuação. A lógica de pontuação vai depender da proposta do seu jogo, se será acrescido na pontuação após desviar o obstáculo, pegar algum item, após certo tempo, etc.

No jogo que estamos propondo a pontuação é baseada em fazer o personagem conseguir pular o obstáculo, somando 1 ponto. Diante disso, criamos uma condição de verificação da posição **x** do obstáculo de forma que, quando passar do personagem, sem colidi-lo, ou seja, **pxobjeto** for -1, seja acrescido 1 a pontuação.

```
if (pxobjeto == -1){
      score++;
}
```

Para finalizar o seu jogo que tal acrescentar a velocidade do seu obstáculo e ir aumentando aos poucos? Outra função muito legal de se implementar é o game over. Você pode criar uma função, chama-la caso o player perca o jogo e imprimir uma mensagem com a pontuação. Ou quem sabe fazer um menu ao iniciar? As possibilidades são inúmeras!

É isso pessoal! Esperamos que tenham gostado do curso e do material que preparamos pra vocês. Daqui em diante você pode usar sua criatividade e o conhecimento que adquiriu para explorar funcionalidades e possibilidades pro seu jogo.