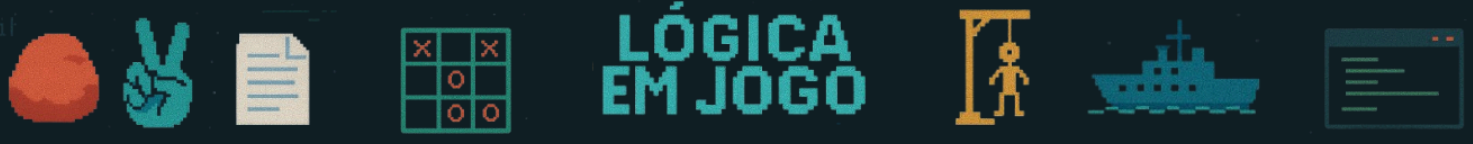

Sessão 07

Jogo da forca em arquivos

Professor Thiago Goveia

1.0 Objetivo da Sessão

Nesta sessão, vamos transformar nosso jogo de uma experiência estática para uma dinâmica e com rejogabilidade.

1. **Versão 3:** Vamos parar de usar uma única palavra "hardcoded" (fixa no código) e criar um **banco de palavras interno** (uma matriz) em um **arquivo separado (.h)**, selecionando uma palavra aleatoriamente.
2. **Versão 4:** Vamos evoluir o banco de palavras para que ele seja lido de um **arquivo externo (.txt)**, tornando nosso jogo fácil de atualizar e expandir.

2.0 Focos Conceituais Chave

1. **Aleatoriedade:** Como gerar números aleatórios em C para sortear a palavra.
2. **Matrizes de char:** Como armazenar uma lista de palavras (strings) em C.
3. **Modularização:** O uso de arquivos de cabeçalho (.h) para declarar funções (protótipos) e arquivos .c para implementá-las.
4. **Manipulação de Arquivos:** Como abrir, ler e fechar um arquivo de texto (.txt) em C.

Para as versões de hoje, precisaremos adicionar duas novas bibliotecas no topo do nosso arquivo .c:

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>  // Para converter para maiúsculas
#include <stdlib.h> // Para aleatoriedade e para sair do programa
#include <time.h>   // Para aleatoriedade
```

3.0 Versão 3 (Banco de Palavras Interno e Aleatoriedade)

Vamos parar de usar `char palavra_secreta[7] = "LOGICA";` e, em vez disso, sortear a palavra de uma lista.

3.1 Revendo aleatoriedade (rand/srand)

Vamos precisar recorrer à função `rand()`, assim como fizemos no jogo Pedra, Papel, Tesoura, para sortear a jogada do computador. Para sortear, usamos duas funções da `<stdlib.h>`:

1. `srand(time(NULL))`: Esta função "semeia" o gerador de números aleatórios. Ela usa a hora atual (`time(NULL)`) para garantir que, a cada vez que o programa rodar, a sequência de números seja diferente.
    - **REGRA DE OURO:** Esta função deve ser chamada **APENAS UMA VEZ** no programa, logo no início da sua função `main()`.
2. `rand()`: Esta função retorna um número aleatório. Para limitar o número a um intervalo (ex: de 0 a 9), usamos o operador módulo (`%`).
    - **Exemplo:** `int indice_aleatorio = rand() % 10;` (gera um número entre 0 e 9).

3.2 Conceito: Matriz de Palavras (char)

Para armazenar uma lista de palavras, usamos uma matriz 2D de `char`. A sintaxe é `char nome_da_matriz;`.

- **Exemplo:** `char palavras[3][10] = {"CASA", "BOLA", "NAVIO"};`

3.3 Esqueleto de Código (V3)

Vamos modificar nossa função `main()` e o início do jogo.

```
//... includes no topo do arquivo...
int main() {
    // PASSO 1: Chamar srand() UMA ÚNICA VEZ
    // PASSO 2: Definir o banco de palavras (Matriz 2D)
    // PASSO 3: Obter o total de palavras no banco
    // PASSO 4: Sortear um índice aleatório
```

```
    // PASSO 5: Copiar a palavra sorteada para a variável de jogo
    // PASSO 6: O jogo continua como antes (V2)

    return 0;
}
```

3.4 Criando uma biblioteca de palavras

Já vimos a importância do uso de funções para a organização e modularização do código. É possível aprimorar esta modularização, organizando o código em diferentes arquivos.

Um arquivo .h, é um arquivo de cabeçalho e possibilita a criação de bibliotecas e assim, o reuso de código.

Atenção: O ideal é que o arquivo .h contenha apenas os protótipos das funções e a implementação de cada uma seja feita em arquivos .c de mesmo nome. **Por simplicidade**, faremos toda a implementação dentro do arquivo .h.

```
// banco_palavras.h

// PASSO 1: "Include Guards"
// Evitam que o compilador inclua este arquivo mais de uma vez
// e gere erros de "definição duplicada".,
#ifndef BANCO_PALAVRAS_H
#define BANCO_PALAVRAS_H

#include <time.h>
#include <string.h>

#define TOTAL_PALAVRAS 3

// PASSO 2: Definir o banco de dados
// Esta matriz é CONSTANTE. Nenhuma função pode alterá-la.
   const char PALAVRAS[TOTAL_PALAVRAS][20] = {
        "SATURAÇÃO",
        "HEMOGLOBINA",
        "SOCIALISMO"
    };
```

```
//PASSO 3: Implementar as funções

void inicializar_aleatoriedade() { //Deve ser chamada na main
    // Esta lógica agora vive aqui, e não em main.c
    srand(time(NULL));
}

void sortear_palavra(char* palavra_destino) {
    int indice_sorteado = rand() % TOTAL_PALAVRAS;

    // Copia a palavra sorteada para o ponteiro 'palavra_destino'
    // que foi passado por main.c
    strcpy(palavra_destino, banco_palavras[indice_sorteado]);
}

// Fim do Include Guard
#endif
```

No main.c, adicione a linha `#include "banco_palavras.h"` e faça as chamadas e modificações necessárias.

> **Hora do jogo!** Teste o jogo com a sua equipe!
> - Identifiquem os defeitos e as possibilidades de melhorias.
> - Façam o commit no **Github**, descrevendo as funcionalidades e limitações desta versão

4.0 Versão 4 (Lendo Palavras de um Arquivo `.txt`)

Manter as palavras dentro do código `.c` não é prático. O ideal é mantê-las em um arquivo `.txt`, em que qualquer pessoa possa adicionar ou remover palavras.

4.1 Conceito: Leitura de Arquivos (`FILE*`)

Para ler arquivos, usamos um ponteiro especial chamado `FILE` e três funções principais da `<stdio.h>`:

1. `FILE* f;`: Declara um ponteiro para um arquivo.

2. `f = fopen("nome_arquivo.txt", "r");`: Abre o arquivo. O `"r"` significa "modo de leitura" (read).
3. `fscanf(f, "...",...)`: Funciona como o `scanf`, mas lê *do arquivo* (`f`) em vez do teclado.
4. `fclose(f);`: Fecha o arquivo. Essencial para liberar os recursos do sistema.

4.2 Formato do `palavras.txt`

Para o nosso código saber quantas palavras sortear, vamos adotar um formato simples: **A primeira linha do arquivo deve conter o número total de palavras.**

Crie um arquivo chamado `palavras.txt` na *mesma pasta* do seu executável (`.exe`) com este conteúdo:

```
5
PARALELOGRAMO
HIPOTENUSA
PROPAROXÍTONA
EXONERAÇÃO
PINDAMANHANGABA
```

4.3 Esqueleto de Código (V4)

Vamos transformar a lógica de seleção de palavra (V3) em uma nova função.

```
// Criamos uma função dedicada a escolher a palavra do arquivo
void escolhe_palavra(char* palavra_secreta) {
    // PASSO 1: Abrir o arquivo de palavras
    FILE* f = fopen("palavras.txt", "r");

    // PASSO 2: Verificar se o arquivo foi encontrado
    if (f == NULL) {
        printf("Erro: Banco de palavras nao encontrado.\n");
        exit(1); // Sai do programa se não encontrar o arquivo
    }
```

```
    // PASSO 3: Ler o número total de palavras (a primeira linha)
    int total_palavras = 0;
    fscanf(f, "%d", &total_palavras);

    // PASSO 4: Sortear um índice aleatório (igual à V4)
    int indice_sorteado = rand() % total_palavras; [14]

    // PASSO 5: Ler as palavras do arquivo até chegar na sorteada
    // fscanf() lê uma palavra de cada vez.
    // Vamos "pular" as palavras até o índice sorteado.
    for (int i = 0; i <= indice_sorteado; i++) {
        fscanf(f, "%s", palavra_secreta);
    }

    // PASSO 6: Fechar o arquivo
    fclose(f);
}
```

Hora do jogo! Teste o jogo com a sua equipe!

- Identifiquem os defeitos e as possibilidades de melhorias.
- Façam o commit no **Github**, descrevendo as funcionalidades e limitações desta versão

5.0 DICA BÔNUS

Normalização: Verifique se o seu código funciona para maiúsculas e minúsculas. Certamente não! Para que isso seja possível, é necessário padronizar(ou normalizar) as letras, isto é, convertê-las para minúsculas ou maiúsculas o chute para maiúsculo.

Converta as letras do `chute` e da `palavra_secreta` para `maiúsculas` (requer `#include <ctype.h>`) e verifique novamente!

```
chute = toupper(chute); //Converte chute para maiúsculas
```

Hora do jogo! Teste o jogo com a sua equipe!

- Identifiquem os defeitos e as possibilidades de melhorias.
- Façam o commit no **Github**, descrevendo as funcionalidades e limitações desta versão

7.0 Conclusão da Sessão 06

Ao final desta sessão, seu Jogo da Forca estará muito mais robusto:

1. **V3:** Você aprendeu a usar matrizes 2D para guardar dados e a criar e usar arquivos `.h`.
2. **V4:** Você aprendeu a ler dados de um arquivo externo usando `fopen()`, `fscanf()` e `fclose()`, permitindo que qualquer pessoa adicione novas palavras ao jogo sem precisar recompilar o código.