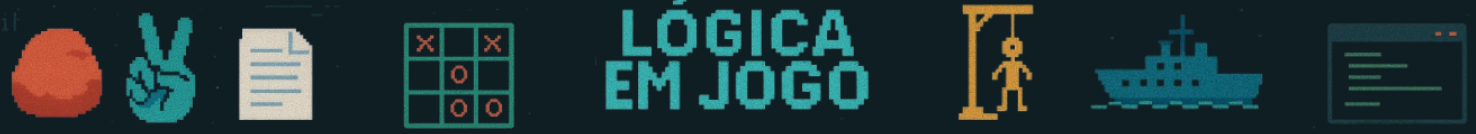

Sessão 06

Jogo da forca

Professor Thiago Goveia

1.0 Objetivo da Sessão

Nesta sessão, construiremos a fundação do nosso Jogo da Forca. Diferente do "Pedra, Papel, Tesoura", nosso foco muda da lógica condicional (decisão) para o **gerenciamento de estado** (rastreamento de dados ao longo do tempo). Implementaremos as três primeiras versões do jogo, evoluindo de uma lógica de iteração manual para uma otimização com bibliotecas C.

2.0 Focos Conceituais Chave

1. **A "String" em C:** Entender que *strings* são, na verdade, vetores (ou *arrays) de caracteres terminados por um caractere nulo* (`\0`).
2. **Gerenciamento de Estado:** Usar múltiplos vetores para rastrear a palavra secreta, o *display* (`_ _ _`) e as letras já tentadas.
3. **Iteração vs. Otimização:** Comparar a iteração manual (`for`) com o uso de funções de biblioteca (`string.h`).

3.0 O "Chassi" do Jogo (Fundamentos)

Antes de escrever a V1, precisamos das variáveis que controlam o fluxo do jogo.

Definindo o Estado: Estas são as variáveis que controlam o *loop* principal:

```
int erros = 0;
int max_tentativas = 6;
int acertou = 0; // Flag booleana (0 ou 1)
int enforcou = 0; // Flag booleana (0 ou 1)
```

O Loop Principal (`while`): O coração do jogo. Ele continuará executando *enquanto* o jogador não tiver ganho (`!acertou`) E não tiver perdido (`!enforcou`).

```
while (!acertou &&!enforcou)
{
    // 1. Mostrar estado (forca, display, letras tentadas)
    // 2. Pedir entrada (chute do usuário)
    // 3. Processar entrada (verificar acerto, erro ou repetição)
    // 4. Atualizar estado (incrementar erros, atualizar display)
    // 5. Verificar condições de término (acertou = 1? enforcou = 1?)
}
```

Ponto Crítico: Devemos garantir que, a cada rodada, o estado seja atualizado de forma **mutuamente exclusiva**, isto é, um chute não pode ser um acerto *e* um erro ao mesmo tempo.

4.0 Versão 1: A Palavra Estática

Objetivo: Implementar a mecânica básica com uma palavra "hardcoded" (fixa no código), focando na iteração manual de *arrays*.

Bibliotecas Necessárias: `<stdio.h>` e `<string.h>`

Estruturas de Dados (Arrays):

```
char palavra_secreta[4] = "LOGICA";
char display[4];
int tamanho = 0;
```

Conceito-Chave: `strlen()` e o Terminador Nulo (`\0`) Em C, *strings devem* terminar com `\0`. A função `strlen()` (da `string.h`) não conta o tamanho do *array*, mas sim o número de caracteres *antes* do primeiro `\0`.

A inicialização `char palavra = "LOGICA";` automaticamente adiciona o `\0` no final.

Lógica de Jogo (V1):

1. **Obter Tamanho:** `tamanho = strlen(palavra_secreta);`
2. **Inicializar `display`:** Devemos preencher o `display` com `_` e, crucialmente, terminá-lo com `\0` para que ele também seja uma *string* válida.

```
for (int i = 0; i < tamanho; i++) {
    display[i] = '_';
}
display[tamanho] = '\0';
```

Loop de Chute (Dentro do `while`):

Pedir a letra:

```
char chute;
printf("Digite uma letra: ");
scanf(" %c", &chute); // O espaço antes de %c é vital!
```

Flag de controle da rodada (assumimos que o jogador errou até prova em contrário): `int errou_nesta_rodada = 1;`

Processar Chute (Iteração Manual): Varremos a `palavra_secreta` letra por letra, conforme o esboço a seguir:

```
for (...) { // Laço que percorre a palavra letra a letra
    if (...) { // Verifica se a letra atual é igual ao cute
        // Atualizar o display para mostrar a letra
        // Atualizar errou_nesta_rodada para falso (0)
    }
}
```

Atualizar Estado do Jogo: `if (errou_nesta_rodada == 1) { erros++; }`

Condições de Término:

```
// Se o display for igual à palavra secreta, o jogador venceu
if (...) { // Verificar com strcmp se as palavras são iguais
    acertou = 1;
}
// Se os erros atingirem o máximo, o jogador perdeu
if (erros == max_tentativas) {
    enforcou = 1;
}
```

Hora do jogo! Teste a Versão 01 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 Versão 2: Gerenciamento de Estado

Objetivo: Corrigir um *bug* da V1: o jogador é penalizado (perde uma vida) se digitar uma letra repetida.

Nova Estrutura de Dados: Precisamos de um *array* para armazenar o histórico de chutes.

```
char letras_tentadas[7]; // 6 tentativas + \0
int num_tentativas = 0;
```

Lógica de Jogo (V2) Refatorada: Modificamos o "Passo 3" (Processar Chute) do *loop* principal.

1. **Receber Chute:** `scanf(" %c", &chute);`
2. Verificação de Repetição : *Antes* de processar o chute, verificamos se ele já está no *array* `letras_tentadas`.

```
int ja_tentou = 0;
for (...) { // Laço para percorrer cada uma das letras tentadas
    if (...) { // Verifica se o chute é igual à letra tentada
        // Atualiza a variável ja_tentou
        break;
    }
}

if (ja_tentou) {
    printf("Voce ja tentou esta letra!\n");
    continue; // Pula esta rodada e volta ao início do 'while'
}
```

Adicionar ao Histórico: Se não for repetida, adicionamos a letra ao histórico.

```
letras_tentadas[num_tentativas] = chute;
num_tentativas++;
letras_tentadas[num_tentativas] = '\0'; // inclui o \0 ao final
```

Processar Chute (Iteração Manual): Idêntico à V1 (o `for` que atualiza o `display` e a *flag* `errou_nesta_rodada`).

Atualizar Estado: Idêntico à V1 (`if (errou_nesta_rodada) { erros++;}`).

Condição de Vitória (Alternativa): Embora `strcmp` (da V1) funcione, uma forma mais robusta é verificar se ainda existem `_` no *display*. Se não houver, o jogador venceu.

```
int letras_restantes = 0;
for (...) { //Laço para percorrer o display
    if (...) { //Verifica se há '_' no display
        letras_restantes = 1; // Ainda faltam letras
        break;
    }
}
```

Hora do jogo! Teste a Versão 01 com a sua equipe!

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

6.0 Versão 3: Otimização e Eficiência (`strchr`)

Objetivo: Refatorar a V2 para usar funções de biblioteca (`string.h`), tornando o código mais legível e eficiente.

Conceito-Chave: `strchr()` A função `strchr(string, caractere)` procura a *primeira* ocorrência de um caractere em uma *string*.

- Retorna um *ponteiro* (um endereço de memória) se encontrar.
- Retorna `NULL` (zero) se não encontrar.

Em um `if`, `if (strchr(string, c) != NULL)` é verdadeiro se a letra existir.

A Armadilha de "BANANA": Não podemos *substituir* o *loop* `for` pelo `strchr`. Se o chute for 'A', `strchr` só encontrará o *primeiro* 'A'. Precisamos de um *loop* para atualizar *todos* os 'A's.

O Padrão "Check-then-Update":

1. **Check (Checar):** Usamos `strchr` para uma verificação rápida (Acertou ou Errou?).
2. **Update (Atualizar):** Se acertou, *ainda* usamos o *loop* `for` para atualizar *todas* as ocorrências.

Verificação de Repetição (com `strchr`): Mais elegante que o *loop* da V2.

```
if (strchr(letras_tentadas, chute)!= NULL) {
    printf("Repetido!\n");
    continue;
}
```

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 a Versão 01 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, teremos um Jogo da Forca funcional e robusto.

- **V1:** Estabeleceu o *loop* principal e a lógica de exibição.
- **V2:** Implementou o gerenciamento de estado (histórico de chutes) para evitar penalidades injustas.
- **V3:** Refatorou o código para usar funções C padrão (`strchr` e `toupper`), tornando-o mais limpo e eficiente.