


Trabalho Prático do Grau B – Algoritmos e Programação: Fundamentos

## Simulador de Combate por Turnos



Fonte da Imagem: Imagem provisória

### Individual ou grupos de até 3 participantes

 **DATA DE ENTREGA:** até 22/06/2025 (Domingo), via Moodle

### Instruções para envio do trabalho:

Apenas **1 integrante do grupo** deve enviar o link para o repositório do projeto na atividade aberta no Moodle até as **23h59min do dia 18/06/2025**. O diretório do projeto deve conter:

- O código fonte do trabalho, desenvolvido em C/C++.
- Um arquivo **LEIAME.md** com o nome completo dos integrantes do grupo e instruções de uso do programa. Você pode se basear [neste template](#) para escrever o README.

### Introdução

Nas profundezas das Terras Selvagens, dois grupos de aventureiros se enfrentam por honra, tesouros ou simplesmente sobrevivência. Você foi convocado(a) para desenvolver um **sistema de simulação de combates por turnos** entre essas equipes, aplicando os conceitos aprendidos em aula durante todo o semestre.

### Objetivo

O objetivo deste trabalho é exercitar os conceitos estudados até o momento, especialmente a criação e uso de arrays, structs e funções. Para isso, os estudantes devem **desenvolver um programa em C que simule o combate entre duas equipes**, considerando a descrição e regras abaixo. O código **deve ser**

desenvolvido em C/C++, utilizando o conteúdo trabalhado em aula, e deve estar corretamente indentado e comentado.

### Descrição do Problema

Cada equipe deve conter **5 aventureiros**, cada um pertencente a uma das **cinco classes disponíveis**.

Todos os personagens começam com **100 pontos de vida (saúde)**.

#### Tabela de Classes

Classe	Ataque	Defesa	Habilidade Especial
Guerreiro	20	10	Golpe Crítico (dano dobrado, 20% de chance)
Mago	30	5	Bola de Fogo (ignora defesa, 25% de chance)
Caçador	18	8	Ataque Duplo (ataca duas vezes, 15% de chance)
Paladino	15	12	Regeneração (recupera 20% da vida perdida se sobreviver ao ataque, 30% de chance)
Bárbaro	25	6	Nunca erra o ataque (ignora chance de falha sempre)

### Regras de Combate

- O time que **inicia a batalha** deve ser sorteado no início do programa.
- As batalhas ocorrem por **rodadas alternadas**, onde cada time executa **um ataque por turno**.
- A **seleção do personagem atacante** de cada time deve ser baseada na **maior razão saúde/ataque** entre os membros vivos da equipe.
- O **alvo** do ataque é sorteado aleatoriamente entre os membros vivos da equipe adversária.
- O **dano básico** é calculado como:

$$\text{dano} = \text{ataque}_{\text{atacante}} - \text{defesa}_{\text{defensor}}$$

Se o dano for negativo, será considerado **zero**.

- Ao longo do turno, é necessário verificar:
  - **20% de chance de erro do ataque** (ataque é ignorado)
  - **20% de chance de falha da defesa** (dano não é reduzido pela defesa)
  - **Habilidade especial da classe**, conforme porcentagem definida
- **Paladino** recupera 20% da vida perdida caso sobreviva a um ataque e ative sua habilidade.
- A **batalha termina** quando todos os personagens de uma das equipes estiverem com 0 de vida. Então deve-se apresentar o time vencedor (ou empate, caso houver).
- Após **cada rodada** (um ataque por time), deve-se imprimir o estado de **todos os membros de ambas as equipes**, incluindo vida, classe e status.

### Exemplo de Log de Combate:

```
>>> Rodada 3:
Time 1 – Bárbaro (Vida: 72) ataca Mago do Time 2 (Vida: 34)
→ Ataque bem-sucedido
→ Dano aplicado: 19
→ Habilidade especial do Mago ativada: ignorar defesa
→ Mago agora com 15 de vida

Time 2 – Paladino (Vida: 85) ataca Caçador do Time 1 (Vida: 52)
→ Ataque errou! Nenhum dano aplicado.

>>> Estado Atual:
Time 1:
    Guerreiro: Vida 0
    Caçador: Vida 52
    Mago: Vida 100
    Paladino: Vida 68
    Bárbaro: Vida 72

Time 2:
    Guerreiro: Vida 0
    Caçador: Vida 0
    Mago: Vida 15
    Paladino: Vida 85
    Bárbaro: Vida 100
```

### Requisitos principais

- Representar cada personagem com uma struct Personagem contendo:
  - o classe, vida, ataque, defesa, habilidade\_ativa
- Usar **arrays de structs** para representar os dois times
- Implementar funções para evitar repetição de código em ações bem definidas, como inicializar as equipes, calcular dano etc..

### Referências

EM BREVE!

**BOM TRABALHO!** 😊

E lembre-se: problemas grandes podem ser resolvidos quebrando-os em problemas menores!!!