



UNIVERSIDADE FEDERAL DE MINAS GERAIS  
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

**TRABALHO PRÁTICO:**  
**Programa e Desenvolvimento de Software 1 - DCC- 2024/1**

Matheus Antonio Barbosa Santos

Julho 2024

## Sumário

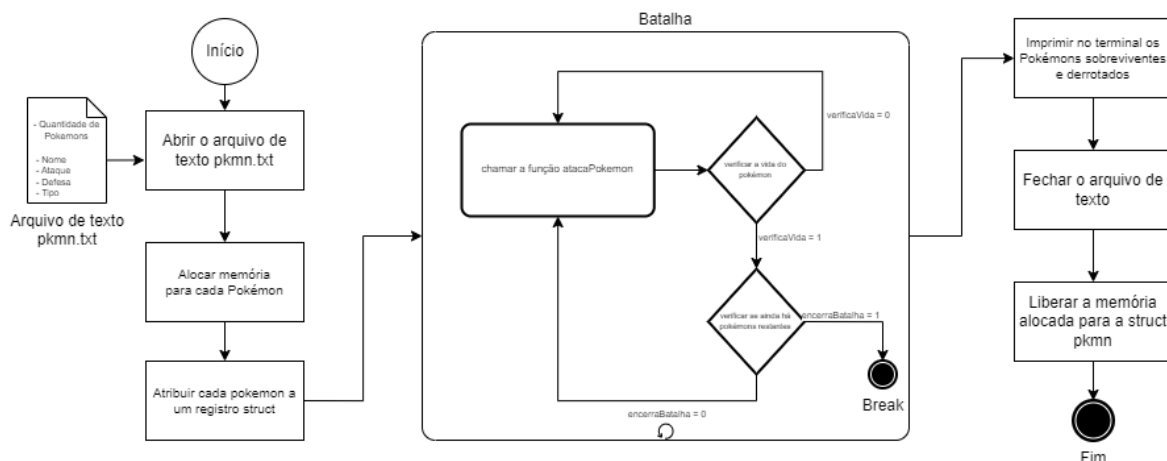
1. Introdução .....	3
2. Descrição do Algoritmo e Procedimentos utilizados .....	3
<b>Fluxograma de Funcionamento do Programa</b> .....	3
Funcionamento do Programa: .....	3
<b>Exemplo de Execução:</b> .....	4
3. Testes e Erros .....	6
<b>Processo de Criação do Algoritmo</b> .....	6
<b>Exemplos de Erros Comuns e Correções</b> .....	7
<b>Testes Realizados</b> .....	8
4. Conclusão .....	9

## 1. Introdução

Com a temática “Pokémon”, o problema a ser resolvido foi a simulação de uma batalha onde a quantidade e os pokémons de cada jogador são identificados de um arquivo de texto. A batalha considera os valores de ataque, defesa e tipagem para definir o pokémon vencedor. O resultado da batalha indica o jogador vencedor e quais são os pokémons sobreviventes e os derrotados.

## 2. Descrição do Algoritmo e Procedimentos utilizados

### Fluxograma de Funcionamento do Programa



### Funcionamento do Programa:

O jogo começa com o número de pokémons dos **jogadores 1 e 2** sendo informados na primeira linha do arquivo de texto. Nas linhas subsequentes, são identificados os pokémons com o **Nome, Ataque, Defesa, Vida e Tipo**.

As  $n$  primeiras linhas correspondem aos pokémons do primeiro jogador e as  $j$  linhas remanescentes, ao segundo jogador.

A leitura é realizada pela função **fscanf** e armazenadas nas variáveis **quantidadePkmnP1**, **quantidadePkmnP2** (a quantidade de pokémons de cada jogador) e **pokemonP1** e **pokemonP2** (ponteiros para a **struct pkmn** onde são registrados os atributos de cada pokémon).

Após a leitura do arquivo de texto, a batalha é iniciada um loop onde, por turno, é chamada a função **atacaPokemon** que, por meio do retorno da função **superEfetivo**, altera o atributo de ataque do pokémon naquela execução da função **atacaPokemon**.

Após isso, a condicional verifica se o retorno da função **verificaVida** indica se a vida do pokémon atacado é igual, ou inferior a zero. Caso positivo, imprime no terminal que o pokémon atacante venceu o pokémon defensor. Ainda nesta condicional, pelo retorno da função **encerraBatalha**, é indicado, com referência ao valor das variáveis **pkmp1** (para o jogador 1) e **pkmnP2** (para o jogador 2), se a quantidade de pokémons iterados no loop é igual a atribuída a quantidade de pokémons do jogador atacado. Em caso positivo desta igualdade, o loop é encerrado pelo comando **break**.

Por fim, é chamada a função **statusVidaPokemons** que recebe a struct dos pokémons e da quantidade de pokémons de cada jogador. Em um loop **for**, verifica se o atributo vida é maior que zero. Se sim, imprime que o pokémon indicado é um sobrevivente. Feito isso, em um novo **for** verifica se a vida é menor, ou igual a zero. Caso positivo, imprime o pokémon como derrotado.

## Exemplo de Execução:

### Input:

1 2

Squirtle 15 25 40 agua

Charmander 25 12 36 fogo

Pikachu 25 15 24 eletrico

### Processo:

1. Leitura do número de Pokémon para cada jogador.
2. Alocação de memória, em uma struct, para cada pokémon.
3. Registro dos atributos de cada pokémon nos parâmetros da struct.
4. Inicia a batalha e aciona a função **atacaPokemon** passando os parâmetros do pokémon atacante e do pokémon defensor. Essa função faz a chamada da função **superEfetivo** e realiza o desconto na vida do pokémon defensor conforme a

comparação entre a pontuação de ataque do atacante e a defesa do defensor. Caso a comparação, realizada na função **superEfetivo**, do tipo do pokémon seja compatível, o valor de ataque do atacante é aumentado em 20%.

5. Compara a vida do pokémon defensor chamando a função **verificaVida**. Caso o retorno seja igual a 0 (zero), retorna ao passo 4, mas invertendo os pokémons atacante e defensor. Se o retorno da **verificaVida** for igual a 1, incrementa o contador de pokémons do jogador defensor.

6. Verifica, pelo retorno da função **encerraBatalha**, se a variável incrementada no passo anterior tem o mesmo valor à quantidade total de pokémons do jogador defensor. Se o retorno é igual a 0 (zero), retorna ao passo 4, mas invertendo os pokémons atacante e defensor. Caso o retorno seja igual a 1, imprime que o jogador atacante é o vencedor e encerra o loop pelo comando **break**.

7. Faz a chamada da função **statusVidaPokemons** para validar a vida de cada pokémon dos jogadores 1 e 2. Imprime no terminal a mensagem "Pokémons sobreviventes:" e inicia a verificação do atributo vida. Caso seja maior que zero, imprime o atributo nome. Após isso, imprime no terminal a mensagem "Pokémons derrotados:" e faz novamente a verificação do atributo vida. Caso seja menor, ou igual a zero, imprime o atributo nome.

#### **Output:**

Squirtle venceu Charmander.

Pikachu venceu Squirtle.

Jogador 2 venceu!

Pokemons sobreviventes:

Pikachu

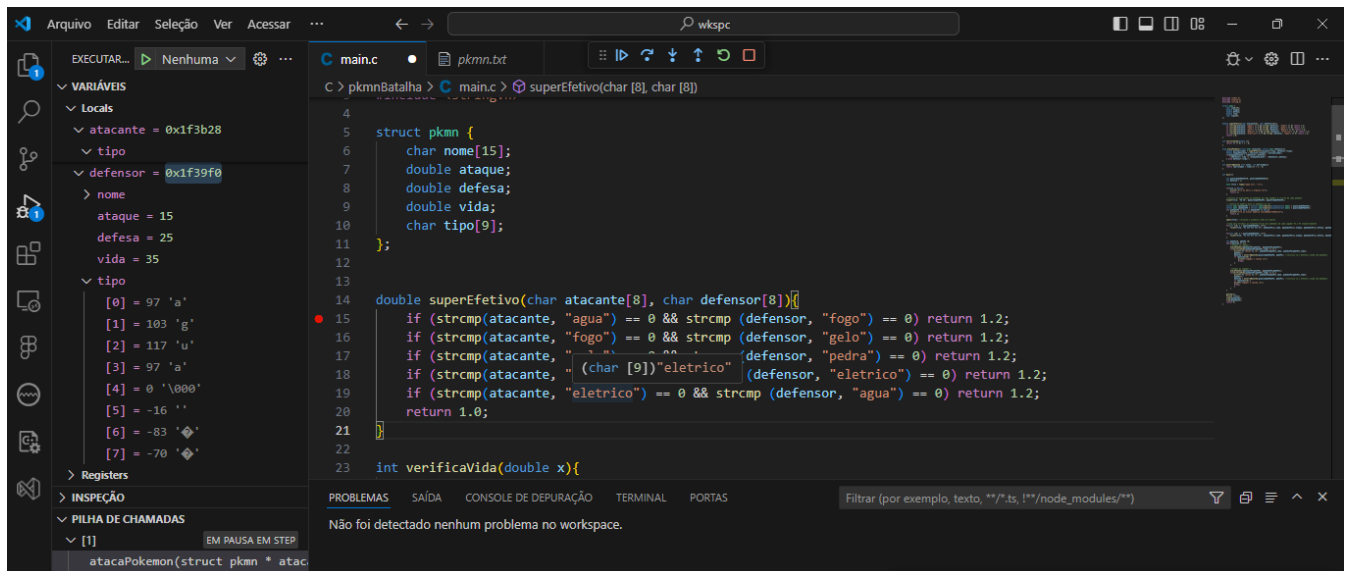
Pokémons derrotados:

Squirtle

Charmander

### 3. Testes e Erros

Nas primeiras tentativas, atribuí o valor máximo do vetor char “tipo” como 8. Nessa situação, ao processar o ataque dos pokémons com tipo elétrico, a validação pela função **superEfetivo** não ocorria e o ataque do pokémon nunca era aumentado. O erro foi tratado depurando o código e identificando a etapa onde a comparação pelo método strcmp(variável, stringProcurada) estava retornando diferente de zero. Corrigi o problema aumentando o vetor para 9 caracteres.



```
4
5 struct pkmn {
6     char nome[15];
7     double ataque;
8     double defesa;
9     double vida;
10    char tipo[9];
11 };
12
13
14 double superEfetivo(char atacante[8], char defensor[8]){
15     if (strcmp(atacante, "agua") == 0 && strcmp(defensor, "fogo") == 0) return 1.2;
16     if (strcmp(atacante, "fogo") == 0 && strcmp(defensor, "gelo") == 0) return 1.2;
17     if (strcmp(atacante, "gelo") == 0 && strcmp(defensor, "pedra") == 0) return 1.2;
18     if (strcmp(atacante, "pedra") == 0 && strcmp(defensor, "eletrico") == 0) return 1.2;
19     if (strcmp(atacante, "eletrico") == 0 && strcmp(defensor, "agua") == 0) return 1.2;
20     return 1.0;
21 }
22
23 int verificaVida(double x){
```

Locals:

- atacante = 0x1f3b28
- tipo
- defensor = 0x1f39f0
- nome
  - ataque = 15
  - defesa = 25
  - vida = 35
- tipo
  - [0] = 97 'a'
  - [1] = 103 'g'
  - [2] = 117 'u'
  - [3] = 97 'a'
  - [4] = 0 '\000'
  - [5] = -16 ''
  - [6] = -83 ''
  - [7] = -70 ''

### Processo de Criação do Algoritmo

Primeiro desafio foi o de escolher a melhor estrutura de dados para armazenar os valores lidos no arquivo de texto. A struct foi escolhida justamente por conseguir agrupar diferentes tipos de dados e dinamizar o processo de alocação de memória.

O segundo foi o de retornar o ataque “Super Efetivo”. Fiz a tentativa de alterar o atributo “tipo” da struct para receber outra Struct contendo os possíveis elementos. Mas pela complexidade em estruturar o código, resolvi utilizar a função strcmp da biblioteca string.h que tornou mais acessível a manipulação do tipo do pokémon.

O terceiro desafio foi o de retornar o dano Super Efetivo. Com o método em double, pelo arredondamento, ao realizar a batalha entre Pikachu (elétrico) e Sandshrew (Pedra) o pokémon Pikachu sobrevive à batalha com 1.8 pontos de vida. Caso o arredondamento para INT seja considerado, o pokémon Sandshrew vence a batalha.

## Exemplos de Erros Comuns e Correções

Erro: Arredondamento do ataque

1. Correção: Pelo arredondamento do ataque, por exemplo, o pokémon Pikachu sobrevive à batalha com 1.8 pontos de vida. Podendo assim finalizar o pokémon Sandshrew mesmo que o pokémon do tipo Pedra tenha vantagens sobre o Elétrico.

## Testes Realizados

Entrada	Saída
<p>3 2</p> <p>Squirtle 10 15 15 agua</p> <p>Vulpix 15 15 15 fogo</p> <p>Onix 5 20 20 pedra</p> <p>Golem 20 5 10 pedra</p> <p>Charmander 20 15 12 fogo</p>	<p>Squirtle venceu Golem</p> <p>Charmander venceu Squirtle</p> <p>Vulpix venceu Charmander</p> <p>Jogador 1 venceu</p> <p>Pokemon sobreviventes:</p> <p>Vulpix</p> <p>Onix</p> <p>Pokemon derrotados:</p> <p>Squirtle Golem Charmander</p>
<p>11 5</p> <p>Pachirisu 15 17 9 eletrico</p> <p>Ampharos 15 14 11 agua</p> <p>Infernape 18 9 23 fogo</p> <p>Spheal 14 18 18 gelo</p> <p>Rampardos 9 18 22 pedra</p> <p>Lotad 14 18 20 agua</p> <p>Magmortar 21 16 24 fogo</p> <p>Sealeo 15 19 28 gelo</p> <p>Suicune 16 15 25 agua</p> <p>Dugtrio 12 26 24 pedra</p> <p>Charmander 18 12 26 fogo</p> <p>Lombre 17 5 39 agua</p> <p>Heatran 19 12 11 fogo</p> <p>Walrein 15 14 20 gelo</p> <p>Bonsly 11 19 17 pedra</p> <p>Pikachu 35 30 24 eletrico</p>	<p>Pachirisu venceu Lombre</p> <p>Heatran venceu Pachirisu</p> <p>Ampharos venceu Heatran</p> <p>Walrein venceu Ampharos</p> <p>Infernape venceu Walrein</p> <p>Bonsly venceu Infernape</p> <p>Spheal venceu Bonsly</p> <p>Pikachu venceu Spheal</p> <p>Pikachu venceu Rampardos</p> <p>Pikachu venceu Lotad</p> <p>Pikachu venceu Magmortar</p> <p>Pikachu venceu Sealeo</p> <p>Pikachu venceu Suicune</p> <p>Pikachu venceu Dugtrio</p> <p>Pikachu venceu Charmander</p> <p>Jogador 2 venceu</p> <p>Pokémons sobreviventes:</p> <p>Pikachu</p> <p>Pokémons derrotados:</p> <p>Pachirisu</p> <p>Ampharos</p> <p>Infernape</p> <p>Spheal</p> <p>Rampardos</p> <p>Lotad</p> <p>Magmaotar</p> <p>Sealeo</p> <p>Suicune</p> <p>Dugtrio</p> <p>Charmander</p> <p>Lombre</p> <p>Heatran</p> <p>Walrein</p> <p>Bonsly</p>



## 4. Conclusão

A temática ajudou no engajamento para realizar o trabalho. Consegui aprender a utilizar diversas ferramentas para desenvolvê-lo, por exemplo o Visual Studio Code (construir o código), Repl.it (construir o código, mas online), Git (versionamento de código) o Draw.io (diagramas). Durante a construção do jogo foi o momento de aplicar os conceitos vistos em aula, principalmente de registros, ponteiros e até mesmo despertando o interesse para aplicação de outras linguagens e Frameworks, como JavaScript, C# (dotNet) e C++. Fiz a publicação do trabalho no repositório do GitHub: <https://github.com/sgtomt/batalhaPkmn>