

# Relatorio de Laboratório de Introdução à Ciência da Computação II – Avaliativo 1

Leonardo Kenzo Tanaka e Pedro Teidi de Sá Yamacita

27 de agosto de 2025

## 1 Análise do Código da IA

```
1 // Funcao para verificar se eh palindromo
2 int eh_palindromo(char *str) {
3     char processada[1000];
4     int j = 0;
5     // Processa a string: considera apenas caracteres alfanumericos
6     for(int i = 0; str[i] != '\0'; i++) {
7         char c = str[i];
8         if(isalnum(c)) {
9             processada[j++] = tolower(c);
10        }
11    }
12    processada[j] = '\0';
13    int len = j;
14    // Strings vazias ou com 1 caractere sao palindromos
15    if(len <= 1) {
16        return 1;
17    }
18    // Verifica se eh palindromo comparando caracteres das extremidades
19    for(int i = 0; i < len / 2; i++) {
20        if(processada[i] != processada[len - 1 - i]) {
21            return 0;
22        }
23    }
24    return 1;
25 }
26
27 int main() {
28     char linha[1000];
29     // Le linha por linha ate EOF
30     while(fgets(linha, sizeof(linha), stdin) != NULL) {
31         // Remove o '\n' do final se existir
32         int len = strlen(linha);
33         if(len > 0 && linha[len-1] == '\n') {
34             linha[len-1] = '\0';
35         }
36         // Verifica se eh palindromo e imprime resultado
37         if(eh_palindromo(linha)) {
38             printf("Sim\n");
39         } else {
40             printf("Nao\n");
41         }
42     }
43     return 0;
44 }
```

### 1.1 Análise da versão IA

O código acima é a forma iterativa para verificação de palíndromos, gerado por inteligência artificial, mais especificamente o Claude Sonnet 4.

É possível notar que esse código utiliza a biblioteca ctype.h para tratar os caracteres de entrada, tornando-os case-insensitive (com a função tolower) e considerando somente alfanuméricos (com a função isalnum).

Além disso, a função (eh palindromo) é iterativa, na qual percorre caractere por caractere do input original, processando-o em uma string melhor formatada e depois percorrendo essa nova string para identificar se é ou não um palíndromo, imprimindo “Sim” se for ou “Não” se não for.

Para ler o input, o código do Claude utilizou while(fgets) para ler linha por linha até acabar as entradas, armazenando em uma string de tamanho fixo de 1000 caracteres. Ele também remove o '\n' que essa string recebe, substituindo-o por '\0', garantindo, assim, uma leitura correta dos valores.

No geral, esse código é funcional e cumpre tarefas simples para verificar se uma frase é um palíndromo. No entanto, caso a entrada supere 1000 caracteres, pode ocorrer um buffer overflow. Ademais, ele utiliza uma quantidade de memória desnecessária ao criar outra string com 1000 espaços para cada nova verificação.

## 2 Análise do Código Iterativo

```
1 #define TAM 100
2 bool Palindromo(char *linha);
3
4 int main()
5 {
6     char linha[TAM];
7     // Enquanto existir linhas para serem lidas vai executar o código abaixo
8     while (fgets(linha, TAM, stdin) != NULL)
9     {
10         if (Palindromo(linha))
11             printf("Sim\n");
12         else
13             printf("N o\n");
14     }
15     return 0;
16 }
17 // Funcao que verifica se eh ou nao palindromo
18 bool Palindromo(char *linha)
19 {
20     char *linhaLimpa = (char *)malloc(sizeof(char) * 1);
21     int tamanhoLinhaLimpa = 0;
22     // Processa a linha bruta em uma linha limpa para melhor analise de palindromo
23     for (int i = 0; linha[i] != '\0'; i++)
24     {
25         char caracter = linha[i];
26         // Filtra somente os caracteres alfanumericos
27         if (!(caracter >= 'A' && caracter <= 'Z') && !(caracter >= 'a' && caracter <=
28             'z')) && !(caracter >= '0' && caracter <= '9'))
29             continue;
30         // Converte todas as letras maiusculas em minusculas
31         if (caracter <= 'Z' && caracter >= 'A')
32             caracter += 32;
33         // Passa todas as letras validas para a linha limpa
34         linhaLimpa = realloc(linhaLimpa, (tamanhoLinhaLimpa + 1) * sizeof(char));
35         linhaLimpa[tamanhoLinhaLimpa] = caracter;
36         tamanhoLinhaLimpa++;
37     }
38     linhaLimpa[tamanhoLinhaLimpa] = '\0';
39     // Verifica se a string eh um palindromo
40     for (int i = 0; i < strlen(linhaLimpa) / 2; i++)
41     {
42         if (linhaLimpa[i] != linhaLimpa[tamanhoLinhaLimpa - 1 - i])
43         {
44             free(linhaLimpa);
45             return false;
46         }
47     }
48     free(linhaLimpa);
49     return true;
50 }
```

## 2.1 Análise do Código Iterativo

O código acima apresenta a forma iterativa para verificação de palíndromos. Como é possível perceber, o código foi piorado intencionalmente utilizando a biblioteca `stdlib.h` para realizar alocação dinâmica ao criar uma linha limpa, além de usarmos `realloc` a cada novo caractere inserido na string.

Além disso, o filtro de caracteres alfanuméricos e o método para tornar o input case-insensitive foram implementados por meio dos intervalos de valores desses caracteres na tabela ASCII, sendo posteriormente passados para a linha limpa.

Dessa forma, conclui-se que o código se torna custoso devido à quantidade de chamadas ao `realloc`, ainda que economize memória ao adotar esse método. Ademais, ele não utiliza as funções já disponíveis na biblioteca `ctype.h`, que realizam a mesma tarefa de filtragem e poderiam melhorar a legibilidade do código.

## 3 Comparação Entre os Dois Códigos

A principal diferença entre os dois códigos está nas bibliotecas utilizadas e na forma de alocação dinâmica das strings. Essas diferenças não interferem significativamente no tempo de execução nem no uso de memória nos casos de teste utilizados no Run Codes, mas podem ser percebidas em situações mais extremas.

O código gerado pela IA utiliza uma quantidade de memória muito maior que o iterativo, enquanto o iterativo consome mais processamento para economizar memória devido ao uso do `realloc`, resultando em uma complexidade de  $O(n^2)$ , ao passo que o da IA apresenta complexidade de  $O(n)$ .

```
1 //IA - Complexidade O(n)
2 #include <ctype.h>
3 tolower();
4 isalnum();
5
6 //Iterativa - Complexidade O(n^2)
7 #include <stdlib.h>
8 malloc();
9 realloc()
```

No entanto, ambos os códigos apresentam um problema caso a entrada seja muito extensa e ultrapasse o tamanho da string inicial do input, o que pode causar um buffer overflow. Além disso, ambos processam a string de entrada em uma nova linha limpa e formatada, com todos os caracteres filtrados e organizados, a fim de realizar a verificação de palíndromos.

```
1 //RISCO DE BUFFER OVERFLOW
2 char linha[tamanho]
3
4 //Usada dentro da funcao palindromo
5 char linhaLimpa[tamanho]
```

Portanto, a versão mais simples e legível, com um código mais enxuto e limpo, é a gerada pela IA, que utiliza a biblioteca `ctype.h` e não realiza realocação dinâmica de forma desnecessária a cada iteração de cada caractere, ainda que consuma uma quantidade maior de memória.

## 4 Recursiva x Iterativa x Iterativa IA

Dentre todos os modelos para realizar a verificação de palíndromos, o mais eficiente é o desenvolvido pela IA, já que sua complexidade é  $O(n)$ , assim como a versão recursiva. No entanto, a solução iterativa da IA utiliza uma única função para verificar o palíndromo, sem a necessidade de múltiplas chamadas recursivas da mesma função, o que evita o overhead adicional no código.