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

```c
// Funcao para verificar se eh palindromo
int eh_palindromo(char *str) {
    char processada[1000];
    int j = 0;
    // Processa a string: considera apenas caracteres alfanumericos
    for(int i = 0; str[i] != '\0'; i++) {
        char c = str[i];
        if(isalnum(c)) {
            processada[j++] = tolower(c);
        }
    }
    processada[j] = '\0';
    int len = j;
    // Strings vazias ou com 1 caractere sao palindromos
    if(len <= 1) {
        return 1;
    }
    // Verifica se eh palindromo comparando caracteres das extremidades
    for(int i = 0; i < len / 2; i++) {
        if(processada[i] != processada[len - 1 - i]) {
            return 0;
        }
    }
    return 1;
}

int main() {
    char linha[1000];
    // Le linha por linha ate EOF
    while(fgets(linha, sizeof(linha), stdin) != NULL) {
        // Remove o '\n' do final se existir
        int len = strlen(linha);
        if(len > 0 && linha[len-1] == '\n') {
            linha[len-1] = '\0';
        }
        // Verifica se eh palindromo e imprime resultado
        if(eh_palindromo(linha)) {
            printf("Sim\n");
        } else {
            printf("Nao\n");
        }
    }
    return 0;
}
```

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 da 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, no qual ele percorre caracter por caracter do input original, processando-o em uma string melhor formatada e depois percorre essa nova string para identificar se é ou não um palíndromo, printando "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 1000 fixo. Ele também remove o '$\backslash n$' que essa string recebe substituindo-a por '\0' , dessa forma garante 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 uma outra string com 1000 espaços para cada nova verificação.

2 Análise do Código Iterativo

```c
#define TAM 100
bool Palindromo(char *linha);

int main()
{
    char linha[TAM];
    // Enquanto existir linhas para serem lidas vai executar o codigo abaixo
    while (fgets(linha, TAM, stdin) != NULL)
    {
        if (Palindromo(linha))
            printf("Sim\n");
        else
            printf("N  o\n");
    }
    return 0;
}
// Funcao que verifica se eh ou nao palindromo
bool Palindromo(char *linha)
{
    char *linhaLimpa = (char *)malloc(sizeof(char) * 1);
    int tamanhoLinhaLimpa = 0;
    // Processa a linha bruta em uma linha limpa para melhor analise de palindromo
    for (int i = 0; linha[i] != '\0'; i++)
    {
        char caracter = linha[i];
        // Filtra somente os caracteres alfanumericos
        if (!(caracter >= 'A' && caracter <= 'Z') && !(caracter >= 'a' && caracter <=
            'z') && !(caracter >= '0' && caracter <= '9'))
            continue;
        // Converte todas as letras maiusculas em minusculas
        if (caracter <= 'Z' && caracter >= 'A')
            caracter += 32;
        // Passa todas as letras validas para a linha limpa
        linhaLimpa = realloc(linhaLimpa, (tamanhoLinhaLimpa + 1) * sizeof(char));
        linhaLimpa[tamanhoLinhaLimpa] = caracter;
        tamanhoLinhaLimpa++;
    }
    linhaLimpa[tamanhoLinhaLimpa] = '\0';
    // Verifica se a string eh um palindromo
    for (int i = 0; i < strlen(linhaLimpa) / 2; i++)
    {
        if (linhaLimpa[i] != linhaLimpa[tamanhoLinhaLimpa - 1 - i])
        {
            free(linhaLimpa);
            return false;
        }
    }
    free(linhaLimpa);
    return true;
}
```

2.1 Análise do Código Iterativo

O código acima é a forma iterativa para verificação de palíndromos. Como é possível perceber, utilizamos a biblioteca stdib.h para realizar alocação dinâmica ao criar uma linha limpa, e também usamos realloc a cada novo caracter que colocamos dentro da string.

Além disso, o filtro de caracteres alfanuméricos e o método de tornar o input case-insensitive foi feito através dos intervalos de valores desses caracteres da tabela ASCII e passado para a linha limpa.

Dessa forma, conclui-se que o código é muito custoso pela quantidade de realloc que utilizamos, apesar de economizarmos no uso de memória ao utilizar esse método. Também não utiliza as funções que já existem da biblioteca ctype.h que realizam a mesma tarefa de filtragem, o que melhoraria a legibilidade do código.

3 Comparação Entre os Dois Códigos

A principal diferença entre os dois códigos são as bibliotecas utilizadas e a alocação dinâmica das strings, essas diferenças não interferem significativamente no tempo de execução de cada código e nem do uso de memória pelos casos teste utilizados no run codes, mas se pode ser percebido com casos mais extremos. O código gerado pela IA usa uma quantidade de memória muito maior que a iterativa, enquanto a iterativa consome muito processamento em troca de economizar memória pelo realloc com uma complexidade de $O(n^2)$, enquanto o da IA é $O(n)$.

```
//IA - Complexidade O(n)
#include <ctype.h>
tolower();
isalnum();

//Iterativa - Complexidade O(n^2)
#include <stdlib.h>
malloc();
realloc()
```

No entanto, ambos os códigos possuem um problema caso a entrada seja muito extenso e ultrapasse o tamanho da string inicial do input, podendo causar um buffer overflow. Além disso, elas também processam a string de entrada para uma nova linha limpa e processada com todos os caracteres filtrados e organizados para realizar a verificação de palíndromos.

```
//RISCO DE BUFFER OVERFLOW
char linha[tamanho]

//Usada dentro da funcao palindromo
char linhaLimpa[tamanho]
```

Então, a versão mais simples e mais legível com um código mais enxuto e limpo é o da IA, que utiliza a biblioteca ctype.h e não fica utilizando realocação dinâmica de forma desencessária a cada iteração de cada caracter, apesar desta gastar uma quantidade maior de memória.