Relatório Projeto Entregável 01 - Palíndromo

João Pedro Neves e Juan Pablo Tomba

29 de agosto de 2025

1 Introdução

Neste relatório, explicaremos o processo de criação de 3 versões de código que visam resolver ao mesmo problema: classificar uma string fornecida pelo usuário como sendo um palíndromo ou não. O intuito da criação de distintas versões, é realizar um comparativo de eficiência entre elas, sendo uma apoiada no método de iteração, uma na recursão, e a última uma versão de iteração criada por IA (inteligência artificial). Aqui, discutiremos sobre tópicos como tempo de execução, manutenibilidade, simplicidade e repertório.

2 Metodologia

O primeiro passo do projeto foi criar um repositório público no GitHub onde ambos os membros do grupo poderiam subir os códigos. Dessa forma, poderíamos desenvolver simultaneamente os diferentes códigos e ao final ter uma forma prática de juntá-los e/ou acessá-los.

https://github.com/joaopneves1570/Projeto-1---Lab-ICC2

Em seguida, criados os arquivos para o projeto, utilizamos o ambiente de desenvolvimento do Visual Studio Code para escrever e compilar os mesmos.

3 Códigos

Após isso, confeccionamos os respectivos códigos:

3.1 Iterativo Manual

```
// funcao que troca letras maiusculas por minusculas e "," por "backspace"
void limpafrase(char frase[], int tam){
    for (int i = 0; i < tam; i++){
        int aux = frase[i];
        if (aux > 64 && aux < 91){
            aux = aux + 32;
            frase[i] = (char)aux;
        }
        if (!((aux >= 48 && aux <= 57) || (aux >= 97 && aux <= 122))) {
            aux = 32;
            frase[i] = (char)aux;
        }
    }
}

// funcao que remove espacos da string
void remove_espaco(char str[]) {
    int i = 0;
    int j = 0;
    while (str[j] != '\0') {
        if (!isspace(str[j])) {
            str[i] = str[j];
            i++;
        }
        j++;
    }
```

```
    str[i] = '\0';
}

// funcao que verifica se e palindromo e printa o resultado
void ehpalindromo(char str[], int tamanho){
    int inicio = 0, fim = tamanho - 1;
    for (int i = 0; i < (tamanho/2); i++){
        if (str[inicio] != str[fim]){
            printf("Nao\n");
            return;
        }
        inicio++;
        fim--;
    }
    printf("Sim\n");
    return;
}
```

3.2 Iterativo por IA

```
// Funcao para verificar se a string e palindromo
int ehPalindromo(const char *str) {
    int i = 0;
    int j = strlen(str) - 1;

    while (i < j) {
        // Ignorar nao alfanumericos
        while (i < j && !isalnum((unsigned char)str[i])) i++;
        while (i < j && !isalnum((unsigned char)str[j])) j--;

        // Comparar em minusculas
        if (tolower((unsigned char)str[i]) != tolower((unsigned char)str[j])) {
            return 0; // Nao e palindromo
        }

        i++;
        j--;
    }

    return 1; // E palindromo
}
```

3.3 Recursivo Manual

```
//Funcao que "limpa a frase", retira caracteres nao alfanumericos.
void limpaFrase(char *frase){
    int j = 0;
    for (int i = 0; frase[i] != '\0'; i++){
        if (frase[i] >= 'A' && frase[i] <= 'Z') frase[i] = frase[i] + 32;
        if ((frase[i] >= 'a' && frase[i] <= 'z') || (frase[i] >= '0' && frase[i]
             <= '9'))
           frase[j++] = frase[i];
    }
    frase[j] = '\0';
}

//Funcao recursiva que verifica se e palindromo ou nao
//A funcao compara o primeiro caractere da string limpa com o ultimo e retorna
    falso caso sejam diferentes
//Caso sejam iguais, ela cria uma nova string sem esses mesmos caracteres e
    chama a funcao de novo para essa nova string
bool ePalindromo(char *frase){
    if (((int)(strlen(frase) - 1 )< 1)){
        return true;
```

```
18      }
19
20      if (frase[0] == frase[strlen(frase)-1]){
21          char *novaFrase = (char*)malloc(sizeof(char)*(strlen(frase)));
22          int j = 0;
23          for (int i = 1; i < strlen(frase) - 1; i++){
24              novaFrase[j] = frase[i];
25              j++;
26          }
27          novaFrase[j] = '\0';
28          strcpy(frase, novaFrase);
29          free(novaFrase); novaFrase = NULL;
30          return ePalindromo(frase);
31      } else {
32          return false;
33      }
34  }
```

4 Resultados

Comparando esses três códigos, obtivemos os seguintes resultados:

4.1 Tempo de Execução

Utilizamos a biblioteca windows.h para medir os tempos de resposta dos códigos, rodamos 15 vezes em uma mesma compilação a string "Socorram-me, subi no ônibus em Marrocos"e tiramos a média simples das medições para cada um dos códigos. Os resultados foram os seguintes:

- **Iterativo manual:** Média ≈ 0.000709s (709 µs por execução)
- **Iterativo por IA:** Média ≈ 0.00000144s (1,44 µs por execução)
- **Recursivo manual:** Média ≈ 0.0000144s (14,4 µs por execução)

4.2 Manutenibilidade

Em relação a manutenibilidade, o código feito pelo chatGPT utiliza funções de bibliotecas próprias da linguagem C, tais quais **isalnum** e **tolower**, além de possuir apenas uma função *int ehPalindromo* para além da main, deixando o código simples, legível, e de fácil manutenção caso as diretrizes mudem futuramente e seja necessário considerar strings com acento por exemplo.

Já o código iterativo feito manualmente, possui também as funções *limpafrase* e *removeespaco* para além da main que podem ser consideradas um pouco redundantes, visto que reimplementam algumas funções existentes em ctype.h e fazem as conversões de **tolower** "na raça", o que pode deixar o código menos coeso e talvez mais suscetível a bugs no futuro, ou caso as diretrizes mudem. A função *void ehpalindromo*, no entanto, é clara em seu funcionamento e lógica.

Por último, o código recursivo manual, possui apenas a função *void limpafrase* que possui uma lógica simples para descartar caracteres não alfanuméricos, mas poderia ser evitada utilizando funções próprias de bibliotecas em C, evitando possíveis bugs. A função *bool ePalindromo*, por outro lado, utiliza de alocação dinâmica (malloc / free) em cada iteração do loop, o que é caro computacionalmente e arriscado em termos de vazamento de memória. Ela modifica a string original no processo através de **strcpy** e utiliza a função **strlen** muitas vezes. Corre o risco de sofrer "stack overflow"para strings muito grandes e a lógica de implementação não é nada simples ou intuitiva.

4.3 Simplicidade

Ao avaliar a simplicidade, buscamos analisar o quão direto e compreensível é o código para alguém que o lê ou realiza a manutenção. Quanto menos funções auxiliares desnecessárias e maior uso de bibliotecas, mais simples tende a ser a implementação. Com isso, podemos deduzir o seguinte sobre cada abordagem:

- **Iterativo manual:** simples, mas dividido em várias funções auxiliares que tornam o código mais "verbal".
- **Iterativo por IA:** mais compacto e direto, usa funções muito específicas da biblioteca, sendo o mais simples de entender.

- **Recursivo manual:** menos simples, pois envolve manipulação de strings e alocação dinâmica.

5 Conclusões

A partir da análise realizada, observamos que cada abordagem apresenta pontos positivos e negativos.

O código **iterativo manual** se mostrou funcional, porém mais verboso e com o maior tempo de execução. Entretanto, é bem modularizado, com diversas funções auxiliares, como *limpafrase* e *remove-espaco*, que já existem em bibliotecas, mas foram implementadas manualmente.

O **iterativo por IA** foi o que apresentou melhor desempenho e maior simplicidade, além de utilizar de forma eficiente as bibliotecas padrão do C, tornando-se a solução mais prática.

Já o **recursivo manual**, embora didático para o estudo de recursão, trouxe maior complexidade na implementação, como o uso de alocação dinâmica, sendo menos indicado para aplicações reais.

Em resumo, a versão gerada por IA se destacou como a mais eficiente e robusta, enquanto as versões manuais tiveram valor principalmente como exercício de aprendizado e fixação do conteúdo para programação.