Relatório Projeto Entregável 03 - Volta USP (Parte 1)

João Pedro Neves e Juan Pablo Tomba

19 de setembro de 2025

1 Introdução

Neste relatório, explicaremos o processo de criação de duas versões de código que visam resolver o mesmo problema: ler uma sequência de participantes de uma competição no CEFER, onde cada linha contem NOME - (usp ou externo), e depois separando em dois vetores(usp e externo), cada um com o número de caracteres de cada nome, separados em sua devida categoria.

O objetivo da criação dessas diferentes versões é realizar um comparativo de eficiência entre elas: uma baseada no método de iteração e outra na recursão.

Serão discutidos aspectos 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/Lab-ICC2/tree/main/aula4

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

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <stdbool.h>

//funcao para verificar se e usp ou externo
bool usp_externo(char str[]) {
    int i = 0;
    while (str[i] != '-' && str[i] != '\0') {
        i++;
    }

    if (str[i + 1] == 'u') {
        return true;
    } else {
        return false;
    }
}

//funcao que retorna o tamanho do nome da pessoa
int tam_nome(char str[]){
    char letra;
    int tam = 0, i = 0;
```

```
    while (str[i] != '-'){
        tam++;
        i++;
    }

    return tam;
}

//funcao para remover espa os
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';
}

int main(){
    char frase[102];
    int usp[11], externo[11], tamanho_nome;
    int i = 0, j = 0, k = 0, l = 0;
    while (fgets(frase, sizeof(frase), stdin) != NULL){
        frase[strcspn(frase, "\n")] = '\0';
        remove_espaco(frase);
        tamanho_nome = tam_nome(frase);
        if (usp_externo(frase)){
            usp[i] = tamanho_nome;
            i++;
        }
        else{
            externo[j] = tamanho_nome;
            j++;
        }
    }

    printf("USP - [");
    for (int k = 0; k < i; k++) {
        if (k > 0) printf(", ");
        printf("%d", usp[k]);
    }
    printf("]\n");

    printf("Externa - [");
    for (int l = 0; l < j; l++) {
        if (l > 0) printf(", ");
        printf("%d", externo[l]);
    }
    printf("]");

    return 0;
}
```

3.2 Recursivo Manual

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int conta_letras(char string[]){
    char letra = string[0];
```

```
    if (letra == '-')
        return 0;

    return (isalnum(string[0])? 1 : 0) + conta_letras(string + 1);


}


int main(){
    char string[50];

    int USP[100];
    int Externa[100];

    int i = 0, j = 0;

    while (fgets(string, 50, stdin) != NULL){
        string[strcspn(string, "\n")] = '\0';
        int tamanho_string = strlen(string) - 1;
        if (string[tamanho_string] == 'a')
            Externa[i++] = conta_letras(string);
        else
            USP[j++] = conta_letras(string);
    }

    printf("USP - [");
    if (j > 0){
        for (int k = 0; k < j - 1; k++) printf("%d, ", USP[k]);
        printf("%d]\n", USP[j - 1]);
    } else {
        printf("]\n");
    }

    printf("Externa - [");
    if (i > 0){
        for (int k = 0; k < i - 1; k++) printf("%d, ", Externa[k]);
        printf("%d]\n", Externa[i - 1]);
    } else {
        printf("]\n");
    }

    return 0;
}
```

4 Resultados

Comparando esses dois códigos, obtivemos os seguintes resultados:

4.1 Tempo de Execução

Para medir os diferentes tempos de execução, fizemos o upload dos dois diferentes códigos no RunCodes e de lá retiramos o tempo de execução fornecido pela própria plataforma. Para cada código, calculamos a média entre os tempos de execução dos 5 testes:

- **Iterativo manual:** Média 5 casos: 0,00112 s
- **Recursivo manual:** Média 5 casos: 0,00112 s

Para os dois códigos, o envio retornou os mesmos tempos de execução. Tentando evitar isso, reenviei cinco vezes cada um dos códigos, e calculei assim a média das 5 médias, para tentar obter alguma informação mais relevante. Com isso, obtive:

- **Iterativo manual:** Média 5 médias: 0,001216 s
- **Recursivo manual:** Média 5 médias: 0,001152 s

4.2 Manutenibilidade

A manutenibilidade está relacionada à clareza do código e à facilidade de adaptá-lo para futuras modificações, como mudança no formato da entrada ou inclusão de novas categorias:

- **Iterativo manual:** apresenta uma lógica com laços de repetição bem definidos. A modularização em funções como:(`usp_externo`, `tam_nome`, `remove_espaco`) melhora a organização e facilita alterações futuras. Caso seja necessário expandir para além de USP e externos, a adaptação seria simples.
- **Recursivo manual:** concentra boa parte da lógica em uma função recursiva (`conta_letras`). Embora elegante, a recursão torna a manutenção menos direta, principalmente para quem não está acostumado com recursão.Caso seja necessário expandir para além de USP e externos, exigiriam maior cuidado para modificação no código.

4.3 Simplicidade

A simplicidade avalia a legibilidade e a facilidade de compreensão:

- **Iterativo manual:** mais simples de compreender, pois segue passo a passo a lógica de percorrer as strings, remover espaços, identificar a categoria e armazenar o tamanho dos nomes. É a versão mais didática. Com algoritmo $O(n)$, com "n"sendo o tamanho da entrada.
- **Recursivo manual:** menos simples, já que envolve chamadas recursivas e exige compreender como o problema é reduzido a casos menores. Apesar disso, sua implementação é compacta. Com algoritmo támbem $O(n)$ no pior caso.

5 Conclusões

Com base na comparação, notamos que cada abordagem tem pontos fortes e fracos:

O código **iterativo manual** é o mais simples e direto. Por outro lado, seu fluxo pode se tornar maior conforme novas funcionalidades sejam adicionadas, com verificações extras.

O **recursivo manual** apresenta uma forma mais elegante de resolver a contagem de caracteres, além de ser compacto. Contudo, sua lógica pode ser menos intuitiva para iniciantes.

Em resumo, a versão iterativa manual se destacou pela clareza e facilidade de manutenção, enquanto a versão recursiva trouxe uma solução mais concisa, ainda que menos simples para quem está começando. Ambos os códigos perfomaram de forma relativamente parecida em relação ao tempo de execução, com uma diferença percentual de 5,41% a favor do recursivo.