Universidade Federal de Ouro Preto - UFOP
Departamento de Ciência da Computação - DECOM

# Relatório atividade 3 - FMT

BCC402 - ALGORITMOS E PROGRAMACAO AVANCADA

Kayo Xavier Nascimento Cavalcante Leite - 21.2.4095

Professor: Rafael Alves Bonfim

Ouro Preto
31 de março de 2025

# Sumário



# Lista de Códigos Fonte

## 1 Código e enunciado.

Na Atividade 3 - o problema selecionado foi o FMT. O problema tem como objetivo formatar um texto de entrada em linhas com no máximo 72 caracteres, seguindo regras específicas. O código comentado e documentado, casos de teste e executável pré compilado se encontram no .zip da atividade. O código foi feito com base na referência encontrada no site:

`https://codingstrife.wordpress.com/2013/07/23/solution-uva848-pc110308-fmt/`

Caso queira, para rodar e compilar o código, é necessário ter o compiler g++ e utilizar o seguinte comando no terminal dentro do diretório da pasta da atividade específica:

Compilando e rodando o exercício

```
para compilar:
g++ fmt.cpp -o executavel

e para rodar basta utilizar .\executavel no cmd.

para utilizar os cenários de teste:
.\executavel < input.txt
```

## 2 Pseudocódigo e descrição do problema

# 3 Problema FMT (Formatação de Texto)

**Objetivo**: Formatar um texto de entrada em linhas de até **72 caracteres**, seguindo regras específicas de quebra e espaçamento.

### 3.1 Regras de Formatação

- **Quebras de linha** apenas em espaços, sem espaços no início/fim de linhas.
- **Quebras do input** podem ser substituídas por espaços se não seguidas por outra quebra ou espaço.
- **Palavras longas** (mais que72 caracteres sem espaços) são mantidas em linhas isoladas.
- **Espaços extras** são removidos das extremidades das linhas.

### 3.2 Entrada e Saída

- **Entrada**: Texto com múltiplas linhas (terminado por linha iniciando com '*').
- **Saída**: Texto formatado com linhas de $\leq$72 caracteres.

### 3.3 Estratégia de Solução

**Componentes-Chave**:

- `trimBack/trimFront`: Remove espaços do início/fim de strings.
- `buffer`: Armazena palavras que excedem o limite da linha atual.
- `output`: Acumula o texto formatado final.

**Passos**:

1. **Processamento Iterativo**:

- Leia linhas até encontrar o marcador '*' ou EOF.
- Concatene o conteúdo do `buffer` com a nova linha.

2. **Quebra de Linha**:
   - Se a linha exceder 72 caracteres:
     (a) Procure o último espaço $\leq$72 para quebrar.
     (b) Se não houver espaço, mantenha a palavra inteira (mesmo ¿72).
     (c) Atualize `buffer` com o texto excedente.
3. **Gerenciamento de Espaços**:
   - Remova espaços extras após quebras (`trimBack`).
   - Evite espaços no início de novas linhas (`trimFront`).
4. **Finalização**:
   - Adicione o conteúdo residual do `buffer` ao `output`.
   - Remova quebras de linha extras no final.

## 3.4 Detalhes de Implementação

- **Buffer Dinâmico**:
  - Armazena palavras que não couberam na linha atual para processamento na próxima iteração.
  - Exemplo: `line = "part1 part2"; buffer = "part2"` após quebra em "part1".
- **Lógica de Quebra**:

$$\text{Posição de Quebra} = \begin{cases} \text{Último espaço} \leq 72, & \text{se existir} \\ \text{Tamanho total}, & \text{para palavras longas} \end{cases}$$

- **Otimizações**:
  - Uso de `cin.peek()` para evitar quebras prematuras.
  - Flags (`LineCreated`) para controle de fluxo entre iterações.

```
// Objetivo: Ler texto, formatalo em linhas de no maximo 72 caracteres, quebrando em espacos.

// Funcao trimBack(str): Remove espacos do final de str.
// Funcao trimFront(str): Remove espacos do inicio de str.

// Procedimento main():
// 1. Inicializar output = "", buffer = "", linha_atual = "", foi_quebrada = false.
// 2. Loop Infinito:
// 3.   linha_lida = ""
// 4.   Se foi_quebrada for false:
// 5.     Tentar ler linha_lida da entrada. Se falhar (EOF), sair do loop.
// 6.   Senao (foi_quebrada e true):
// 7.     foi_quebrada = false. // Usa o resto que ja esta em linha_atual
// 8.   Se linha_lida (ou linha_atual se foi quebrada) comecar com '*', sair do loop.
// 9.   parte_nova = linha_lida (ou string vazia se foi_quebrada).
// 10.  trimBack(parte_nova).
// 11.  linha_atual = buffer + parte_nova. // Monta a linha a processar
// 12.  buffer = "". // Esvazia o buffer temporariamente
```

```
// 13.  Se comprimento(linha_atual) > 72:
// 14.    Encontrar pos_quebra = indice do ultimo espaco em linha_atual tal que pos_quebra <= 72.
// 15.    Se nao encontrado, encontrar pos_quebra = indice do ultimo espaco em linha_atual.
// 16.    Se pos_quebra foi encontrado:
// 17.      buffer = substring de linha_atual apos pos_quebra.
// 18.      linha_atual = substring de linha_atual antes de pos_quebra.
// 19.      foi_quebrada = true.
// 20.      trimBack(linha_atual).
// 21.      trimFront(buffer).
// 22.    // Se nao achou espaco, linha_atual fica como esta (pode ser > 72), buffer fica vazio.
// 23.  Senao se comprimento(linha_atual) < 72:
// 24.    Ver proximo caractere da entrada (peek).
// 25.    Se linha_atual nao vazia E nao termina com espaco E peek nao e espaco/newline/EOF:
// 26.      buffer = linha_atual + " ". // Guarda para juntar com proxima linha
// 27.      linha_atual = "".           // Linha atual nao sera impressa agora
// 28.      Continuar para proxima iteracao do loop.
// 29.    // Senao (linha termina aqui), buffer continua vazio.
// 30.  // Se comprimento == 72, buffer continua vazio.
// 31.  Se linha_atual nao esta vazia:
// 32.    Adicionar linha_atual + "\n" ao output.
// 33. Fim do Loop.
// 34. Se buffer nao esta vazio: // Adiciona resto final
// 35.   trimBack(buffer).
// 36.   Adicionar buffer + "\n" ao output.
// 37. Se output nao vazio e termina com '\n', remover ultimo caractere.
// 38. Imprimir output.
```

Código 1: Pseudocódigo do problema.

## 4 Casos teste - Input e output esperado.

Para os casos de teste do problema, foi disponibilizado junto a pasta do mesmo os seguintes arquivos :input.txt, sendo o próprio caso de teste disponibilizado pelo exercício. Além disso, encontra-se também o arquivo com os outputs esperados para cada input. Ambos os resultados foram validados e tiveram o output esperado.

input.txt

```
The unix fmt program reads lines of text, combining
and breaking lines so as to create an
output file with lines as close to without exceeding
72 characters long as possible. The rules for combining and breaking
lines are as follows.
1. A new line may be started anywhere there is a space in the input.
If a new line is started, there will be no trailing blanks at the
end of the previous line or at the beginning of the new line.
2. A line break in the input may be eliminated in the output, provided
it is not followed by a space or another line break. If a line
break is eliminated, it is replaced by a space.
```

output esperado

```
The unix fmt program reads lines of text, combining and breaking lines
so as to create an output file with lines as close to without exceeding
72 characters long as possible. The rules for combining and breaking
lines are as follows. 1. A new line may be started anywhere there is a
space in the input. If a new line is started, there will be no trailing
blanks at the end of the previous line or at the beginning of the new
line. 2. A line break in the input may be eliminated in the output,
provided it is not followed by a space or another line break. If a line
break is eliminated, it is replaced by a space.
```