

# **Relatório**

**Mateus Barbosa e Matheus de Oliveira Rocha**

Universidade do Vale do Itajaí - UNIVALI

Escola do Mar, Ciência e Tecnologia

Ciência da Computação

`{mateus.barbosa, matheus.rocha}@edu.univali.br`

## **Estrutura De Dados**

Trabalho de programação - B

**Marcos Cesar Cardoso Carrard**

**13/04/2023**

## 1. Introdução

Esse é um programa em C/C++ que trabalha com polinômios, usando a estrutura de listas duplamente encadeadas (LDE). Ele contém funções básicas para a manipulação de polinômios, como inicialização da lista, inserção de termos, remoção de termos, busca por termos e impressão da lista.

O programa utiliza uma estrutura de dados para representar cada termo do polinômio, contendo o coeficiente, a letra da variável e o expoente. A inserção na lista é feita de forma ordenada pelo expoente do termo. Além disso, a impressão do polinômio é feita em ordem decrescente de expoente, e os termos com coeficiente 1 são exibidos sem o coeficiente e os termos com coeficiente -1 são exibidos com um sinal negativo.

E temos uma UI interativa que permite ao usuário digitar polinômios matemáticos de maneira mais intuitiva e realizar operações sobre os tais polinômios também.

## 2. Implementação das funções

Esse código em C++ implementa uma Lista Duplamente Encadeada (LDE) que representa um polinômio matemático. Cada nó da lista possui um coeficiente (float), uma letra (char) que representa a variável do polinômio, um expoente (int) e dois ponteiros, um que aponta para o próximo nó (eloP) e outro que aponta para o nó anterior (eloA). Devido a grande necessidade de inserção de nosso programa, foi escolhido a LDE, ao invés de uma Lista Unicamente Encadeada (LUE), já que ela possibilita melhor performance em momento de inserir elementos na lista e retirar esses elementos.

Para o correto funcionamento do programa, foi importado 3 bibliotecas. A biblioteca iostream para a visualização e inserção de valores no console/terminal; a string para receber uma string através de um input do terminal e evitar a utilização do char[] no cin que impossibilitava o uso de espaços; e por último o uso da biblioteca regex que permitiu melhor controle no input do usuário e filtro para as informações necessárias para o programa.

O programa fornece funções básicas que permitem inicializar a lista, inserir um termo no final da lista e inserir um termo ordenado pelo expoente do polinômio, remover um termo da lista, determinar se um termo existe ou não na lista, e imprimir a lista.

Na função `inserir_ordenado()`, caso a lista esteja vazia, o termo é adicionado ao início da lista. Caso contrário, o termo é inserido no início, fim ou meio da lista, conforme a ordem dos expoentes do polinômio.

A função `buscar_por_expoente()` recebe um expoente e percorre a lista em busca do termo correspondente a esse expoente. Se encontrar o termo, retorna o ponteiro para o nó correspondente. Caso contrário, retorna nulo (nullptr).

A função `retirar_por_expoente()` recebe um expoente e retira da lista o termo correspondente a esse expoente. Essa função faz uso da função `buscar_por_expoente()` para encontrar o termo a ser removido. Se o termo não for encontrado, retorna falso. Se for encontrado, verifica se o termo a ser retirado é o primeiro, último ou do meio da lista, e ajusta os ponteiros dos nós correspondentes.

As demais funções são autoexplicativas: a função `inicializar()` atribui os valores *nullptr* aos ponteiros início e fim da lista; a função `mostrar_lista()` percorre a lista e imprime na tela os termos do polinômio; e a função `inserir_final()` inserem um termo no final da lista.

Com isso, terminamos as funções básicas para que a LDE funcione. Mas existem mais funções que fazem os cálculos de polinômios sobre a LDE, são elas: `somar_monomios()`, `somar_polinomios()`, `subtrair_polinomios()`, `multiplicacao_escalar()`, `multiplicacao_polinomios()` e `valor_numerico()`, que serão explicadas abaixo.

A função `somar_monomios()` recebe uma LDE que representa um polinômio e tem como objetivo agrupar monômios de mesmo expoente e somar seus coeficientes. Para isso, percorre a lista encadeada, analisando cada monômio e comparando seu expoente com o do próximo monômio. Caso esses expoentes sejam iguais, soma os coeficientes e avança para o próximo monômio com o mesmo expoente. Caso contrário, insere o monômio agrupado na LDE `polinomio_soma`, que será o resultado da soma dos monômios. Por fim, retorna a LDE `polinomio_soma`.

A função `somar_polinomios()` recebe duas LDEs que representam polinômios e tem como objetivo somá-las. Para isso, percorre as duas listas encadeadas, inserindo cada monômio na LDE `polinomio_resultado`. Após ter inserido todos os monômios das duas LDEs, chama a função `somar_monomios` para agrupar monômios de mesmo expoente e somar seus coeficientes. Por fim, retorna a LDE `polinomio_resultado`, que é o resultado da soma dos dois polinômios.

A função `subtrair_polinomios()` também recebe duas LDEs que representam polinômios e tem como objetivo subtrair o segundo polinômio do primeiro. Para isso, percorre as duas listas encadeadas, inserindo cada monômio da primeira LDE na LDE `polinomio_resultado` e cada monômio da segunda LDE com o coeficiente negativo, fazendo o “jogo do sinal” com os monômios da segunda LDE. Após ter inserido todos os monômios das duas LDEs, chama a função `somar_monomios` para agrupar monômios de mesmo expoente e somar seus coeficientes. Por fim, retorna a LDE `polinomio_resultado`, que é o resultado da subtração dos dois polinômios.

A função `multiplicacao_escalar()` recebe uma LDE que representa um polinômio e um escalar  $k$  e tem como objetivo multiplicar cada coeficiente do polinômio pelo escalar  $k$ . Para isso, percorre a lista encadeada, multiplicando cada coeficiente por  $k$  e inserindo o monômio resultante na LDE `polinomio_resultado`. Após ter inserido todos os monômios da LDE, chama a função `somar_monomios` para agrupar monômios de mesmo expoente e somar seus coeficientes. Por fim, retorna a LDE `polinomio_resultado`, que é o resultado da multiplicação do polinômio pelo escalar  $k$ .

A função `multiplicacao_polinomios()` recebe duas LDEs que representam polinômios e tem como objetivo multiplicá-las. A função retorna uma nova lista duplamente encadeada que representa o resultado da multiplicação dos polinômios. A primeira parte da função é a inicialização da lista resultante. Em seguida, é feita uma verificação para ver se algum dos polinômios de entrada está vazio. Se algum dos polinômios for vazio, a função retorna a lista resultante vazia. A partir daí, o algoritmo

passa por cada elemento de cada um dos polinômios de entrada, e realiza a multiplicação dos coeficientes e somas dos expoentes. A função `inserir_ordenado()` é chamada para inserir cada novo termo calculado na lista resultante, garantindo que a lista permaneça ordenada por ordem decrescente dos expoentes. Depois que todos os termos foram adicionados à lista resultante, é chamada uma função auxiliar chamada `somar_monomios()`, que soma termos semelhantes para simplificar a lista resultante, ou seja, termos com o mesmo expoente são somados. Por fim, a função retorna a lista resultante final, que contém os termos do polinômio resultante da multiplicação dos dois polinômios de entrada.

A função `valor_numerico()` recebe como parâmetros uma lista duplamente encadeada LDE contendo o polinômio e um valor real `valor_real` que representa o ponto em que se deseja avaliar o polinômio. Em seguida, a função percorre a lista encadeada com um ponteiro `aux_polinomio` que aponta para o início da lista. Para cada nó da lista, a função verifica se o coeficiente é diferente de zero e se o expoente é maior do que zero. Se essas condições forem satisfeitas, a função calcula o resultado exponencial elevando o `valor_real` à potência do expoente do nó. Por fim, a função multiplica o resultado exponencial pelo coeficiente do nó e adiciona o resultado ao `resultado_final`. Ao final do laço, a função retorna o `resultado_final`, que é o valor numérico do polinômio no ponto `valor_real`.

Note que o código faz uso de diretivas *#pragma region* para organizar e facilitar a visualização do código, mas essas diretivas não afetam a execução do programa.

Além disso, outro ponto de observação é a utilização do `nullptr`, ao invés do `NULL`. O uso de `nullptr` é preferível ao uso de `NULL` por diversas razões. `nullptr` é um tipo, mais seguro, mais expressivo e mais consistente em diferentes sistemas operacionais. Isso pode tornar o código mais legível e menos suscetível a erros em tempo de execução. No entanto, `NULL` ainda é amplamente utilizado em código legado e bibliotecas, e o seu uso não é considerado inseguro ou obsoleto. Claro que com isso, nosso programa se limita a ser usado na versão C++ 11 em diante.

### 3. Análise básica de performance

Para essa análise dos algoritmos, usamos a notação Big O, que nos ajudará a escrever de forma matemática a complexidade e performance das funções:

- 1) **inicializar:** a função percorre toda a lista encadeada, atribuindo `nullptr` a todos os ponteiros de nós. Isso resulta em um tempo de execução linear em relação ao tamanho da lista. Portanto, a complexidade de tempo da função é  $O(n)$ , onde  $n$  é o número de nós na lista.
- 2) **mostrar\_lista:** a função percorre toda a lista encadeada e imprime o coeficiente, a variável e o expoente de cada nó. Portanto, a complexidade de tempo da função é  $O(n)$ , onde  $n$  é o número de nós na lista.
- 3) **inserir\_ordenado:** a função insere um novo nó na lista encadeada em ordem crescente de expoente. Isso envolve a busca pelo ponto de inserção e a manipulação dos ponteiros dos nós adjacentes. A complexidade de tempo da função depende do tamanho da lista encadeada e da posição de inserção do novo nó. No pior caso, em que o novo nó é inserido no final da lista, a complexidade de tempo é  $O(n)$ , onde  $n$  é o número de nós

na lista. No melhor caso, em que o novo nó é inserido no início da lista, a complexidade de tempo é  $O(1)$ .

4) **inserir\_final:** a função insere um novo nó no final da lista encadeada. Isso envolve a busca pelo último nó da lista e a criação do novo nó com o ponteiro de eloP apontando para nullptr. A complexidade de tempo da função é  $O(n)$ , onde  $n$  é o número de nós na lista.

5) **buscar\_por\_expoente:** a função busca um nó na lista encadeada com um determinado expoente. Isso envolve a busca linear na lista até que o nó com o expoente desejado seja encontrado ou o final da lista seja alcançado. A complexidade de tempo da função é  $O(n)$ , onde  $n$  é o número de nós na lista.

6) **retirar\_por\_expoente:** a função remove um nó da lista encadeada com um determinado expoente. Isso envolve a busca linear na lista até que o nó com o expoente desejado seja encontrado, a manipulação dos ponteiros dos nós adjacentes e a exclusão do nó. A complexidade de tempo da função depende do tamanho da lista encadeada e da posição do nó a ser removido. No pior caso, em que o nó a ser removido está no final da lista, a complexidade de tempo é  $O(n)$ , onde  $n$  é o número de nós na lista. No melhor caso, em que o nó a ser removido está no início da lista, a complexidade de tempo é  $O(1)$ .

7) **somar\_monomios:** a função recebe uma lista encadeada de monômios e retorna uma lista encadeada de monômios com termos semelhantes combinados. Isso envolve a iteração sobre todos os nós da lista e a combinação de termos semelhantes. A complexidade de tempo da função depende do número de nós na lista e do número de termos semelhantes. No pior caso, em que todos os termos são diferentes, a complexidade de tempo é  $O(n^2)$ , onde  $n$  é o número de nós na lista.

8) **somar\_polinomios:** a eficiência dessa função é  $O(n \log n)$ , onde  $n$  é o número de monômios nos polinômios de entrada. Isso ocorre porque a função utiliza a função `inserir_ordenado()` para adicionar os monômios aos polinômios de resultado, e essa função tem complexidade  $O(\log n)$  devido à inserção ordenada. Como o número de monômios somados é  $n$ , a complexidade final é  $O(n \log n)$ .

9) **subtrair\_polinomios:** a eficiência dessa função é  $O(n \log n)$ , onde  $n$  é o número de monômios nos polinômios de entrada. A função usa a função `inserir_ordenado()` para adicionar os monômios aos polinômios de resultado, que tem complexidade  $O(\log n)$  devido à inserção ordenada. Como o número de monômios subtraídos é  $n$ , a complexidade final é  $O(n \log n)$ .

10) **multiplicacao\_escalar:** a eficiência dessa função é  $O(n)$ , onde  $n$  é o número de monômios no polinômio de entrada. A função percorre a lista de monômios e multiplica cada coeficiente pelo escalar. Como o tempo de execução da multiplicação é constante e o número de monômios é  $n$ , a complexidade final é  $O(n)$ .

11) **multiplicacao\_polinomios:** a eficiência dessa função é  $O(n^2)$ , onde  $n$  é o número de monômios nos polinômios de entrada. A função usa dois loops aninhados para multiplicar cada monômio do primeiro polinômio com cada monômio do segundo polinômio. Como há  $n^2$  multiplicação de monômios, a complexidade final é  $O(n^2)$ .

12) **valor\_numerico:** a eficiência dessa função é  $O(n)$ , onde  $n$  é o número de monômios no polinômio de entrada. A função percorre a lista de monômios e calcula o valor do polinômio para o valor  $x$ . Como o tempo de execução da multiplicação e soma é constante e o número de monômios é  $n$ , a complexidade final é  $O(n)$ .

## 4. Programa Final

O programa final inclui um menu para selecionar as operações a serem realizadas, e uma forma mais intuitiva de informar os polinômios. Para fazer isso, temos 2 funções extras que formatam o polinômio digitado pelo usuário, a `remove_espacos()` e `solicita_input_polinomio()`.

A função `remove_espacos` recebe uma string `str` como entrada e remove todos os espaços em branco dela, retornando a string resultante sem espaços. Essa função é usada na função `solicita_input_polinomio` para remover espaços da entrada do usuário antes de processá-la, evitando assim a necessidade de em outras funções ter que validar o espaço em branco.

A função `solicita_input_polinomio` solicita ao usuário que insira um polinômio em formato de string, por exemplo, `"2x^3 + 4x^1 - 1.5"`. Em seguida, ela utiliza uma expressão regular para extrair os coeficientes, letras, expoentes e constantes do polinômio e insere cada termo na lista duplamente encadeada com o nome de parametro `polinomio`. A expressão regular (Regex) utilizada na função é a seguinte: `(-?\d*(\.\d+)?)?([a-zA-Z])\^\?(-?\d+)?|(-?\d+(\.\d+)?)`. Essa expressão regular (Regex) é capaz de identificar termos como `"2x^3"`, `"4x^1"`, `"-1.5"` e `"3"` em uma string. Para cada termo identificado na string, a função `solicita_input_polinomio` insere o coeficiente, a letra, o expoente ou a constante correspondente na lista `polinomio` usando a função `inserir_ordenado`. A função repete esse processo até que todos os termos da string tenham sido identificados e inseridos na lista `polinomio`.

Dentro da `main`, temos um menu interativo para realizar operações explicadas anteriormen. A variável `menu` armazena uma string que contém a apresentação do menu de opções para o usuário. Em seguida, é definida uma variável booleana `executando_aplicacao` como verdadeira, que é utilizada para controlar o loop `while` que executa o menu.

Dentro do loop, o menu é apresentado com `cout << menu`, e o usuário é solicitado a escolher uma das seis opções, que é lida com `cin >> input_usuario`. Depois, o programa verifica qual opção foi escolhida com um bloco `if-else`. Para cada opção, uma determinada função é chamada passando as variáveis necessárias como parâmetros, e o resultado é exibido na tela utilizando a função `mostrar_lista`. Antes de continuar, o usuário é solicitado a pressionar Enter para continuar com a execução do programa. Se o usuário escolher a opção 6, a variável `executando_aplicacao` é definida como falsa, e o programa é encerrado com um comando `return 0`.

O comando `system("cls || clear")` é usado para limpar a tela do console/terminal, independentemente do sistema operacional.

### 4.1 Executando o programa

Após executar o programa compilado, você se encontrara no menu principal do programa, em que poderá selecionar as operações usando os números do intervalo 1 ao 6, sendo que a opção 6 sai do programa. Por simplicidade, nos exemplos abaixo iremos utilizar polinômios de 3 monômios. Para que o programa funcione corretamente, deve-se informar o grau de todos os monômios mesmo se forem igual a 1 ( $x^2$ ,  $x^1$ , etc), apenas não precisa informar para as constantes (1, 5, 65, etc).

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
//
// HP12C-no-rpn
//
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Somar Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
//
// Qual operacao deseja fazer? █
```

Imagem do menu principal do programa

Suponhamos que queira descobrir o valor numérico de um polinômio. Primeiramente digite que a operação é 1, depois informe o polinômio, como por exemplo:  $2x^2 - 2x^1 + 5$ , após isso informe o valor do coeficiente (x) é igual a 2, o resultado final deve ser 9, pois:

$$f(x) = 2x^2 - 2x^1 + 5$$

$$f(2) = 2*2^2 - 2*2^1 + 5 = 2*4 - 4 + 5 = 8 - 4 + 5 = 9$$

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
//
// HP12C-no-rpn
//
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Somar Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
//
// Qual operacao deseja fazer? 1
// Digite um polinomio (Ex: 4x^2 + x^1 - 5): 2x^2 - 2x^1 + 5
// Digite o valor do coeficiente: 2
//
// Resultado: 9
// Precione Enter tecla para continuar...
//
//
```

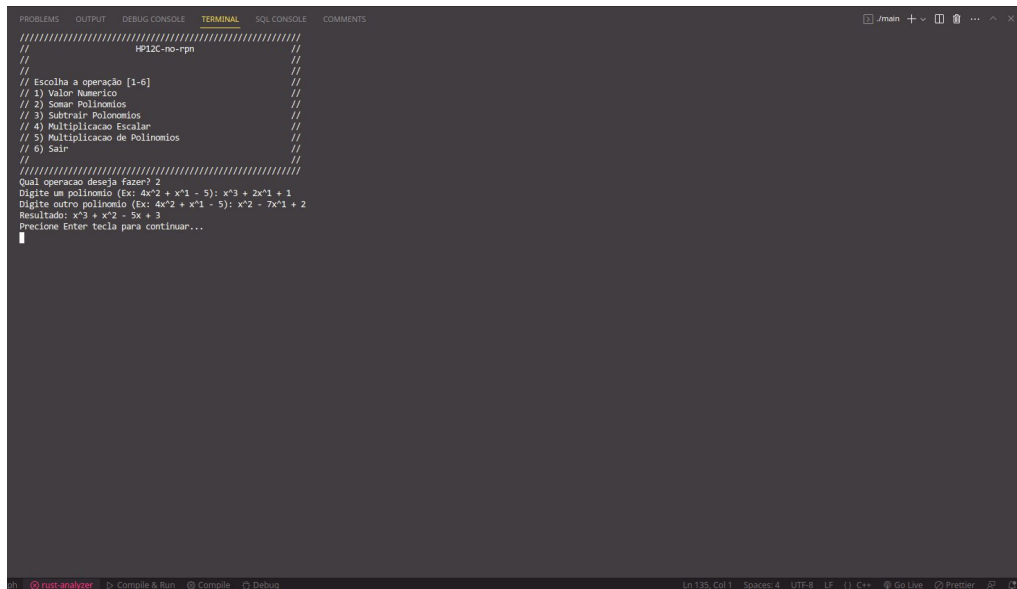
Resultado da operação 1) Valor Numérico

Para a segunda e terceira operação, soma e subtração respectivamente, iremos utilizar os mesmos polinômios, sendo eles:  $x^3 + 2x^1 + 1$  e  $x^2 - 7x^1 + 2$ .

A soma funciona da seguinte maneira, será solicitado para que o usuário informe o primeiro polinômio e após isso digite o segundo polinômio, eles não precisam ser de mesmo tamanho ou possuir os mesmos graus. Depois disso junta-se os monômios de mesmo grau:

$$f(x) = (x^3 + 2x + 1) + (x^2 - 7x + 2)$$

$$f(x) = x^3 + x^2 - 5x + 3$$



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
//
// HP12C-no-rpn
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Soma Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
// Qual operacao deseja fazer? 2
// Digite um polinomio (Ex: 4x^2 + x^1 - 5): x^3 + 2x^1 + 1
// Digite outro polinomio (Ex: 4x^2 + x^1 - 5): x^2 - 7x^1 + 2
// Resultado: x^3 + x^2 - 5x + 3
// Precione Enter tecla para continuar...
```

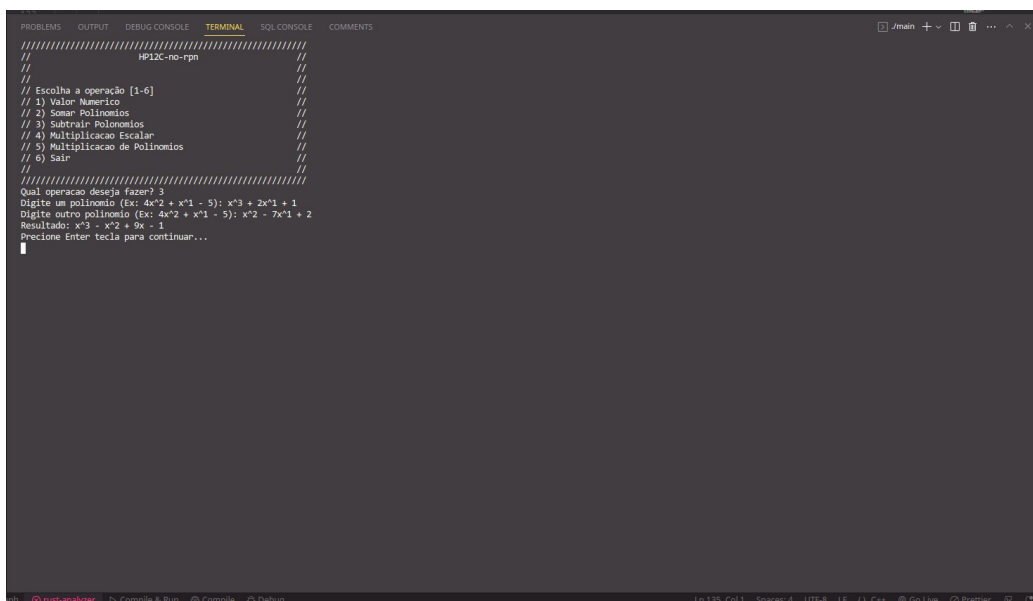
Resultado da operação 2) Soma de Polinômios

A subtração funciona igual a soma, mas invertendo os sinais do segundo polinômio:

$$f(x) = (x^3 + 2x + 1) - (x^2 - 7x + 2)$$

$$f(x) = x^3 + 2x + 1 - x^2 + 7x - 2$$

$$f(x) = x^3 - x^2 + 9x - 1$$



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
//
// HP12C-no-rpn
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Soma Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
// Qual operacao deseja fazer? 3
// Digite um polinomio (Ex: 4x^2 + x^1 - 5): x^3 + 2x^1 + 1
// Digite outro polinomio (Ex: 4x^2 + x^1 - 5): x^2 - 7x^1 + 2
// Resultado: x^3 - x^2 + 9x - 1
// Precione Enter tecla para continuar...
```

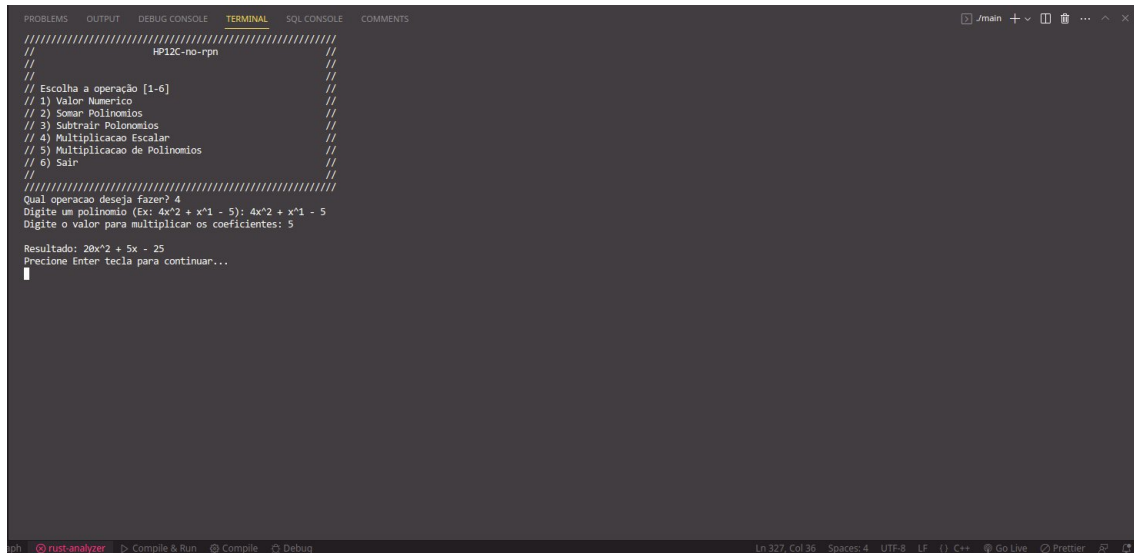
Resultado da operação 3) Subtração de Polinômios



A multiplicação escalar é simples, ela vai simplesmente multiplicar os coeficientes e constantes por um número informado, utilizaremos o seguinte polinomio:  $4x^2 + x^1 - 5$  e o seguinte valor escalar = 5.

$$f(x) = 5*(4x^2 + x^1 - 5)$$

$$f(x) = 20x^2 + 5x^1 - 25$$



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
/////////////////////////////////////////////////////////////////
//
// HP12C-no-rpn
//
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Somar Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
/////////////////////////////////////////////////////////////////
Qual operacao deseja fazer? 4
Digite um polinomio (Ex: 4x^2 + x^1 - 5): 4x^2 + x^1 - 5
Digite o valor para multiplicar os coeficientes: 5

Resultado: 20x^2 + 5x - 25
Precione Enter tecla para continuar...
|
```

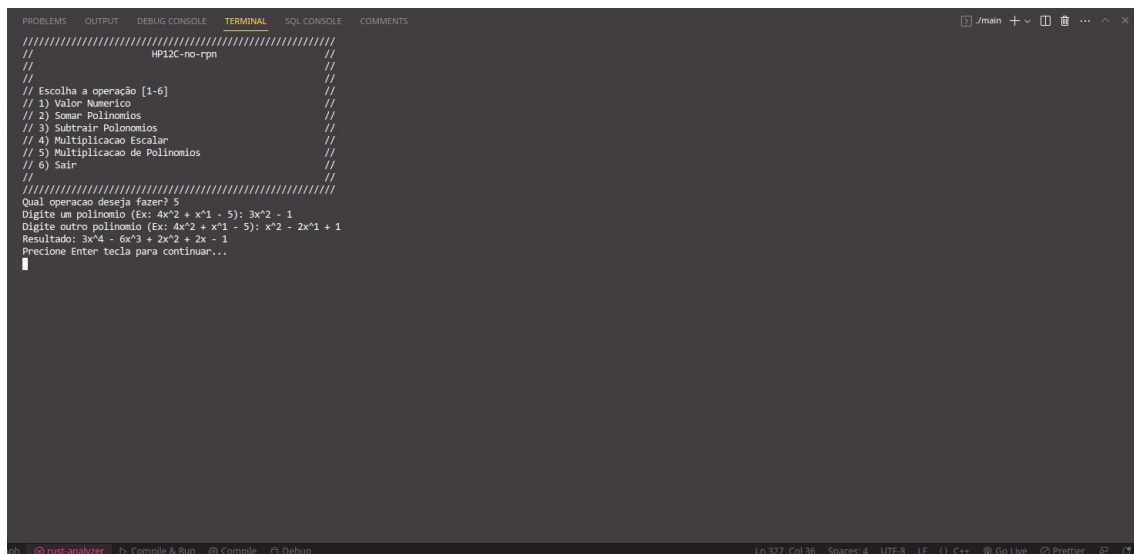
Resultado da operação 4) Multiplicação Escalar

A multiplicação de polinômios irá solicitar que seja digitado 2 polinômios, um de cada vez, nesse exemplo iremos usar os seguintes:  $3x^2 - 1$  e  $x^2 - 2x^1 + 1$ . Para multiplica-lós será feita a distributiva (conhecido também como “chuveirinho”) de cada elemento nos polinômios, e irá multiplicar os coeficiente e somar os expoentes.

$$f(x) = (3x^2 - 1)*(x^2 - 2x^1 + 1)$$

$$f(x) = (3x^4 - 6x^3 + 3x^2 - x^2 + 2x^1 - 1)$$

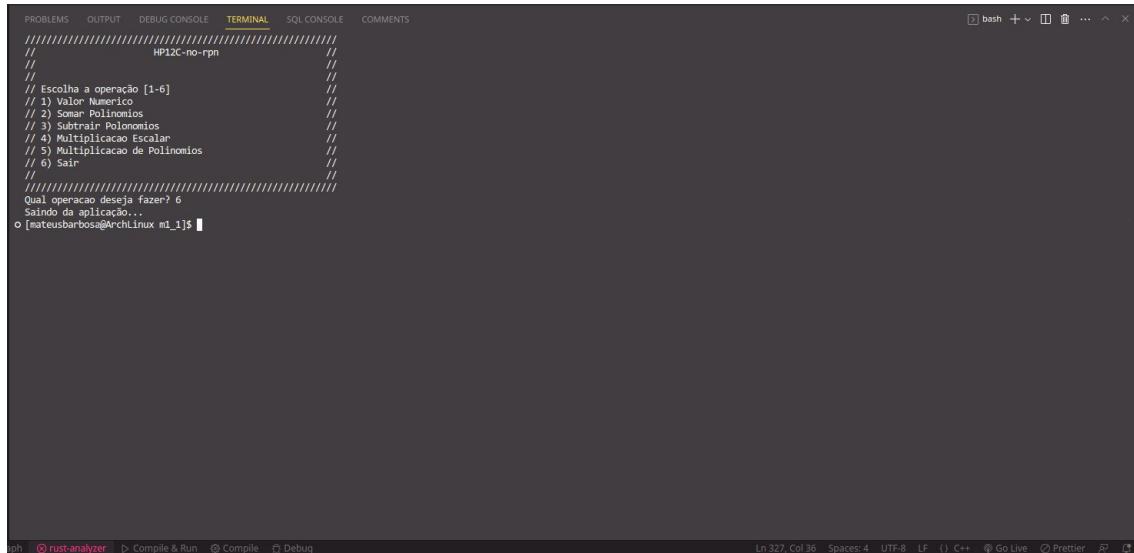
$$f(x) = 3x^4 - 6x^3 + 2x^2 + 2x^1 - 1$$



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
/////////////////////////////////////////////////////////////////
//
// HP12C-no-rpn
//
//
// Escolha a operação [1-6]
// 1) Valor Numerico
// 2) Somar Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
/////////////////////////////////////////////////////////////////
Qual operacao deseja fazer? 5
Digite um polinomio (Ex: 4x^2 + x^1 - 5): 3x^2 - 1
Digite outro polinomio (Ex: 4x^2 + x^1 - 5): x^2 - 2x^1 + 1
Resultado: 3x^4 - 6x^3 + 2x^2 + 2x - 1
Precione Enter tecla para continuar...
|
```

Resultado da operação 5) Multiplicação de Polinômios

E finalmente, após voltar ao menu principal, pressionamos 6 e Enter para sair do programa:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE COMMENTS
/////////////////////////////////////////////////////////////////
//
// HP12C-no-rpn
//
// Escolha a operacao [1-6]
// 1) Valor Numerico
// 2) Somar Polinomios
// 3) Subtrair Polinomios
// 4) Multiplicacao Escalar
// 5) Multiplicacao de Polinomios
// 6) Sair
//
// Qual operacao deseja fazer? 6
// Saindo da aplicacao...
o [mateusbarbosa@ArchLinux ml_1]$
```

Resultado da seleção 6) Sair do Programa

## 5. Conclusão

Portanto, podemos compreender que o programa em questão, que utiliza a LDE é extremamente eficaz e performático no que realiza, as lógicas apesar de serem básicas e de fácil entendimento, possuem uma ampla gama de implementações em programas reais, como uma calculadora por exemplo.

Além disso, percebemos que não basta simplesmente saber programar, é necessário entender o que e como se está programando, para encontrar e melhor solução para um problema. Graças as estruturas como LDE, LUE, pilha, fila, etc, podemos desenvolver programas muito mais complexo e limitando as ações para somente aquilo que queremos permitir na estrutura.

## Bibliografia

BRASIL ESCOLA. Polinômios. Disponível em:  
<https://brasilescola.uol.com.br/matematica/polinomios.htm>. Acesso em: 08 abr. 2023.

MUNDO EDUCAÇÃO. Polinômios. Disponível em:  
<https://mundoeducacao.uol.com.br/matematica/polinomios.htm>. Acesso em: 08 abr. 2023.

ESCOLA KIDS. Polinômios. Disponível em:  
<https://escolakids.uol.com.br/matematica/polinomios.htm>. Acesso em: 08 abr. 2023.

YOUTUBE. Polinômios - Matemática - Descomplica. Disponível em:  
[https://www.youtube.com/watch?v=N\\_oFTs1-mMg](https://www.youtube.com/watch?v=N_oFTs1-mMg). Acesso em: 08 abr. 2023.

YOUTUBE. Polinômios - Aula 01 - Introdução. Disponível em:  
<https://www.youtube.com/watch?v=Nbb7phGDDZM>. Acesso em: 08 abr. 2023.

YOUTUBE. Polinômios - Aula 02 - Operações Básicas. Disponível em: <https://www.youtube.com/watch?v=Nzl7EIN-30M>. Acesso em: 08 abr. 2023.

STACK OVERFLOW. Null vs nullptr: why was it replaced? Disponível em: <https://stackoverflow.com/questions/20509734/null-vs-nullptr-why-was-it-replaced>. Acesso em: 08 abr. 2023.

STROUSTRUP, Bjarne. The C++ Programming Language, FAQ: Null Pointers. Disponível em: [https://www.stroustrup.com/bs\\_faq2.html#null](https://www.stroustrup.com/bs_faq2.html#null). Acesso em: 08 abr. 2023.

UNICAMP. Algoritmos e Programação de Computadores. Disponível em: <https://www.ic.unicamp.br/~ra069320/PED/MC102/1s2008/Apostilas/Cap10.pdf>. Acesso em: 08 abr. 2023.

FREECODECAMP. O que é a notação Big O? Complexidade de tempo e de espaço. Disponível em: <https://www.freecodecamp.org/portuguese/news/o-que-e-a-notacao-big-o-complexidade-de-tempo-e-de-espaco/>. Acesso em: 08 abr. 2023.