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

**Marcos Cesar Cardoso Carrard**

**22/06/2023**

1. Introdução (TODO)

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 do Código

Esse programa em C++ foi realizado para comparar o desempenho de diferentes algoritmos de ordenação em vetores. O programa permite que o usuário especifique o número de vetores, o número de elementos de cada vetor e o número de execuções dos algoritmos.

Algumas alterações foram realizadas nas funções de ordenação para permitir que sejam chamadas dinamicamente dentro da função de “ordenar”. Basicamente o que foi realizado é a troca da constante TAM por um parâmetro tam. E na função de ordenação “merge” (chamada dentro da MergeSort), removemos o vetor dinamico com new e delete por um array estático.

Além disso, criamos pseudo funções de cada função de ordenação, para facilitar a chamada dinâmica delas dentro da função de “ordenar”. Todas as pseudo funções recebem o mesmo parâmetro, sendo ele: o vetor em que será feito a ordenação.

A função preencher\_aleatorio recebe um vetor e seu tamanho como parâmetros e preenche o vetor com valores aleatórios.

A função preencher recebe uma matriz, o número de vetores e o número de elementos como parâmetros. A matriz é redimensionada para ter o número especificado de vetores e elementos. Em seguida, os dois primeiros vetores são preenchidos de forma ordenada e invertida, respectivamente. A partir do terceiro vetor, a função preencher\_aleatorio é chamada para preencher os vetores restantes com valores aleatórios.

Na função mostrar é uma sobrecarga para poder exibir somente o vetor ou a matriz inteira, ela pode receber um vetor ou matriz. Ela verifica se o vetor ou matriz está vazio e, em seguida, imprime os elementos usando formatação especial, para criar espaçamentos com setw.

A função deletar recebe uma matriz como parâmetro e limpa a matriz com o método clear(), removendo todos os seus elementos.

A função exportar\_para\_csv recebe o nome de um arquivo, um cabeçalho, e uma matriz de valores como parâmetros. Ela exporta os resultados para um arquivo CSV, escrevendo o cabeçalho e, em seguida, iterando sobre a matriz para escrever os valores correspondentes. O arquivo é fechado após a escrita.

A função ordenar recebe uma matriz, o número de vetores, o número de elementos, e o número de execuções como parâmetros. Ela usa uma matriz de funções de ordenação e executa cada função para cada vetor na matriz. O tempo de execução de cada iteração é medido usando chrono::high\_resolution\_clock e armazenado em um vetor resultado\_funcao. Em seguida, são calculados o melhor caso, a média dos casos e o pior caso para cada função de ordenação. Os resultados são armazenados em uma matriz resultados.

A função ordenar tem como objetivo realizar a ordenação de diferentes vetores usando diferentes algoritmos de ordenação. Ela recebe como parâmetros uma matriz contendo os vetores a serem ordenados, o número de vetores, o número de elementos em cada vetor e o número de execuções dos algoritmos. Os resultados das execuções são armazenados na matriz resultados.

A função começa criando um vetor de strings chamado nome\_funcoes, que contém os nomes dos algoritmos de ordenação usados. Em seguida, é criado um vetor de funções chamado funcoes, que contém ponteiros para as funções dos algoritmos de ordenação. Essas funções são definidas como lambdas e implementam os algoritmos de ordenação específicos, como shell sort, bubble sort, quicksort, etc.

A matriz resultados é redimensionada para ter o número de linhas igual ao tamanho do vetor funcoes e o número de colunas igual a 3. Essa matriz será usada para armazenar os resultados das execuções dos algoritmos.

Em seguida, há um loop que percorre cada função de ordenação no vetor funcoes. Para cada função, é criado um vetor chamado resultado\_funcao, que armazena o tempo de execução de todas as iterações da função atual. Esse vetor será usado para calcular o tempo médio, o melhor caso e o pior caso de cada função.

Dentro desse loop, há mais dois loops. O primeiro loop é responsável por executar a função de ordenação numero\_execucoes vezes. O segundo loop percorre os vetores na matriz matriz e chama a função de ordenação para cada um deles. Antes e depois da chamada da função de ordenação, são obtidos o tempo de início e o tempo de término usando a biblioteca chrono.

Após a execução dos loops, o vetor resultado\_funcao contém os tempos de execução de todas as iterações da função atual. Em seguida, são calculados o tempo médio, o melhor caso e o pior caso a partir desse vetor.

A variável soma é usada para calcular a soma de todos os tempos de execução. A variável melhor\_caso é inicializada com o primeiro elemento do vetor resultado\_funcao, enquanto pior\_caso também recebe o primeiro elemento. Em seguida, há um loop que percorre o vetor resultado\_funcao e atualiza os valores de melhor\_caso e pior\_caso se um tempo de execução menor ou maior for encontrado, respectivamente.

Os valores de melhor\_caso, soma / (numero\_execucoes \* matriz.size()) (tempo médio) e pior\_caso são armazenados nas posições apropriadas da matriz resultados para a função atual.

Após o término do loop que percorre as funções de ordenação, uma mensagem é exibida indicando que os algoritmos foram finalizados.

A função menu é o ponto de entrada principal do programa. Ela exibe um menu para o usuário com as opções "Iniciar" e "Sair". Se o usuário escolher "Iniciar", ele pode especificar o número de vetores, o número de elementos e o número de execuções. Em seguida, a matriz é preenchida, os algoritmos de ordenação são executados, os resultados são exportados para um arquivo CSV e exibidos na tela.

Por fim, a função main chama a função menu para iniciar o programa.

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).

4.1 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 intervalor 1 ao 6, sendo que a opção 6 sai do programa. Por simplicidade, nos exemplos abaixos 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).

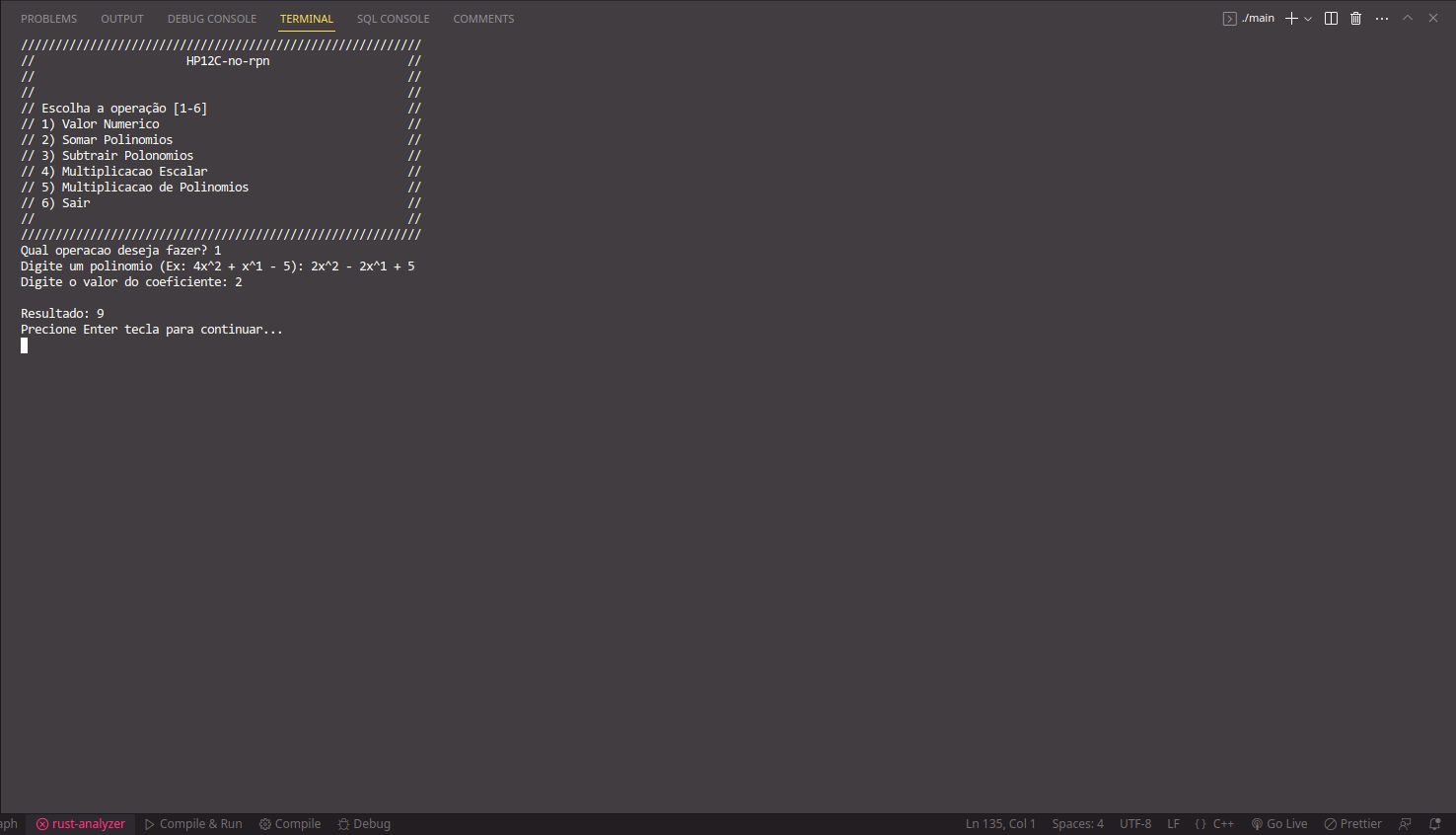


Imagem do menu principal do programa

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



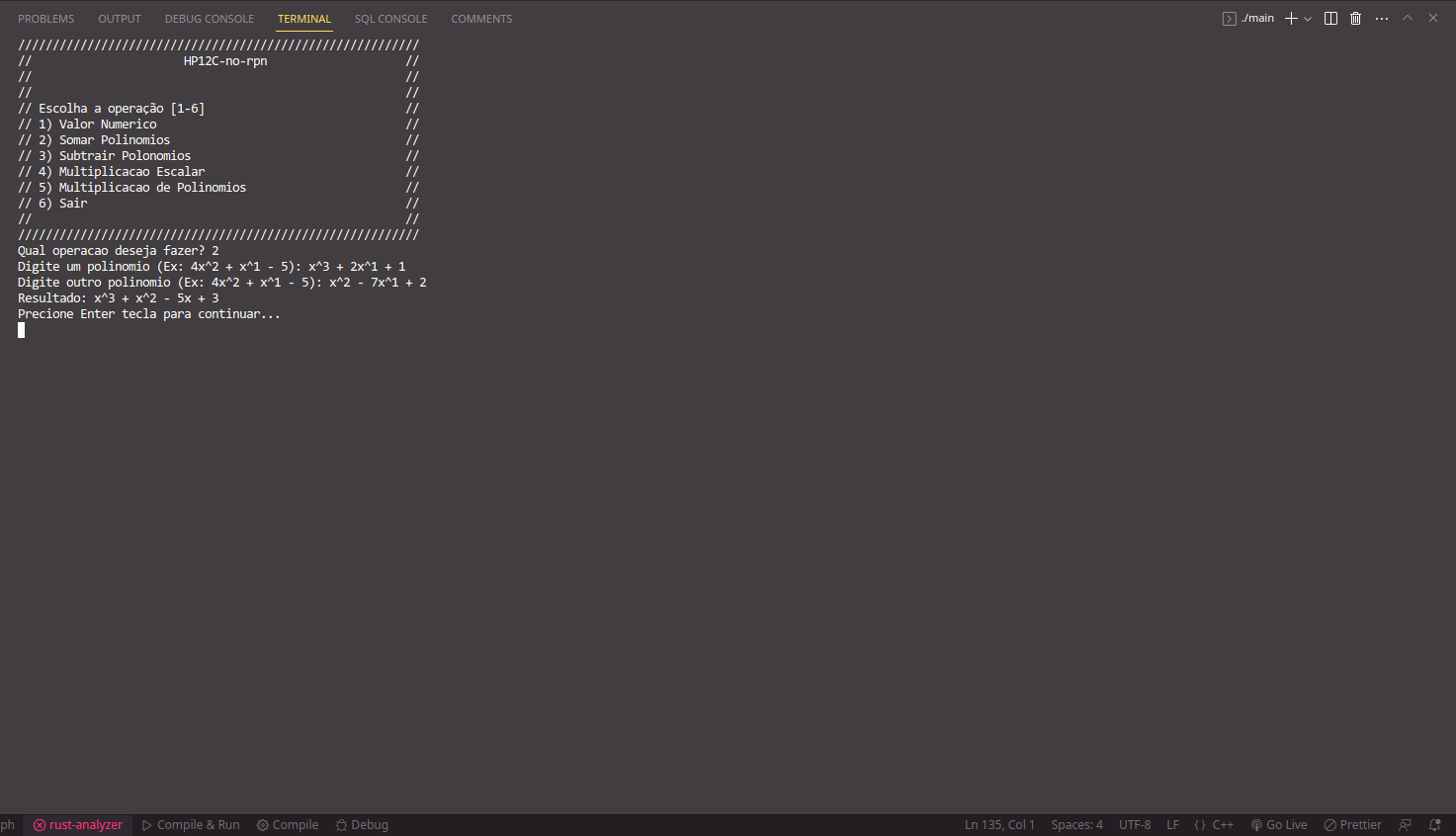
Resultado da operação 1) Valor Numérico

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



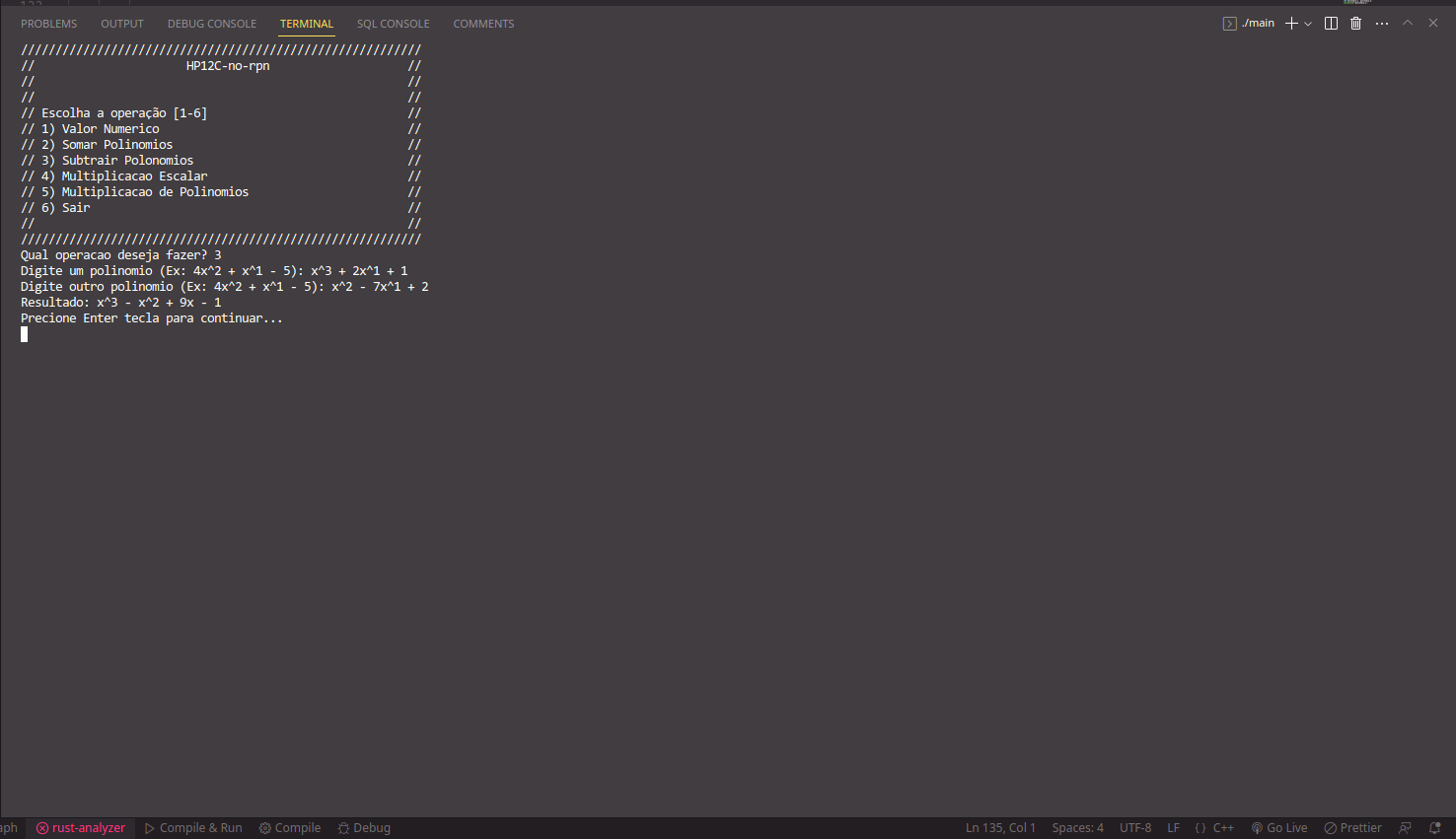
Resultado da operação 2) Soma de Polinômios

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

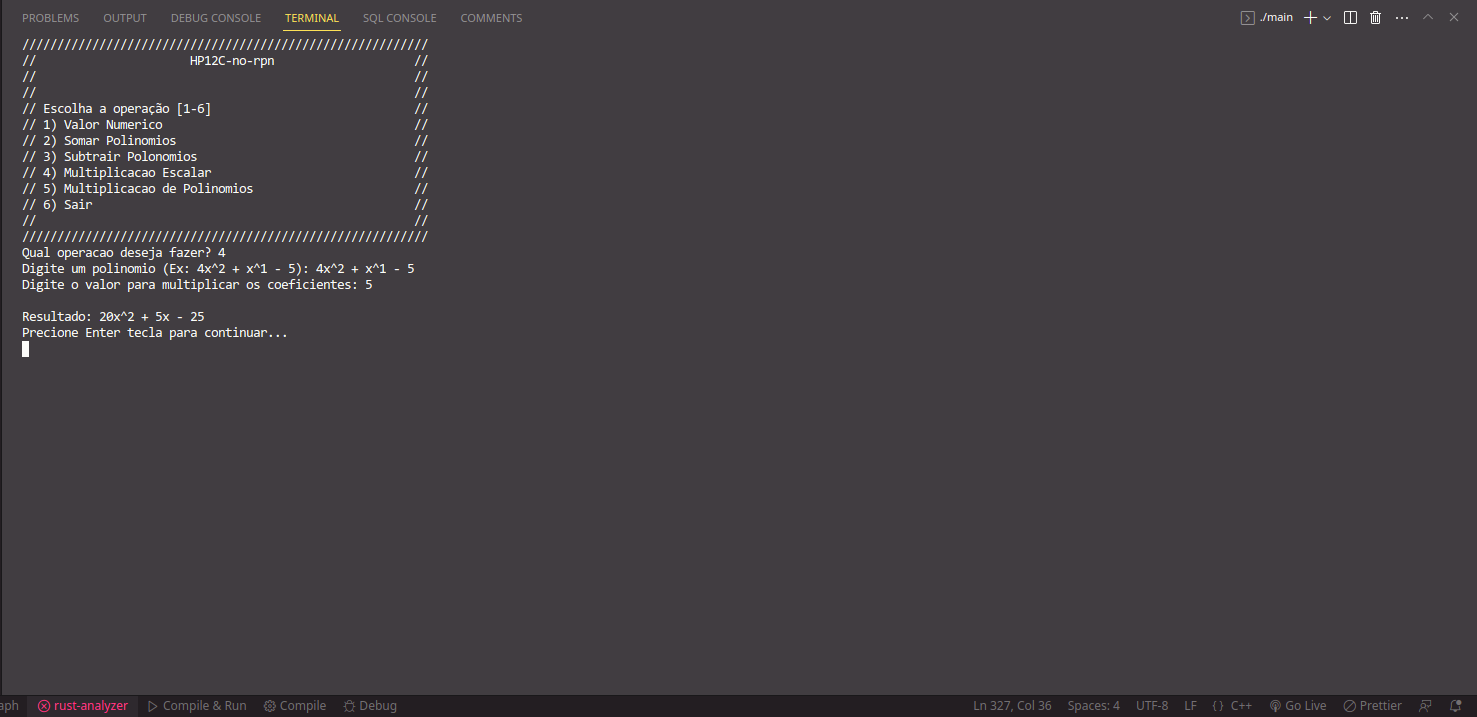


Resultado da operação 3) Subtração de Polinômios

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



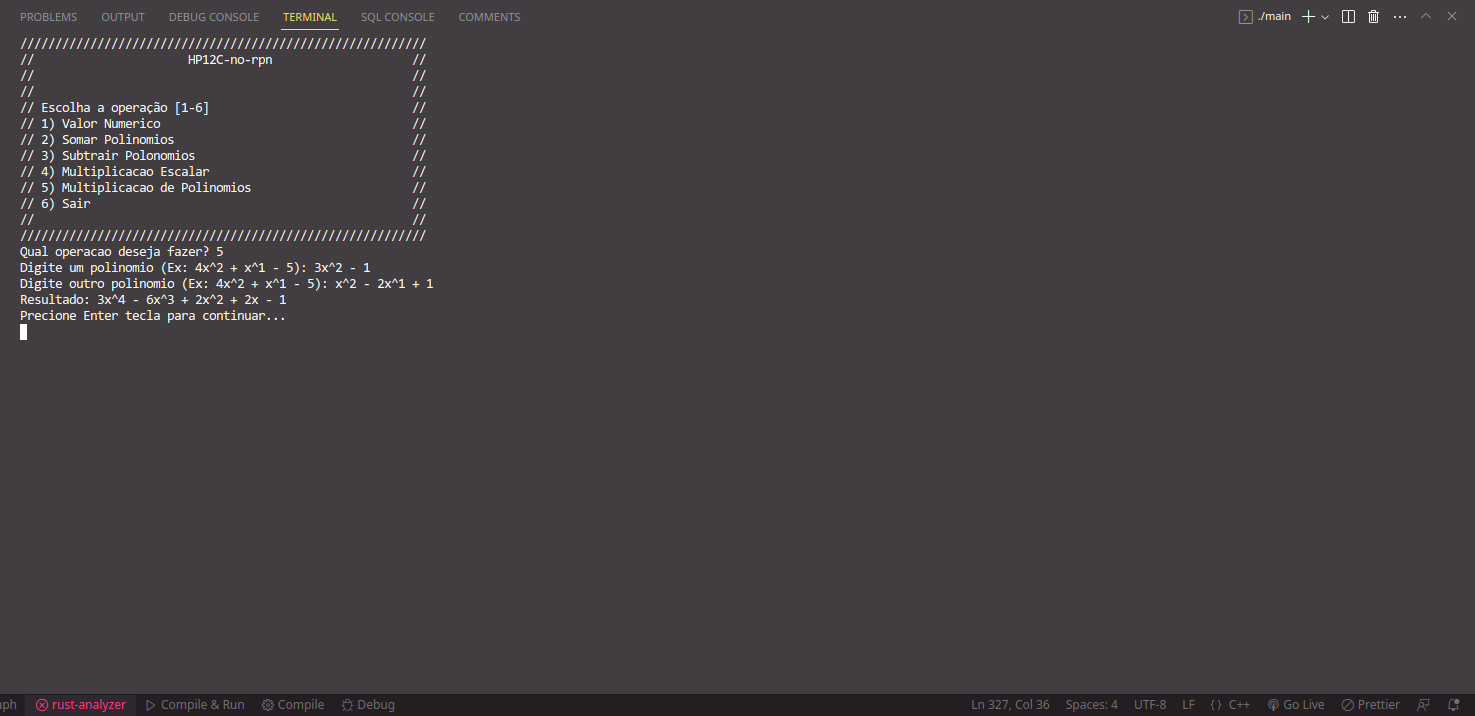
Resultado da operação 4) Multiplicação Escalar

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)

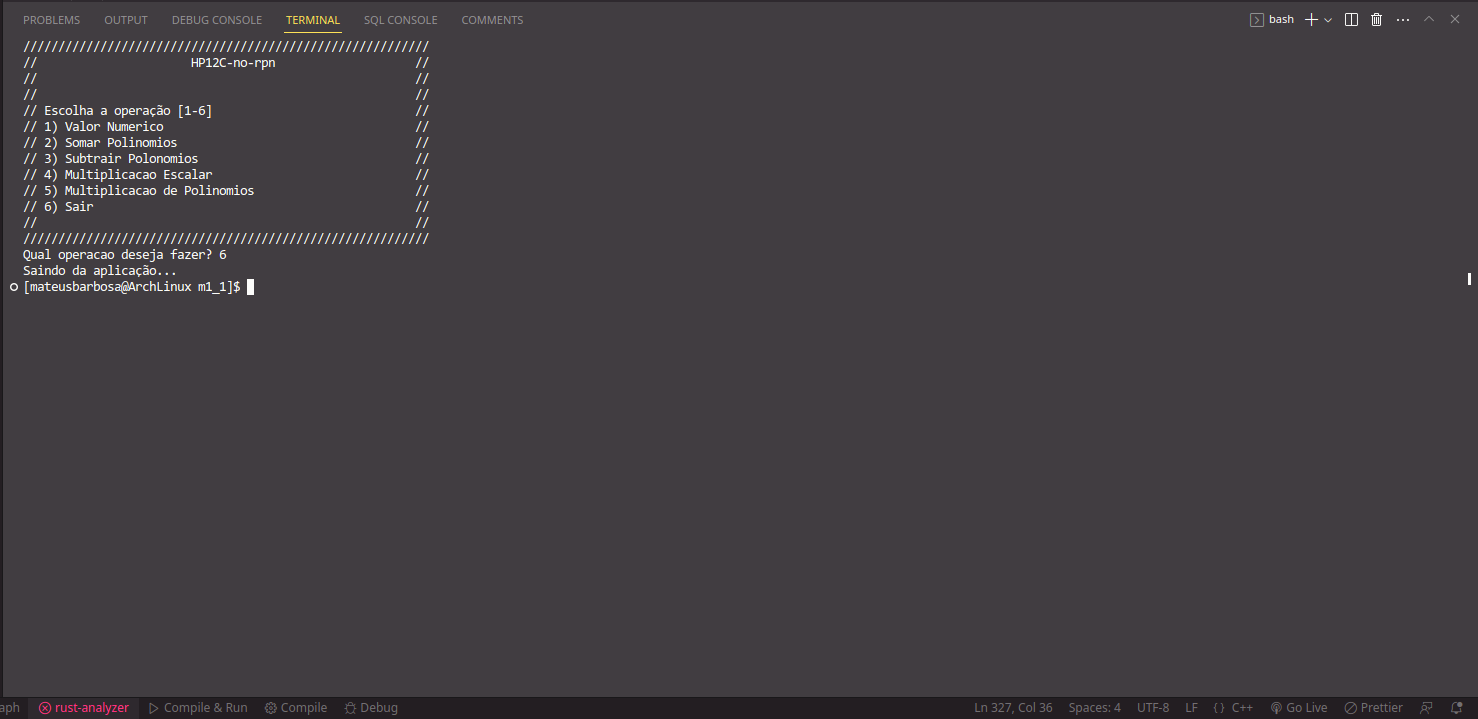
f(x) = (3x^4 - 6x^3 + 3x^2 - x^2 +2x^1 - 1)

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



Resultado da operação 5) Multiplicação de Polinômios

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



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>. 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=Nzl7ElN-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>. 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.