Nomes: João Vitor da Silva Mairena (RA: 11202320852), Andrea Paixão Bueno (R.A: 11202232113), Alexandre Hiroyuki Yamauchi (RA: 11202230885), Gabriel Abreu Silva (RA: 11202231422)
Professor: Roberto Sadao Yokoyama

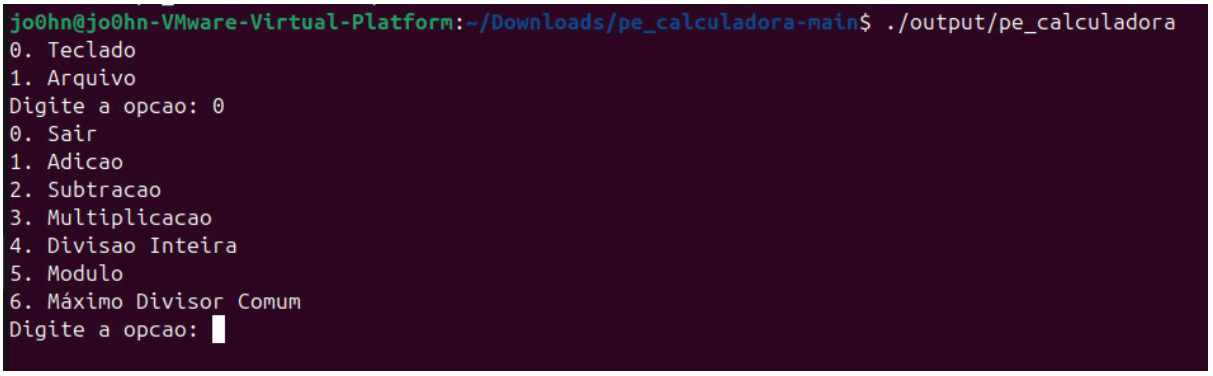
Programação Estruturada - 2025.3
Relatório Projeto Calculadora

Este projeto tem como objetivo desenvolver uma calculadora capaz de operar com números inteiros de tamanho arbitrário (BigInt) utilizando a linguagem C. Como os tipos inteiros nativos da linguagem possuem limites de armazenamento, foi necessário criar uma estrutura que permita representar números muito grandes, superando as restrições do hardware.

Para isso, os números são armazenados em listas encadeadas, onde cada nó guarda um bloco de 32 bits. A partir dessa estrutura, foram implementadas as operações de adição e subtração, incluindo tratamento de sinal, propagação de carry/borrow e remoção de zeros à esquerda. Também foi integrado um sistema de entrada e saída via arquivos, possibilitando o processamento de múltiplos pares de números de forma automática.

Requisitos implementados e casos de teste

A calculadora possui duas opções de entrada de dados. Na opção 0, o usuário insere manualmente os números a serem processados. Na opção 1, o programa realiza a leitura automática de um arquivo .txt para obter os valores.



Opção 0 — Entrada com Dados do Teclado:

Nesta opção, o usuário insere os números pelo teclado.

- ### Opção 1 — Adição de BigInts

  Requisitos implementados:

  1. Soma entre BigInts:

A função soma corretamente números inteiros grandes representados em blocos de 32 bits, independentemente do tamanho.

2. Tratamento de sinais diferentes:
   Quando os números possuem sinais opostos, a função identifica que a operação é uma subtração e encaminha para a função adequada.
3. Cálculo correto do sinal:
   Se ambos os valores possuem o mesmo sinal, o resultado mantém esse sinal, caso contrário, a soma vira uma subtração que determina o sinal final conforme a maior magnitude.
4. Manipulação da estrutura BigInt:
   A função utiliza operações de lista encadeada para acessar, somar e armazenar cada bloco de 32 bits, construindo o resultado de forma organizada e eficiente.
5. Algoritmo de soma com avanço do excesso:
   A soma é realizada bloco por bloco, e sempre que a soma ultrapassa o valor máximo de 32 bits, o excesso é passado para o próximo bloco.
6. Remoção de zeros à esquerda:
   Após o cálculo, os zeros no início do resultado BigInt são removidos, mantendo a forma correta do resultado.

- Teste 1 - 9,999999999999999e^49 + 1: == 10^50:

```
jo0hn@jo0hn-VMware-Virtual-Platform:~/Downloads/pe_calculadora-main$ ./output/pe_calculadora
0. Teclado
1. Arquivo
Digite a opcao: 0
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 1
Adicao
Digite o primeiro numero: 9999999999999999999999999999999999999999999999999
Digite o segundo numero: 1
Resultado: 10000000000000000000000000000000000000000000000000
```

Figura 1.1 – Resultado da operação de adição.

- Teste 2 - 1,2345678901234568e^49 + 1,1111111111111111e^49 == 2,3456790012345679e^49:

```
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 1
Adicao
Digite o primeiro numero: 12345678901234567890123456789012345678901234567890
Digite o segundo numero: 11111111111111111111111111111111111111111111111111
Resultado: 23456790012345679001234567900123456790012345679001
```

Figura 1.2 – Resultado da operação de adição.

- Teste 3 - 9,876543219876543e^49 + 1,2345678912345678e^49 == 1,111111111111111e^50:

```
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 1
Adicao
Digite o primeiro numero: 98765432198765432198765432198765432198765432198765
Digite o segundo numero: 12345678912345678912345678912345678912345678912345
Resultado: 111111111111111111111111111111111111111111111111110
```

Figura 1.3 – Resultado da operação de adição.

● Opção 2 — Subtração de BigInts

Requisitos implementados:

1. Subtração entre BigInts:
   A função realiza corretamente a subtração entre números grandes representados em blocos de 32 bits, independentemente do tamanho.
2. Tratamento de sinais diferentes:
   Quando os números possuem sinais opostos, a função identifica que a operação equivale a uma soma e encaminha para a função de soma.
3. Cálculo correto do sinal:
   Quando os sinais são iguais, o sinal do resultado é determinado comparando as magnitudes dos dois números; o maior em valor absoluto define o sinal final.
4. Manipulação correta da estrutura BigInt:
   A função utiliza operações da lista encadeada para acessar, comparar e atualizar blocos individuais, construindo o resultado passo a passo.

5. Algoritmo completo de subtração com empréstimo:
   A subtração é realizada bloco por bloco, e quando o bloco atual não possui valor suficiente, a função realiza um empréstimo do próximo bloco, garantindo precisão total na operação.
6. Remoção de zeros à esquerda:
   Após a operação, blocos de zeros desnecessários no início da representação são removidos, mantendo o número corretamente formatado.

- Teste 1 - 1,0e^50 - 9,999999999999999e^49 == 1:

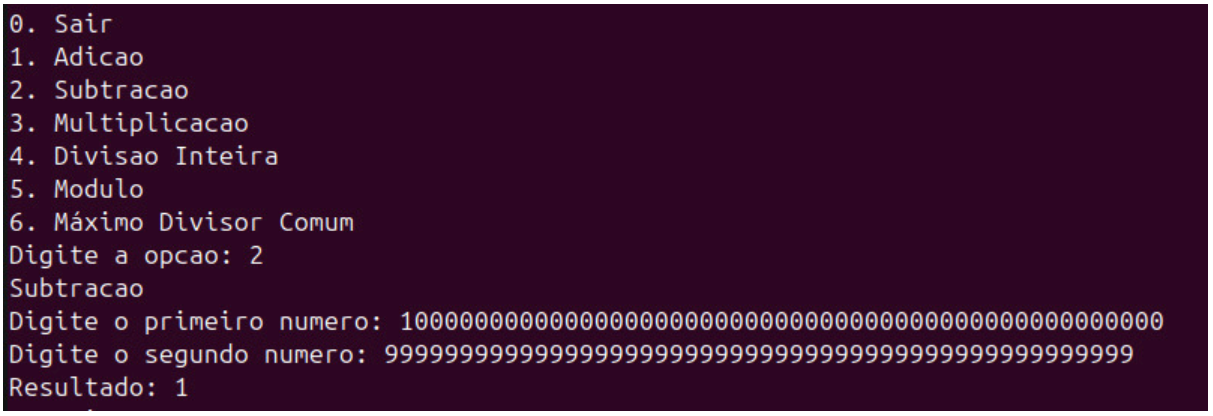


Figura 2.1 – Resultado da operação de subtração.

- Teste 2 - 9,876543219876543e^49 - 1,2345678912345678e^49 == 8,641975328641975e^49:

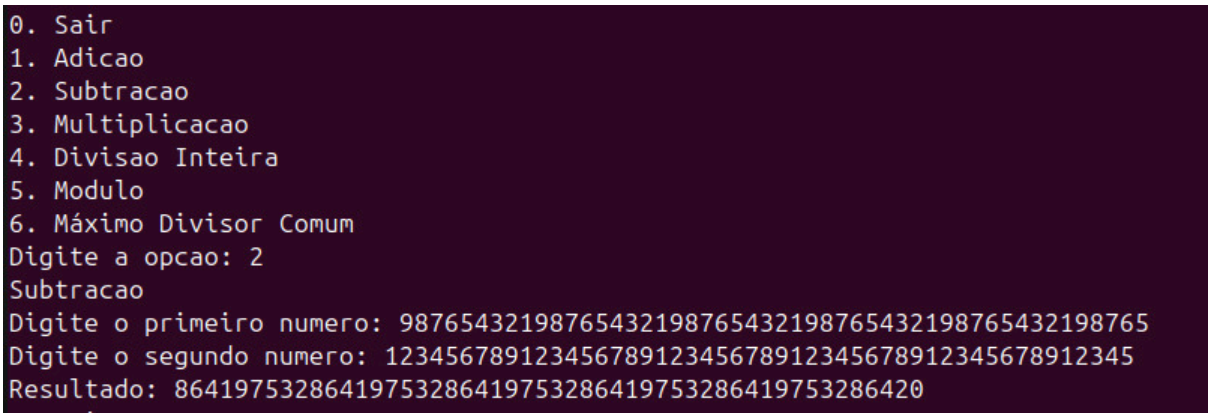


Figura 2.2 – Resultado da operação de subtração.

- Teste 3 - 5,555555555555556e^48 - 9,999999999999999e^48 == -4,444444444444443e^48:

```
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 2
Subtracao
Digite o primeiro numero: 555555555555555555555555555555555555555555555555
Digite o segundo numero: 999999999999999999999999999999999999999999999999
Resultado: -444444444444444444444444444444444444444444444444
```

Figura 2.3 – Resultado da operação de subtração.

- **Opção 3 — Multiplicação de BigInts**

  Requisitos Implementados:

  1. Multiplicação entre BigInts:
     A função realiza corretamente a multiplicação de números inteiros grandes usando blocos de 32 bits.
  2. Tratamento de operandos zero:
     Se qualquer operando for zero, o resultado retornado é zero imediatamente.
  3. Cálculo correto do sinal:
     O resultado recebe sinal positivo ou negativo conforme a combinação dos sinais dos operandos.
  4. Manipulação correta da estrutura BigInt:
     Usamos as funções auxiliares de lista encadeada para armazenar, acessar e atualizar os blocos do número.
  5. Algoritmo completo de multiplicação com carry:
     Implementa o algoritmo de multiplicação, fazendo o excesso do resultado passar para o próximo bloco.
  6. Remoção de zeros à esquerda:
     Após o cálculo, o código remove os zero desnecessários do início do resultado final.

  - Teste 1 - 0 × 987654321 == 0:

```
Digite a opcao: 0
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 3
Multiplicacao
Digite o primeiro numero: 0
Digite o segundo numero: 987654321
Resultado: 0
```

Figura 3.1 – Resultado da operação de multiplicação.

- Teste 2 - 123456789123456789 × 987654321987654321 == 1,2193263e^35:

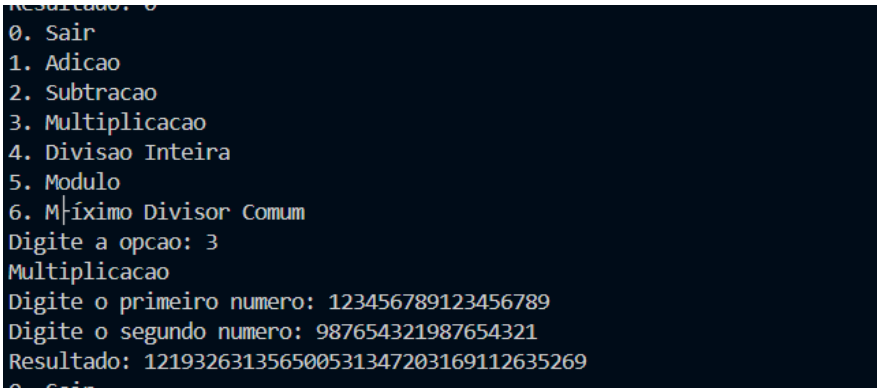


Figura 3.2 – Resultado da operação de multiplicação.

- Teste 3 - 123456789 × -987654321 == -1,2193263e^17 :

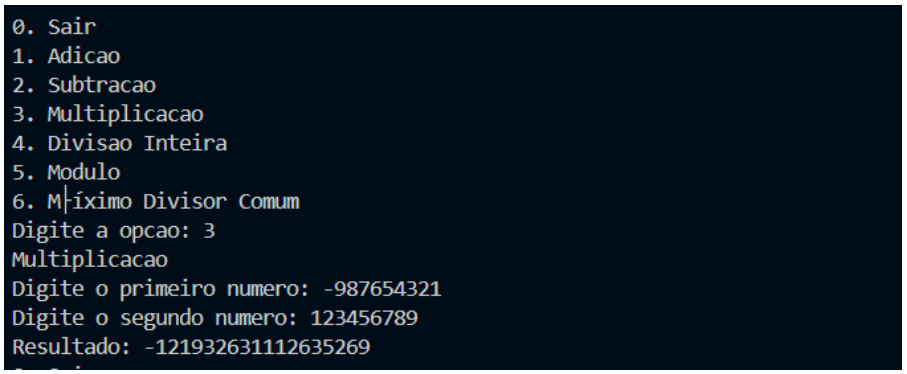


Figura 3.3 – Resultado da operação de multiplicação.

- **Opção 4 — Divisão Inteira de BigInts**

Requisitos Implementados:

1. Divisão entre BigInts:
   A função faz a divisão inteira entre números grandes representados em blocos de 32 bits.
2. Tratamento de divisor zero:
   O código detecta quando o divisor é zero e encerra a operação retornando NULL.
3. Cálculo correto do sinal:
   O quociente recebe sinal positivo ou negativo conforme a combinação dos sinais do dividendo e do divisor.
4. Manipulação correta da estrutura BigInt:
   A função utiliza as operações auxiliares de lista encadeada para copiar valores, armazenar blocos, comparar magnitudes e atualizar resultados.
5. Algoritmo de divisão por aproximação sucessiva:

Implementa uma divisão baseada em subtração, onde é encontrado repetidamente a maior potência de 2 do divisor que ainda cabe no dividendo.

6. Remoção de zeros à esquerda:
   Após ler o quociente, o código remove zeros desnecessários do início do resultado final.

- Teste 1 - 987654321987654321 / 123456789 == 8000000080:

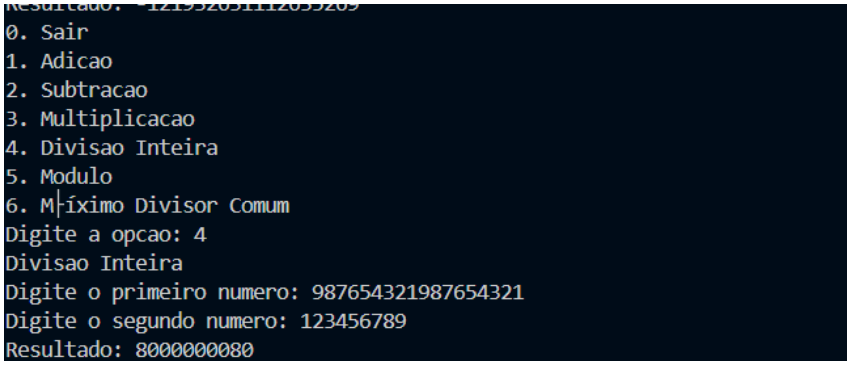


Figura 4.1 – Resultado da operação de divisão.

- Teste 2 - 1000 / 64 == 15:

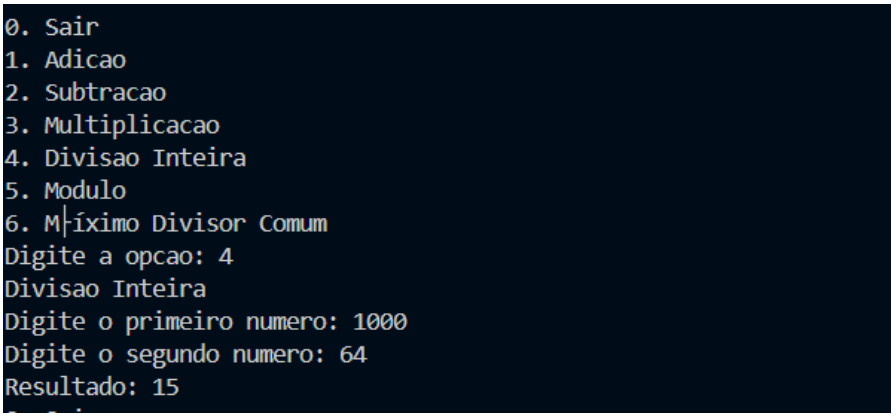


Figura 4.2 – Resultado da operação de divisão.

- Teste 3 - 5000 / -200 == -25:

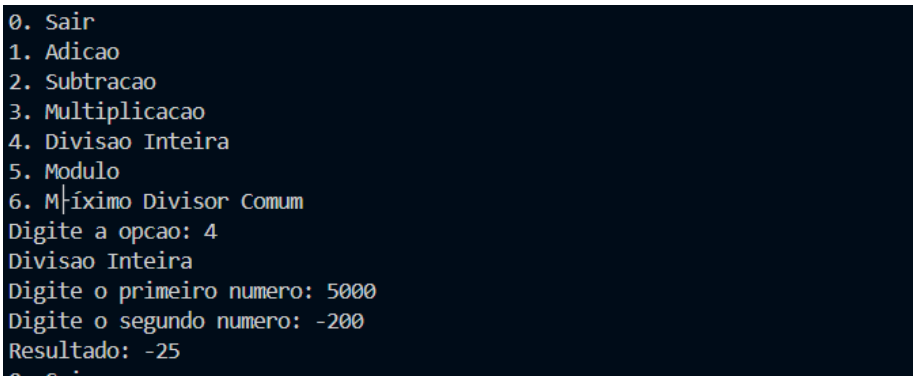


Figura 4.3 – Resultado da operação de divisão.

- **Opção 5 — Módulo de BigInts**

  Requisitos implementados:

  1. Cálculo do módulo entre BigInts:
     A função determina corretamente o resto da divisão entre números grandes representados em blocos de 32 bits.
  2. Tratamento de divisor zero:
     O código verifica se o segundo número é zero e retorna NULL, evitando erro de operação inválida.
  3. Cópia de Valor Absoluto:
     Cria cópias de a e n (dividendo_abs e divisor_abs) e define seus sinais como positivos (sign = 1) para trabalhar com suas magnitudes.
  4. Cálculo da Magnitude do Quociente:
     Calcula o valor absoluto do quociente, $q\_abs = |a| / |n|$, usando bigint_divisao.
  5. Determinação do Sinal do Quociente:
     Se a e n têm sinais diferentes, q é negativo (q->sign = -1). Caso contrário, q é positivo (q->sign = 1).
  6. Cálculo de $q \cdot n$:
     Multiplica o quociente ajustado (q) pelo divisor original (n) usando bigint_multiplicação.
  7. Cálculo do Resto (r):
     Subtrai o resultado de $q \cdot n$ do dividendo original a usando bigint_subtract para obter o resto inicial resto.
  8. Remoção de zeros à esquerda:
     O resultado final tem zeros desnecessários removidos.

  - Teste 1 - 1000 % 25 == 0:

```
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. M├íximo Divisor Comum
Digite a opcao: 5
Modulo
Digite o primeiro numero: 1000
Digite o segundo numero: 25
Resultado: 0
```

Figura 5.1 – Resultado da operação de módulo.

  - Teste 2 - 987654321987654321 % 123456789 == 111111201:

```
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. M├íximo Divisor Comum
Digite a opcao: 5
Modulo
Digite o primeiro numero: 987654321987654321
Digite o segundo numero: 123456789
Resultado: 111111201
```

Figura 5.2 – Resultado da operação de módulo.

- Teste 3 - 350 % 60 == 5:

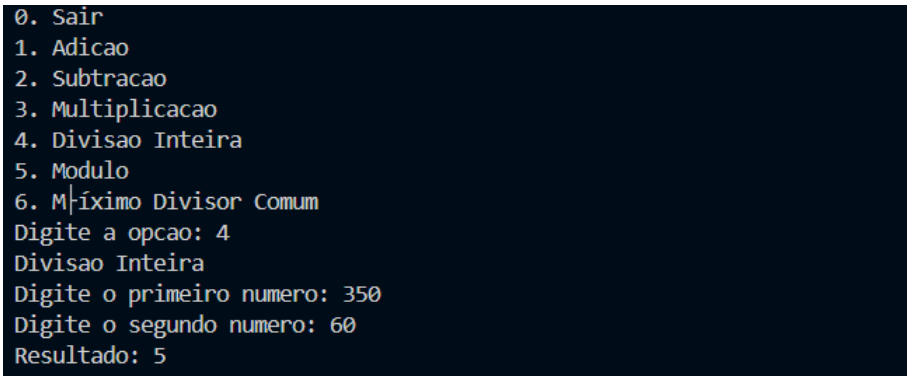


Figura 5.3 – Resultado da operação de módulo.

- ## Opção 6 — Máximo Divisor Comum de BigInts

Requisitos implementados:

1. Cálculo do MDC entre BigInts:
   A função bigint_gcd calcula o máximo divisor comum entre dois números grandes representados em blocos de 32 bits.
2. Uso do algoritmo de Euclides:
   A implementação utiliza o algoritmo clássico de Euclides, repetindo operações de módulo até que um dos valores chegue a zero.
3. Manipulação correta da estrutura BigInt:
   A função trabalha com cópias dos números originais e utiliza operações auxiliares como módulo, cópia, destruição e verificação de zero para gerenciar corretamente os blocos do BigInt.
4. Integração com a função de módulo:
   Cada passo do MDC depende da operação 'a mod b', garantindo que o cálculo siga o comportamento correto da aritmética entre BigInts.
5. Detecção correta de zero:
   A função utiliza bigint_is_zero para identificar quando o valor chegou a zero, o que encerra o algoritmo no momento adequado.
6. Liberação adequada de memória:
   Ao longo do processo, objetos temporários são destruídos corretamente, evitando vazamentos de memória durante o cálculo do MDC.

- Teste 1 - 45397369416530795319729696969741061923382 6 e 28057117299251014003761193241303867718952 5:

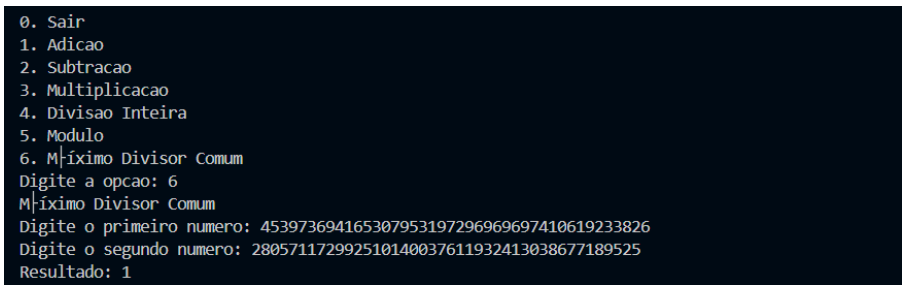


Figura 6.1 – Resultado da operação de MDC.

- Teste 2 - 170141183460469231731687303715884105727 e 2305843009213693951:

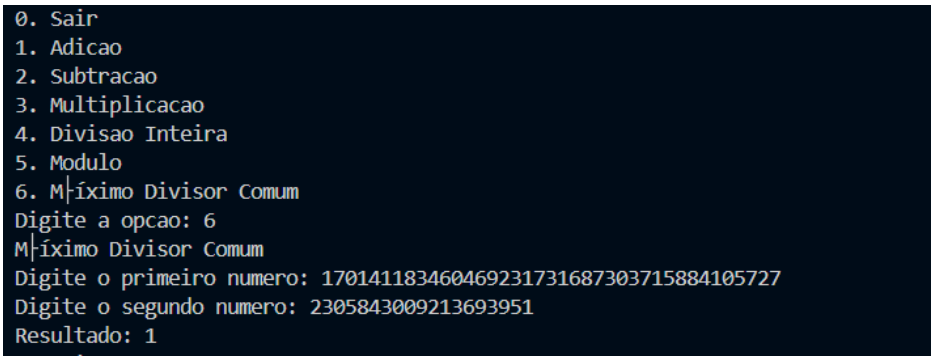


Figura 6.2 – Resultado da operação de MDC.

- Teste 3 - 170141183460469231731687303715884105727 e 170141183460469231731687303715884105728:

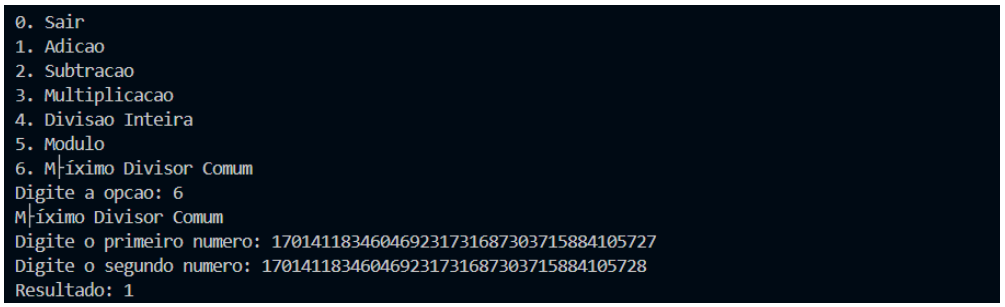
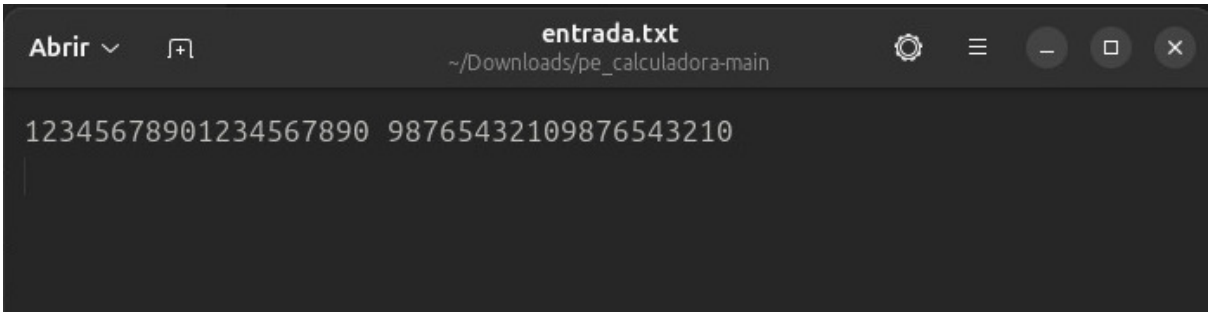


Figura 6.3 – Resultado da operação de MDC.

Opção 1 — Leitura de pares via arquivo (entrada.txt):

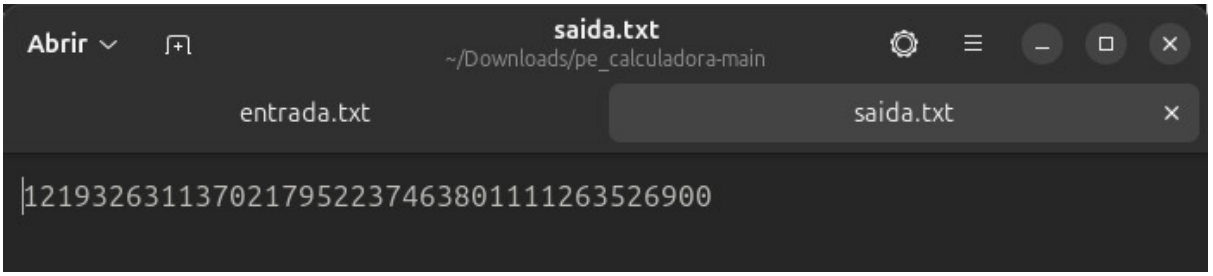
Primeiramente, anexamos um arquivo de entrada com dois números inteiros como no exemplo a seguir:



Em seguida, iniciamos a calculadora na opção 1 e digitamos o nome do arquivo de entrada (arquivo que criamos) e escolhemos um nome para o arquivo de saída:

```
jo0hn@jo0hn-VMware-Virtual-Platform:~/Downloads/pe_calculadora-main$ ./output/pe_calculadora
0. Teclado
1. Arquivo
Digite a opcao: 1
Nome do arquivo de entrada: entrada.txt
Nome do arquivo de saida: saida.txt
0. Sair
1. Adicao
2. Subtracao
3. Multiplicacao
4. Divisao Inteira
5. Modulo
6. Máximo Divisor Comum
Digite a opcao: 3
Multiplicacao
Resultado gravado em saida.txt
```

Agora, consultamos o arquivo de saída com o resultado esperado:



Execução da calculadora em um arquivo com duas entradas.