

POLY – REPRESENTAÇÃO DE POLINÔMIOS

PROFESSOR: ADELARDO ADELINO DANTAS DE MEDEIROS

Escreva uma classe em C++, denominada `Poly`, capaz de representar polinômios $P(x)$ de qualquer grau n ($n \geq 0$) com coeficientes reais a_i , do tipo:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Um polinômio de grau n será representado por um array dinâmico unidimensional de $n+1$ coeficientes, armazenado internamente por dois dados:

- dimensão `D` (inteiro sem sinal, igual a $n+1$); e
- ponteiro `a`, apontando para um array com `D` números reais.

```
class Poly {
private:
    /// Dimensao (grau+1)
    unsigned D;
    /// Coeficientes
    double *a;
    ...
}
```

ATENÇÃO: não tente armazenar a dimensão do polinômio através de uma variável que represente o grau n , pois dessa forma você não conseguirá diferenciar o polinômio de grau 0 (dimensão 1), que é equivalente a um número real, de um polinômio vazio (dimensão 0) para o qual o valor do grau não está definido.

O coeficiente de maior grau, a_n , nunca pode ser nulo, exceto para polinômios de grau 0 ($D=1$), que correspondem a um número real e, portanto, podem ser iguais a zero.

Desenvolva no mínimo as seguintes funcionalidades para a classe `Poly`:

- Crie os construtores (*default*, por cópia e específicos) e o destrutor apropriados:
 - O construtor *default* deve criar um polinômio vazio, com dimensão nula. **ATENÇÃO:** um polinômio vazio (sem coeficientes, $D=0$, $a=nullptr$) é diferente de um polinômio de grau zero (um coeficiente, $D=1$).
 - Um construtor específico deve criar um polinômio do grau (não a dimensão) passado como parâmetro (inteiro não negativo):
 - Se o grau for nulo, deve criar um polinômio com dimensão $D=1$ cujo valor do único coeficiente, a_0 , será 0.0. **ATENÇÃO:** o polinômio criado pelo construtor específico com parâmetro zero (grau=0, $D=1$, $a=new\ double[1]$) é diferente do polinômio criado pelo construtor *default* (grau indefinido, $D=0$).
 - Se o grau for maior que zero, o polinômio deve ter todos os coeficientes nu-

los, com exceção do de maior grau, a_n , que terá valor 1.0.

- Verifique se o construtor específico deve ou não poder ser utilizado como conversor de tipo (ou seja, se ele deve ou não ser *explicit*).

- Crie um método `recrutar` para redefinir o grau e os coeficientes de um polinômio, recebendo como parâmetro (inteiro não negativo) o novo grau (não a dimensão) desejado. A função não retorna nada (*void*). Lembre-se que será necessário prever a liberação de eventual memória alocada anteriormente. O valor dos novos coeficientes deve ser o mesmo que no caso do construtor específico:
 - Se o novo grau $n=0$ ($D=1$) $\rightarrow a_0=0.0$.
 - Caso contrário $\rightarrow a_n=1.0$; $a_i=0.0$, $i=0 \dots n-1$
- Sobrecarregue o `operator=` para atribuir um polinômio a outro. Lembre-se que o objeto que está sendo atribuído pode conter informação anterior, sendo necessário prever a liberação de eventual memória alocada anteriormente.
- Crie um método de consulta `empty` que retorna um booleano que será *true* se o polinômio for vazio (dimensão $D=0$) e *false* caso contrário. A função não tem nenhum parâmetro.
- Crie um método de consulta `isZero` que retorna um booleano que será *true* se o polinômio for de grau zero (dimensão $D=1$) e tiver seu único coeficiente $a_0 = 0$, sendo *false* caso contrário. A função não tem nenhum parâmetro.
- Defina um método de consulta `getGrau` para retornar o grau do polinômio (igual a $D-1$). A função não tem nenhum parâmetro. O valor retornado é um número inteiro com sinal, que será negativo se o polinômio for vazio (*empty*).

ATENÇÃO: o dado membro `D` é um *unsigned*. Se você fizer a subtração $D-1$, quando $D=0$ o resultado não será -1, mas sim um número muito grande (o maior valor possível para um *unsigned*), pois um *unsigned* nunca é negativo (*underflow*). Portanto, ao invés de:

```
int Poly::getGrau() const {
    return D-1; //ERRO se D==0
}
```

faça:

```
int Poly::getGrau() const {
    return int(D)-1; // NAO int(D-1)
}
```

DICA: No restante da implementação, sempre que você precisar utilizar o grau do polinômio, ao invés de fazer a subtração $D-1$ e ter de se preocupar com esse problema de *underflow*, use `getGrau`. Por exemplo, ao invés de:

```
for (int i=0; i<D-1; i++) ... // ERRO
faça:
for (int i=0; i<getGrau(); i++) ...
```

- Defina um método de consulta `getCoef` para retornar o valor do i -ésimo coeficiente do polinômio, sendo i (inteiro não negativo) passado como parâmetro. Por exemplo, se:

$$P(x) = 3x^2 + 5x + 6.2$$

então `P.getCoef(0)` deve retornar 6.2. O valor retornado é um número em ponto flutuante. Lembre-se de checar a validade do índice: caso não corresponda a um dos coeficientes (ou seja, se $i \geq D$), a função deve retornar 0.0 sem emitir nenhuma mensagem de erro.

- Sobrecarregue o `operator[]` utilizando o método `getCoef`, de tal forma que `P[0]` retorne o mesmo que `P.getCoef(0)`.

DICA: No restante da implementação, sempre que você precisar do valor de um coeficiente, ao invés de acessar o elemento do array diretamente e ter de se preocupar com a validade do índice para aquele polinômio, use `getCoef` ou `operator[]`. Por exemplo, ao invés de:

```
prov.a[i] = a[i] + P.a[i];
onde você teria que se preocupar se o índice i
é válido para os dois polinômios, você pode
utilizar getCoef ou operator[], que já retornam
0.0 para coeficientes inexistentes:
prov.a[i] = getCoef(i) + P.getCoef(i);
ou
prov.a[i] = *this[i] + P[i];
```

- Defina um método de consulta `getValor` para retornar o valor do polinômio para um dado valor real de x , passado como parâmetro. Por exemplo, se:

$$P(x) = 3x^2 + 5x + 6.2$$

Então `P.getValor(1.5)` deve retornar um número real igual a:

$$3(1.5)^2 + 5(1.5) + 6.2 = 20.45$$

Caso o polinômio seja vazio (`empty`), a função deve retornar 0.0.

- Sobrecarregue o `operator()` utilizando o método `getValor`, de tal forma que `P(1.5)` retorne o mesmo que `P.getValor(1.5)`.

- Crie um método `setCoef` que permita alterar o valor do i -ésimo coeficiente de um polinômio, recebendo como parâmetros o índice i (inteiro

não negativo) e o novo valor (número real) do coeficiente. A função não retorna nada (`void`). Lembre-se de checar a validade do índice e que o coeficiente de maior grau, a_n , nunca pode ser nulo, **exceto** para polinômios de grau 0. No caso de algum parâmetro inválido, a função deve emitir uma mensagem de erro e não alterar nenhum coeficiente.

- Sobrecarregue o operador `<<` para escrever um polinômio na sua representação usual:
 - Iniciando com o coeficiente de maior grau.
 - Fazendo as adaptações necessárias para coeficientes negativos, unitários ou nulos.
 A tabela a seguir mostra alguns exemplos com o resultado esperado da impressão.

Coeficientes: [a_0 a_1 ... a_{n-1} a_n]	Impressão
[5.7 1.4 3.2 0.2]	$0.2*x^3+3.2*x^2+1.4*x+5.7$
[5.7 1.0 3.2 1.0]	$x^3+3.2*x^2+x+5.7$
[5.7 -1.4 3.2 -0.2]	$-0.2*x^3+3.2*x^2-1.4*x+5.7$
[5.7 -1.0 3.2 -1.0]	$-x^3+3.2*x^2-x+5.7$
[0.0 1.4 0.0 0.2]	$0.2*x^3+1.4*x$
[0.0 1.0]	x
[1.0]	1.0
[0.0]	0.0
Polinômio empty	(nada impresso)

Algoritmo de impressão:

```
// Se grau < 0, não faz nada
Para i de getGrau() até 0:
| Se  $a_i == 0.0$ 
| | Se  $i == 0$  E getGrau() == 0
| | | Imprime  $a_i$ 
| | Fim Se
| Caso contrário
| | // Imprime sinal do coeficiente
| | Se  $a_i < 0.0$ 
| | | Imprime '-'
| | Caso contrário
| | | Se  $i != \text{getGrau}()$ 
| | | | Imprime '+'
| | | Fim Se
| | Fim Se
| // Imprime módulo do coeficiente
| Se  $|a_i| != 1.0$  OU  $i == 0$ 
| | Imprime  $|a_i|$ 
| Fim Se
| // Imprime termo que depende de x
| Se  $i != 0$ 
| | Se  $|a_i| != 1.0$ 
| | | Imprime '*'
| | Fim Se
| | Imprime 'x'
| | Se  $i > 1$ 
| | | Imprime '^'
| | | Imprime i
| | Fim se
| Fim Se
Fim Para
```

- Sobrecarregue o operador `>>` para permitir que o usuário digite os coeficientes de um `Poly`. A dimensão do polinômio não é digitada nessa função e já deve ter sido estabelecida antes da entrada de dados. Lembre-se que o coeficiente de maior grau não pode ser nulo (exceto para polinômios de grau 0): nesse caso, o método deve solicitar que o usuário digite um novo valor até que seja válido.

A digitação deve iniciar pelo coeficiente de maior grau (a_n) até o de menor grau (a_0). Antes de pedir que o usuário digite cada coeficiente (`cin >> ...`), a função deve imprimir um texto que informe ao usuário o grau correspondente ao coeficiente que ele deve digitar (por exemplo, “ x^3 .”, depois “ x^2 .”, etc.).

Caso o operador `>>` seja utilizado para ler um polinômio vazio (`empty`), a função deve emitir uma mensagem de erro e nada deve ser solicitado que seja digitado pelo usuário.

- Crie um método `salvar` para escrever um polinômio em arquivo, recebendo como parâmetro o nome do arquivo (`string`). A função retorna um booleano, indicando se a operação foi bem sucedida (`true`) ou não (`false`).

O formato do arquivo deve ser o seguinte:

- Primeira linha: palavra reservada `POLY` + espaço + dimensão (grau+1) do polinômio.
- Segunda linha: os coeficientes do polinômio, de a_0 até a_n , separados por espaço. Se a dimensão for nula, não há nenhum coeficiente.

- Crie um método `ler` para ler um polinômio de arquivo, recebendo como parâmetro o nome do arquivo (`string`). A função retorna um booleano, indicando se a operação foi bem sucedida (`true`) ou não (`false`).

O formato do arquivo é o mesmo definido para o método `salvar`, porém com bem mais flexibilidade, sendo válido qualquer formato do arquivo tal que as informações possam ser lidas por uma série de chamadas ao `operator>>`:

- Início: palavra reservada `POLY` seguida da dimensão do polinômio (não negativa), informações precedidas e/ou separadas por qualquer caractere separador (espaço, `ENTER`, `TAB`) em qualquer quantidade.
- Em seguida (na mesma linha ou em outra): os coeficientes do polinômio, de a_0 até a_n , precedidos e/ou separados por qualquer caractere separador (espaço, `ENTER`, `TAB`) em qualquer quantidade. Se a dimensão for nula, não há nenhum coeficiente.

Caso o arquivo não possa ser aberto ou não seja válido:

- Não inicia com a palavra reservada `POLY`;

- A dimensão do polinômio é inexistente ou inválida (<0);
- O número de coeficientes é menor do que a dimensão (não é um problema se for maior, ou seja, se houver conteúdo adicional além dos coeficientes necessários); ou
- O coeficiente de maior grau é 0.0 para um polinômio de grau >0 .

a função deve retornar `false` e **o polinômio original não deve ser alterado**.

Exemplos de arquivos que são válidos **em leitura** (só o primeiro é válido em escrita):

POLY 3 6.2 5.0 3.0	POLY 3 6.2 5 3 7 9
POLY 3 6.2 5 3.0 # TEXTO QUALQUER	POLY 3 6.2 5.0 3.0 7.2 9.5

Exemplos de arquivos inválidos:

Poly 3 6.2 5.0 3.0	POLY3 6.2 5.0 3.0
POLY 3 6.2 5.0	POLY 3 6.2 5.0 0.0

DICA: Não complique a implementação do método `ler`, pois o `operator>>` já lida com informações separadas por quantidades e tipos arbitrários de separadores (espaço, `ENTER`, `TAB`). Eventual conteúdo excedente (valores e/ou linhas a mais) é simplesmente ignorado. Não há necessidade de utilizar `getline`, `ws`, `ignore` nem nenhuma outra operação de leitura de dados diferente do `operator>>`.

DICA: Lembre-se que, caso o arquivo não seja válido, o conteúdo anterior do `Poly` não pode ser alterado. Por essa razão, você não pode armazenar diretamente os valores lidos da dimensão e dos coeficientes nos dados membro `D` e `a`, pois se posteriormente for verificado que o arquivo não é válido, os antigos valores terão sido sobrescritos e perdidos. A informação lida do arquivo deve ser salva em variáveis temporárias (por exemplo, `unsigned new_D` e `double* new_a`) e, só após confirmado que o arquivo está totalmente correto, transferida para os dados membros `D` e `a`, liberando a memória eventualmente alocada anteriormente.

- Sobrecarregue os operadores `+` e `-` binários para fazer a soma e a subtração de dois polinômios, retornando o polinômio resultado. Polinômios vazios (`empty`) e nulos (`isZero`), na adição e na subtração, devem ser equivalentes ao zero:

- $P + [\text{empty ou isZero}] \rightarrow P$
- $[\text{empty ou isZero}] + P \rightarrow P$
- $P - [\text{empty ou isZero}] \rightarrow P$
- $[\text{empty ou isZero}] - P \rightarrow -P$

DICA: Para calcular os coeficientes do polinômio resultante, que são a soma/subtração dos coeficientes correspondentes dos dois polinômios, **não é necessário nem eficiente** fazer testes para lidar com o fato dos polinômios poderem ter dimensões diferentes e que, por isso, alguns coeficientes podem não existir em um deles. Basta criar um novo polinômio cujo grau seja o maior (\max) dos graus dos polinômios sendo somados e utilizar `getCoef` ou `operator[]` (ver a dica na seção sobre essa função) para calcular todos os coeficientes do polinômio resultante:

```
Poly Poly::operator+(const Poly& P)
const {
    Poly prov( max(getGrau(),
                  P.getGrau()) );
    ...
    prov.a[i] = getCoef(i) +
               P.getCoef(i);
```

ATENÇÃO: Lembre-se que essas operações podem gerar polinômios de grau inferior ao esperado caso os coeficientes de maior grau resultantes sejam nulos. Por exemplo, a soma de dois polinômios de segundo grau pode resultar em um polinômio de grau zero:

$$(-3x^2 + 5x + 6) + (3x^2 - 5x - 1) = 5$$

Nesses casos, é necessário alterar (reduzir) a dimensão do polinômio resultante e **realocar** o array dinâmico `a` de coeficientes.

DICA: Faça a implementação inicialmente sem se preocupar com a possibilidade de gerar o coeficiente de maior grau nulo. Depois de calculado o polinômio resultante, enquanto o coeficiente de maior grau for nulo e o grau for maior que zero, aloque uma nova área de dimensão menor e copie os coeficientes não nulos para essa nova área, fazendo o ponteiro do `Poly` apontar para ela e liberando a área antiga (algoritmo clássico para fazer um array 1D alterar seu tamanho).

- Sobrecarregue o operador `-` unário para retornar o negativo de um polinômio. O negativo de um polinômio vazio (`empty`) ou nulo (`isZero`), é um polinômio do mesmo tipo:

```
o -empty → empty
o -isZero → isZero
```

- Sobrecarregue o operador `*` para retornar o produto de dois polinômios. Os polinômios vazios (`empty`) e nulos (`isZero`), na multiplicação, geram polinômios vazios e nulos:

```
o empty * [qualquer coisa] → empty
o [qualquer coisa] * empty → empty
o isZero * [qualquer, exceto empty] → isZero
o [qualquer, exceto empty] * isZero → isZero
```

ATENÇÃO: Normalmente, o grau do resultado do produto é a soma dos graus dos dois polinômios sendo multiplicados. Se você fizer previamente o tratamento dos casos dos polinômios vazios e nulos, não existe a possibilidade de gerar polinômios de grau inferior ao esperado. Se não fizer, **o que não é recomendado**, essa possibilidade existe. Por exemplo, o produto de um polinômio nulo (grau zero) por um polinômio de segundo grau resulta em um polinômio de grau zero, e não de grau dois:

$$(0) \times (3x^2 + 5x + 6.2) = 0$$

DICA: Os coeficientes do polinômio resultado do produto $A = B \times C$ são calculados da seguinte forma:

```
Para k de 0 a A.getGrau():
| a_k ← 0.0 // Inicializa com zeros
Fim Para
Para i de 0 a B.getGrau()
| Para j de 0 a C.getGrau()
| | a_{i+j} ← a_{i+j} + b_i x c_j
| Fim Para
Fim Para
```

OPCIONALMENTE (não faz parte da avaliação, fica apenas como estímulo ao aprendizado):

- Sobrecarregue o operador `/` para retornar o resultado (o quociente) da divisão de dois polinômios.
- Sobrecarregue o operador `%` para retornar o resto da divisão de dois polinômios.

A classe `Poly` deve permitir compilar e executar o programa principal de uma minicalculadora de polinômios que está disponível no SIGAA e que oferece as seguintes opções a cada iteração (os itens em azul são opcionais):

- Entrar um novo polinômio.
- Somar os polinômios.
- Subtrair os polinômios.
- Multiplicar os polinômios.
- Dividir os polinômios (retornar quociente)
- Dividir os polinômios (retornar resto)
- Calcular o último polinômio para um valor de x .
- Inverter o sinal do último polinômio.
- Trocar os polinômios.
- Terminar.

No início, o programa lê os arquivos denominados `poly_P1.txt`, `poly_P2.txt` e `poly_result.txt`, caso existam, para fixar valores iniciais dos polinômios. A cada operação, os polinômios são salvos nesses arquivos, de tal forma que, ao reiniciar o programa, ele continue do ponto onde parou.

ATENÇÃO: você não deve modificar o programa principal da minicalculadora que está no SIGAA. Sua classe deve executá-lo corretamente da forma como está disponibilizado.