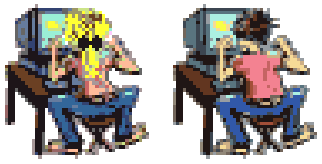
Terceiro Exercício-Programa

Calculadora de polinômios esparsos - Parte II

Entrega: 25 de outubro



O objetivo deste exercício-programa é exercitar o uso de pilhas e notação polonesa. Junto com o EP2, o resultado desse EP é uma calculadora que manipula polinômios esparsos.

Biblioteca de Polinomios Esparsos

No EP2, você implementou uma biblioteca para manipular polinômios esparsos que dá suporte às seguintes operações: soma, subtração, multiplicação, divisão, resto da divisão e o menos unário. Especificamente, você escreveu uma biblioteca que implementa o tipo abstrato de dados **Polinômios**. A interface da sua biblioteca é o arquivo polinomios.h, transcrito aqui:

```
/* ************************************************************** */
/*             Interface para o EP2: polinomios.h                 */
/* ************************************************************** */
/*             Nao faca nenhuma alteracao neste arquivo.          */
/* ************************************************************** */

typedef void *polinomio;

polinomio cria();

polinomio leia();

void      impr(char, polinomio);

polinomio copia(polinomio);

polinomio soma(polinomio, polinomio);

polinomio subt(polinomio, polinomio);

polinomio nega(polinomio);

polinomio mult(polinomio, polinomio);

polinomio quoc(polinomio, polinomio);

polinomio rest(polinomio, polinomio);

void      libera(polinomio);
```

Abaixo relembramos o que cada uma das funções da biblioteca **Polinomios** faz:

- `polinomio cria();`

  Devolve um polinômio nulo.

- `polinomio leia();`

  Lê uma sequência de pares de números, seguida por um zero. Cada par consiste de um número do tipo float não nulo, representando um coeficiente, e um inteiro não negativo, representando um expoente.

Devolve um novo polinômio com os termos dados pelos pares.

- `void impr(polinomio p);`

  Recebe um polinômio `p` e o imprime num formato adequado.

- `polinomio copia(polinomio p);`

  Devolve uma cópia do polinômio `p`.

- `polinomio soma(polinomio p, polinomio q);`

  Recebe dois polinômios, `p` e `q`, e devolve o polinômio que é a soma de `p` e `q`, ou seja, `p+q`.

- `polinomio subt(polinomio p, polinomio q);`

  Recebe dois polinômios, `p` e `q`, e devolve o polinômio que é a diferença de `p` e `q`, ou seja, `p-q`.

- `polinomio nega(polinomio p);`

  Recebe um polinômio `p`, e devolve o polinômio `-p`.

- `polinomio mult(polinomio p, polinomio q);`

  Recebe dois polinômios, `p` e `q`, e devolve o polinômio que é o produto de `p` e `q`, ou seja, `p*q`.

- `polinomio quoc(polinomio p, polinomio q);`

  Recebe dois polinômios, `p` e `q`, e devolve o polinômio que é o quociente da divisão de `p` por `q`.

- `polinomio rest(polinomio p, polinomio q);`

  Recebe dois polinômios, `p` e `q`, e devolve o polinômio que é o resto da divisão de `p` por `q`.

- `void libera(polinomio p);`

  Libera a memória ocupada pelo polinômio `p`.

Nenhuma das funções acima modifica algum polinômio recebido como parâmetro. Todos os polinômios devolvidos por essas funções são alocados dinamicamente e o cliente da biblioteca deve liberá-los mediante chamadas à função `libera` quando não tiverem mais utilidade.

Programa cliente: a calculadora

A sua missão no EP3 é implementar um programa que usará a biblioteca que você fez no EP2, ou seja, um programa cliente, que usará as funções da interface polinomio.h.

O programa cliente deve implementar uma calculadora que mantém um conjunto de 26 variáveis identificadas pelas letras minúsculas de `'a'` a `'z'`. Os valores dessas variáveis são polinômios. O usuário da calculadora pode efetuar operações envolvendo essas variáveis.

Este é um exemplo de uso da calculadora:

```
************************
Calculadora de polinomios
************************

Digite uma expressão ou quit para sair do programa:
> a?
a(x) = 0

Digite uma expressão ou quit para sair do programa:
> p?
p(x) = 0

Digite uma expressão ou quit para sair do programa:
```

```
> p: 4 5 -1 3 1 2 0
p(x) = 4x^5 - x^3 + x^2

Digite uma expressão ou quit para sair do programa:
> q: 2 2 3 4 1 3 -1 0 0
q(x) = 3x^4 + x^3 + 2x^2 - 1

Digite uma expressão ou quit para sair do programa:
> r=p+q
r(x) = 4x^5 + 3x^4 + 3x^2 - 1

Digite uma expressão ou quit para sair do programa:
> s=p*q
s(x) = 12x^9 + 4x^8 + 5x^7 + 2x^6 - 5x^5 + 2x^4 + x^3 - x^2

Digite uma expressão ou quit para sair do programa:
> s=s/p
s(x) = 3x^4 + x^3 + 2x^2 - 1

Digite uma expressão ou quit para sair do programa:
> y=~s
y(x) = -3x^4 - x^3 - 2x^2 + 1

Digite uma expressão ou quit para sair do programa:
> t=s-r
t(x) = -4x^5 + x^3 - x^2

Digite uma expressão ou quit para sair do programa:
> p=p*p
p(x) = 16x^10 - 8x^8 + 8x^7 + x^6 - 2x^5 + x^4

Digite uma expressão ou quit para sair do programa:
> p=p*p
p(x) = 256x^20 - 256x^18 + 256x^17 + 96x^16 - 192x^15 + 80x^14 + 48x^13 - 47x^12 + 12x^11
+ 6x^10 - 4x^9 + x^8

Digite uma expressão ou quit para sair do programa:
> a: 4 5 -1 3 1 2 0
a(x) = 4x^5 - x^3 + x^2

Digite uma expressão ou quit para sair do programa:
> b: 1 1 1 0 0
b(x) = x + 1

Digite uma expressão ou quit para sair do programa:
> u=(a+q)*b
u(x) = 4x^6 + 7x^5 + 3x^4 + 3x^3 + 3x^2 - x - 1

Digite uma expressão ou quit para sair do programa:
> v=(a+q)/b
v(x) = 4x^4 - x^3 + x^2 + 2x - 2

Digite uma expressão ou quit para sair do programa:
> w=(a+q)%b
w(x) = 1

Digite uma expressão ou quit para sair do programa:
> b: -1 2 9 0 0
b(x) = -x^2 + 9

Digite uma expressão ou quit para sair do programa:
> w=(a+q)%b
w(x) = 324x + 269
```

```
Digite uma expressão ou quit para sair do programa:
> u=u-(a+q)%b
u(x) = 4x^6 + 7x^5 + 3x^4 + 3x^3 + 3x^2 - 325x - 270

Digite uma expressão ou quit para sair do programa:
> quit
Tchau!
```

As linhas em negrito foram digitadas pelo usuário da calculadora. As demais linhas são geradas pela calculadora e formam a saída do programa.

Cada linha da entrada contém um comando para a calculadora. Todo comando começa com uma letra minúscula de 'a' a 'z'. Essa letra identifica uma variável da calculadora. O comando "v?" (uma variável da calculadora seguida do caractere "?") é um comando de consulta que imprime o polinômio associado à variável especificada. No início da sua execução, a calculadora inicializa com polinômios nulos todas as variáveis de 'a' a 'z'. (Seu programa deve fazer isso!) Por esse motivo, a consulta "a?" acima imprime a informação "a(x) = 0" (a é um polinômio nulo). A primeira das consultas "p?" imprime informação análoga.

Um comando da forma "v: <lista de pares de números seguida por zero>" atribui à variável v o polinômio cujos termos não nulos são especificados por uma lista de pares de números seguida de 0. Em cada um desses pares de números, o primeiro elemento é o coeficiente do termo (um número real não nulo) e o segundo elemento é o expoente do termo (um número inteiro não negativo). Assim, o comando "p: 1 2 -1 3 4 5 0" atribui à variável p um polinômio com três termos não nulos (o polinômio $x^2 - x^3 + 4x^5$). Em resposta a esse comando, a calculadora imprime o polinômio atribuído à variável p (a informação p(x) = 4x^5 - x^3 + x^2). A lista de pares não contém termos com expoentes repetidos, porém não precisa ser fornecida em ordem crescente ou decrescente de expoente. No comando "q: 3 4 -1 0 1 3 2 2 0" os pares são fornecidos numa ordem arbitrária, mas na impressão os termos do polinômio atribuído à variável q podem aparecer, por exemplo, em ordem decrescente de expoente (q(x) = 3x^4 + x^3 + 2x^2 - 1).

Um comando da forma "v=s+t", ou da forma "v=s-t", ou da forma "v=s*t", ou da forma "v=s/t", ou da forma "v=s%t" atribui à variável v o polinômio resultado de uma operação (soma, subtração, multiplicação, quociente ou resto) aplicada aos polinômios associados às variáveis s e t. Similarmente, um comando da forma "v=~s" atribui à variável v o polinômio resultado da negação do polinômio associado à variável s. Em resposta a comandos assim, a calculadora imprime o polinômio atribuído à variável v. O exemplo acima contém vários comandos desse tipo. Note que a mesma variável pode aparecer nos dois lados da atribuição ("p=p*p").

Na verdade, a nossa calculadora aceita expressões arbitrárias (válidas) com os operadores acima, conforme mostramos no exemplo.

Implementação do Programa Cliente

Note que o segundo caractere de cada comando ('?', ':', ou '=') define o tipo do comando. O primeiro caractere especifica a variável afetada pelo comando. Você pode usar esses fatos para simplificar a implementação do seu programa cliente.

Para implementar as variáveis da calculadora, utilize um vetor com 26 posições (uma para cada letra minúscula de 'a' a 'z'), que contenha em cada posição uma referência para um polinômio.

Mais especificamente, seu programa cliente deve manter um vetor

```
    polinomio var[26]; /* O tipo polinomio é o definido pela biblioteca. */
```

de modo que

- var[0] seja o polinomio associado à variável cujo identificador é a,
- var[1] seja o polinomio associado à variável cujo identificador é b,
- var[2] seja o polinomio associado à variável cujo identificador é c, etc.

Em outras palavras, as 26 variáveis da calculadora (a, b, c, ..., z) são implementadas pelas 26 posições do vetor var (var[0], var[1], var[2],..., var[25]). Dessa forma pode-se

ter acesso a um polinômio de modo bem eficiente, a partir da letra minúscula que identifica o polinômio.

No início da execução do programa, todas as posições do vetor var devem ser inicializadas com polinômios nulos, por meio da função cria.

Tome o seguinte cuidado com a leitura dos dados no seu programa cliente: leia cada linha da entrada até o último caractere da linha (o caractere '\n' que termina toda linha), incluindo esse caractere. Se você não fizer isso, quando você quiser ler o primeiro caractere da próxima linha você acabará pegando o '\n' da linha anterior (ou até mesmo algum caractere que apareceu antes desse '\n', como por exemplo um espaço presente ao final da linha anterior). Para não ter esse tipo de problema, faça seu cliente ler toda linha até o fim, passando para a linha seguinte só depois de ler um '\n'! Para tanto, você pode usar a seguinte função:

```
void descarta_resto_da_linha() {
  char c;

  do {
    scanf("%c", &c);
  } while (c != '\n' && c != EOF);
}
```

Faça o seu programa cliente chamar descarta_resto_da_linha no tratamento de todo comando da calculadora: depois de ler o último caractere útil de um comando, descarte o resto da linha da entrada.

Recomendações

Observe que sua calculadora terá que ser capaz de processar expressões infixas. Para tanto, inspire-se no material abordado na aula 11. Pode ser uma boa ideia primeiro converter a expressão dada para notação posfixa, e depois processar a notação posfixa. Isso facilitará o desenvolvimento do seu programa, pois você poderá testar a conversão para posfixa separadamente do cálculo da expressão.

Note que no EP3 você não vai precisar usar listas encadeadas! As listas encadeadas estão restritas à implementação da biblioteca, que foi o que você fez no EP2.

Modifique o Makefile usado no EP2 substituindo todas as ocorrências de testaEP2 por ep3, e escreva o seu programa no arquivo ep3.c.

Siga as instruções das informações gerais sobre entrega dos EPs, e consulte a página do Prof. Paulo Feofiloff sobre documentação de programas.

Divirta-se com o EP!
