Exercício Programa 1 - MAC0122 Princípios de Desenvolvimento de Algoritmos - POLI

Prof. Ronaldo Fumio Hashimoto

1 Descrição do Exercício Programa

Neste Exercício Programa (EP), você deverá implementar um pacote de aritmética racional e um programa cliente que usa esse pacote (*package*) para computar a aproximação racional do e (número de Euler) e de raízes quadradas. A proposta do EP é o aprendizado dos conceitos de *definições de tipos*, *empacotamento de interfaces*, *implementação*, *compilação separada* e *transbordamento aritmético* (*arithmetic overflow*). Este exercício é fortemente baseado no *Programming Assignment - Rational Arithmetic* [1].

2 Aritmética Racional

Números racionais são números que podem ser representados como a razão de dois inteiros, i.e. qualquer número p/q onde p e q são inteiros é um número racional. A linguagem C aproxima números não-inteiros racionais usando números de ponto-flutuante (*floats* e *doubles*), mas esses tipos são representações imprecisas. Para esse EP, você representará números racionais com uma estrutura com dois inteiros:

```
struct { unsigned long num; unsigned long den; }
```

Essa estrutura representa um número racional com um numerador `num` e denominador `den`. Quando um tipo da linguagem C é usado repetidamente para representar um outro tipo, mais abstrato, é convencional especificar um novo nome para esse tipo por meio de uma definição de tipo conforme mostrado abaixo:

```
typedef struct {
   unsigned long num;
   unsigned long den;
} Rational;
```

Essa diretiva nos permite usar números racionais em programas em C de maneira semelhante a outros tipos, conforme exibido no código de exemplo abaixo:

```
Rational a, b, c;
a = RATinit(1,3);
b = RATinit(1,4);
c = RATadd(a, b);
RATshow(c);
```

Neste EP, você irá aprender a importância do método geral para implementação e uso de novos tipos de dados e suas funções associadas. Para tanto, primeiramente crie o arquivo `RAT.h` com o seguinte código:

```
typedef struct {
   unsigned long num;
   unsigned long den;
} Rational;

Rational RATinit(unsigned long numerator,
                    unsigned long denominator);
void RATshow(Rational r);
Rational RATadd(Rational r1, Rational r2);
Rational RATmul(Rational r1, Rational r2);
Rational RATdiv(Rational r1, Rational r2);
```

Este arquivo é chamado de *interface*. Sua proposta é especificar precisamente qual será o tipo de dado usado (os possíveis tipos de variáveis e as funções que as manipulam). Neste caso, o tipo de dado `Rational` é um par de inteiros, e temos funções para: inicializá-lo; mostrá-lo (imprimí-lo); adicionar dois deles e colocar o resultado em um terceiro `Rational`; multiplicar dois deles, colocando o resultado em um terceiro; e dividir um número racional por outro, armeazenando o resultado em um terceiro.

3 Passos a seguir para a resolução do EP

Passo 1

Crie um arquivo `rat.c` e escreva a implementação para cada função. A primeira linha deste arquivo deve incluir a *interface* como mostrado abaixo:

```
#include "RAT.h"
```

Esta linha de código inclui as declarações de tipos de dados e suas operações (funções) disponibilizadas pelo arquivo `RAT.h`. Permitindo assim que suas implementações tenham acesso a todas as funções a serem implementadas independente da ordem que você as desenvolvam. Este arquivo é chamado de *implementação* do tipo de dado.

Passo 2

Escreva um programa `e.c` que usa as implementações das funções do `Rational` para calcular uma aproximação racional para e (número de Euler), usando a expansão da série de Taylor

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \dots$$

.

Leia um inteiro n da entrada padrão usando `scanf()` e imprima as n primeiras aproximações. Por exemplo, a saída para $n = 6$ é:

```
1/1  2/1  5/2  32/12  780/288  93888/34560
```

No contexto atual, estamos interessados neste programa como um exemplo de *cliente*: um programa que usa o tipo de dado, mas é implementado separadamente. Para usar o tipo `Rational` no arquivo `e.c`, antes de seu código adicione

```
#include "RAT.h"
```

Agora, você pode usar variáveis do tipo `Rational` no seu programa cliente, mas **certifique-se de manipular seu tipo de dado apenas através das funções definidas na interface**. Lembre-se de adicionar o arquivo `rat.c` em seu processo de compilação.

Passo 3

Uma implementação ingênua da *interface* de aritmética de racionais possui um problema chamado de transbordamento aritmético (*arithmetic overflow*) que ocorre quando uma operação aritmética de inteiros produz um número que necessita de mais bits (i.e. um número com muitos dígitos) para ser representado do que o computador está utilizando para armazená-lo . É impossível evitar o transbordamento aritmético, mas é possível reduzir sua ocorrência. Por exemplo, no quarto termo do exemplo acima, tanto o numerador quanto o denominador são divisíveis por 4, então podemos simplificar o termo $\frac{32}{12}$ para $\frac{8}{3}$, que possui números que necessitam de menos bits para serem representados.

Desenvolva uma implementação melhorada `ratbetter.c` que minimiza transbordamento (*overflow*) aritmético através do algoritmo de Euclides [2], usando-o para modificar sua implementação para aderir à convenção que funções somente retornam números racionais cujos numeradores e denominadores são relativamente primos.

Obs: mesmo com a simplificação das frações, o problema de transbordamento aritmético ainda pode ocorrer mesmo em casos que as frações simplificadas podem ser armazenadas. Isso ocorre porque tanto as somas quanto os produtos obtidos pelas funções de aritmética racional podem transbordar, gerando resultados errados. Por exemplo, $\frac{a}{b} \times \frac{c}{d}$ é igual ao valor reduzido de $\frac{ac}{bd}$,

mas tanto ac quanto bd podem transbordar antes da redução, mesmo que a resposta reduzida possa ser representada sem problemas (por exemplo, o método direto de calcular $\frac{2^{40}}{3^{25}} \times \frac{3^{26}}{2^{41}} = \frac{3}{2}$ transbordaria, pois as duas frações sendo multiplicadas já estão em sua forma simplificada e os valores de $2^{40} \times 3^{26}$ e $3^{25} \times 2^{41}$ ultrapassam o valores máximo do tipo `unsigned long`). Similarmente, $\frac{a}{b} + \frac{c}{d}$ é igual ao valor reduzido de $\frac{(ad+bc)}{bd}$, mas tanto $ad + bc$ quanto bd podem transbordar antes da redução, mesmo que a resposta reduzida possa ser armazenada sem problemas. É possível otimizar as operações para contornar este tipo de problema mas, para este EP, tal otimização pode ser ignorada.

Passo 4

Escreva um programa cliente `sqrt.c` que calcule uma aproximação racional da raiz quadrada de um número S utilizando o metódo babilônico [3] conforme mostrado abaixo:

$$\sqrt{S} \approx \begin{cases} x_n = \frac{1}{2}\left(x_{n-1} + \frac{S}{x_{n-1}}\right), & \text{se } n > 0 \\ x_0, & \text{se } n = 0 \end{cases}$$

Seu programa deve ler um valor inteiro $S \geq 0$, um valor inteiro $n \geq 0$ e um "chute inicial" racional $x_0 \geq 0$ para a aproximação e calcular o valor racional aproximado de $\sqrt{S}$ utilizando o metódo acima. Por exemplo, para $S = 2$, $n = 4$ e $x_0 = \frac{1}{1}$, sua saída deve ser:

```
665857/470832
```

4 Compilação e Submissão

Use os seguintes parâmetros para compilar seu código:

```
-Wall -ansi -pedantic -O2
```

Há um link no PACA [4] explicando a utilidade destes parâmetros e como configurar o *codeblocks* para incluir estas opções na hora de compilar seu programa. Procure entregar seu exercício programa sem mensagens de alerta do compilador (*warnings*).

Para submeter seu EP, envie um arquivo compactado contendo os arquivos: `readme.txt`, `rat.c`, `ratbetter.c`, `e.c` e `sqrt.c`. Se você utilizou o *codeblocks* envie os arquivos de projetos também (".cbp"). Você **não** deve alterar o arquivo `RAT.h`. Seus programas clientes `e.c` e `sqrt.c` devem funcionar com as duas implementações `rat.c` e `ratbetter.c`. É esperado que seus programas clientes obtenham respostas mais precisas com a utilização da implementação `ratbetter.c`.

O arquivo de `readme.txt` deve ter instruções de compilação e de uso dos seus programas desenvolvidos, além de conter explicações das melhorias obtidas comparando o `rat.c` com o `ratbetter.c`.

Referências

[1] R. Sedgewick, "Programming Assignments - Rational Arithmetic." http://www.cs.princeton.edu/courses/archive/spr03/cs126/assignments/rat.html, 1999. [Online; acessado dia 21 de março de 2017].

[2] Wikipedia, "Euclidean algorithm ." https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations. [Online; acessado dia 29 de março de 2017].

[3] "Methods of computing square roots." https://en.wikipedia.org/wiki/Methods_of_computing_square_roots. [Online; acessado dia 21 de março de 2017].

[4] "Perguntas e Respostas mais Frequentes sobre Compilação ." http://vision.ime.usp.br/~ronaldo/mac0122-2017/compilacao/. [Online; acessado dia 29 de março de 2017].