MAC2166 – Introdução à Ciência da Computação

Escola Politécnica – Computação / Elétrica – Primeiro Semestre de 2023

Exercício-Programa 2 (EP2)
Data de Entrega: **28 de maio de 2023**

Para se preparar bem para o desenvolvimento de seu EP2, cuja descrição inicia-se na próxima página, leia com atenção as instruções abaixo.

- Utilize **somente** os recursos da linguagem que aprendeu nas aulas.
- Veja em `https://www.ime.usp.br/~mac2166/infoepsC/` as instruções de entrega dos exercícios-programa e atente para as instruções de preenchimento do cabeçalho do seu programa.
- Caso você tenha dúvidas sobre eventuais erros e *warnings* que o compilador produza ao processar o seu programa, consulte o FAQ sobre compilação em `https://www.ime.usp.br/~mac2166/compilacao/`.
- Sempre compile seus programas com as opções **-Wall -ansi -pedantic -O2**. Seu programa deve:
  - funcionar para qualquer entrada que está de acordo com o enunciado (não é necessário verificar se a entrada está "bem formada");
  - estar em conformidade com o enunciado;
  - estar bem estruturado;
  - ser de fácil compreensão, com o uso padronizado da linguagem C.

EP2: π?!

1 Introdução

Inicialmente, assista ao vídeo

▷ *How many collisions*?, acessível em `https://youtu.be/HEfHFsfGXjs`.

Para este EP, a parte essencial desse vídeo são seus primeiros três minutos.[1]

Neste EP você irá escrever um programa para simular um sistema de blocos que colidem de forma elástica, como no vídeo acima, mas sem a parte gráfica. Com seu programa, você será capaz de reproduzir o fenômeno curiosíssimo discutido no vídeo acima.

2 Seu programa

Seu programa simulará um sistema com **quatro** blocos, todos eles de dimensão $L \times L \times L$, onde $L = 0.1$ (neste enunciado, omitimos as unidades das quantidades físicas, e supomos que todas essas quantidades são dadas nas unidades do Sistema Internacional de Unidades; por exemplo, $L = 0.1$ m). Enquanto que todos os blocos têm a mesma dimensão, cada bloco poderá ter massa diferente, a ser especificada pelo usuário do programa.[2]

Os quatro blocos serão 'rotulados' com os números 0, 1, 2 e 3. Para quaisquer blocos i e j com $0 \leq i < j < 4$, supomos que o bloco i está à esquerda do bloco j. Denotaremos por x_i a posição do centro do bloco i, por v_i sua velocidade e por m_i sua massa. O bloco i se movimenta para ∞ quando $v_i > 0$ e se movimenta para $-\infty$ quando $v_i < 0$.

Os valores iniciais da posição x_i, da velocidade v_i e da massa m_i ($0 \leq i < 4$) são especificados, nessa ordem, pelo usuário do programa. Eis um exemplo de entrada para seu programa:

```
0.0  1.0  1.0
1.0  0.0  1.0
2.0  0.0  1.0
3.0  0.0  1.0
3.6
```

A linha `0.0 1.0 1.0` especifica que, no instante $T = 0$ da simulação, o corpo 0 está na posição 0 com velocidade 1 e tem massa 1 (isto é, $x_0 = 0$, $v_0 = 1$ e $m_0 = 0$). Para o corpo 1, temos $x_1 = 1$, $v_1 = 0$ e $m_1 = 1$. As próximas duas linhas especificam a situação inicial dos corpos 2 e 3 de forma similar. O número 3.6 especifica que a simulação deve ser terminada no instante $T = 3.6$.

Com a entrada acima, seu programa deve produzir a seguinte saída:

[1] Note que legendas em português estão disponíveis para esse vídeo.

[2] Supomos também que todos os blocos estão em posição 'ortogonal', isto é, estão com suas faces paralelas aos planos xy, yz e xz. Supomos também que o eixo x passa pelo centro de todos os blocos (isto é, todos os blocos estão 'alinhados'). O sistema que estamos simulando aqui é essencialmente unidimensional.

```
Numero de colisoes por bloco: 1 1 1
Numero total de colisoes: 3
Colisoes dos dois blocos a direita: 2
x0 =  0.900000  v0 =  0.000000  d0 =  0.900000
x1 =  1.900000  v1 =  0.000000  d1 =  0.900000
x2 =  2.900000  v2 =  0.000000  d2 =  0.900000
x3 =  3.900000  v3 =  1.000000  d3 =  0.900000
Nao ha mais colisoes
```

A saída acima deve ser interpretada da seguinte forma: na simulação executada, cada um dos blocos 1, 2 e 3 colidiu exatamente uma vez com o bloco imediatamente à sua esquerda; ocorreram no total 3 colisões durante a simulação; houve um total de 2 colisões que envolveram os blocos 2 e 3. Os valores dos x_i e v_i são aqueles no instante final da simulação (isto é, no instante $T = 3.6$). A saída também inclui qual foi o deslocamento total de cada bloco (esses valores são dados como d_i $(0 \leq i < 4)$). No exemplo acima, todos os blocos deslocaram-se 0.9. Finalmente, como especificado na entrada, a simulação foi interrompida depois de 3.6 segundos, e a mensagem `Nao ha mais colisoes` significa que nesse sistema não haveria colisões adicionais, mesmo que deixássemos a simulação continuar.

A entrada e a saída acima estão nos arquivos `simple0.txt` e `simple0_out.txt`. Para entender bem o comportamento esperado de seu programa, estude cuidadosamente os pares de entrada e saída `simple1.txt`/`simple1_out.txt` e `simple2.txt`/`simple2_out.txt`.[3]

Observações

(i) Os três números na primeira linha da saída do programa indicam quantas colisões houve entre o bloco 1 e o bloco 0, entre o bloco 2 e o bloco 1, e entre o bloco 3 e o bloco 2. Em particular, se esses números são c_1, c_2 e c_3, o número na segunda linha da saída é $c_2 + c_3$.[4]

(ii) Note que em `simple1_out.txt`, vemos que o bloco 1 termina em sua posição inicial (o mesmo vale para o bloco 2). Entretanto, o deslocamento do bloco 1 é 1.9. Isto ocorre porque queremos na saída o *deslocamento total* que o bloco sofreu na simulação.

(iii) Neste ponto de sua leitura, você deve estar completamente satisfeito com os três pares de entrada e saída `simple*.txt`/`simple*_out.txt`.

3 Como obter π

Para obter os primeiros dígitos de $\pi = 3.14159265359\ldots$ como o número de colisões entre blocos, você poderá executar seu programa com as entradas `input*.txt` fornecidas. Por exemplo, com entrada `input04.txt`, sua saída deve ser:

[3] Estude os pares `newton*.txt`/`newton*_out.txt` apenas depois de ter lido a Seção 4 abaixo.
[4] Nesse momento não é claro por que estamos interessados em $c_2 + c_3$.

```
Numero de colisoes por bloco: 15708 15707 15708
Numero total de colisoes: 47123
Colisoes dos dois blocos a direita: 31415
x0 = -8.300000  v0 = -1.000000  d0 =  9.999810
x1 = -1.597879  v1 = -0.073359  d1 = 19.961309
x2 =  1.597879  v2 =  0.073359  d2 = 19.961309
x3 =  8.300000  v3 =  1.000000  d3 =  9.999810
Nao ha mais colisoes
```

Note que `31415` ocorre como o número de colisões que envolvem os blocos 2 e 3 (com a notação usada anteriormente, $c_2 + c_3 = 31415$). Com entrada `input09.txt`, sua saída deve ser:

```
Numero de colisoes por bloco: 1570796327 1570796326 1570796327
Numero total de colisoes: 4712388980
Colisoes dos dois blocos a direita: 3141592653
x0 = -8.300000  v0 = -1.000000  d0 = 10.000000
x1 = -4.313875  v1 = -0.408976  d1 = 49.761413
x2 =  4.313875  v2 =  0.408976  d2 = 49.761413
x3 =  8.300000  v3 =  1.000000  d3 = 10.000000
Nao ha mais colisoes
```

Observação. Note que o inteiro 3141592653 que ocorre na saída acima não 'cabe' em uma variável do tipo `int`. Use variáveis do tipo `long` para contar colisões. (Veja o programa `long_int.c` disponibilizado.)

4 Pêndulo de Newton

Estude agora as entradas `newton*.txt`. Tente prever o que ocorre com os sistemas especificados nesses arquivos de entrada.[5] Você pode achar interessante lembrar-se do pêndulo de Newton:

▷ `https://youtu.be/09vzFodz4SI`

Quando você tiver seu programa pronto, você poderá comparar sua previsão com o resultado da execução de seu programa com as entradas `newton*.txt`.

5 A física envolvida

Descrevemos aqui como devemos simular a colisão de dois de nossos blocos, digamos b e b'. Lembre que nossos blocos tem dimensão $L \times L \times L$. Suponha que b tenha seu centro em x e b' tenha seu centro em x'. Sejam v e v' as velocidades de b e b', respectivamente. Suponha que $v' - v < 0$ e analisemos o momento em que os blocos b e b' irão colidir, i.e., $x + L = x'$. Lembrando que nossas colisões são elásticas, podemos usar a conservação do momento e da energia cinética

[5] Onde termina cada bloco? Quais blocos terminam estáticos? Quanto desloca-se cada bloco?

para determinar a velocidade de b e b' imediatamente após a colisão. Suponha que b tenha massa m e que b' tenha massa m'. Sejam w e w' as velocidades de b e b' imediatamente após a colisão, respectivamente. Vale que

$$w = A - v, \tag{1}$$

$$w' = A - v', \tag{2}$$

onde

$$A = \frac{2(mv + m'v')}{m + m'}. \tag{3}$$

A derivação de (1)–(3) fica como um excelente exercício `:-)`.

6 Alguns detalhes de implementação

6.1 Simulação orientada a eventos

Seu programa deverá executar a simulação considerando os *eventos de interesse*. Aqui, os eventos de interesse são as *colisões entre os blocos* e a *chegada do momento de término da simulação* (lembre que a simulação deve ocorrer do instante $T = 0$ até o instante $T = T_{\max}$, onde $T_{\max}$ é dado pelo usuário em sua entrada). Grosseiramente falando, o esqueleto de seu programa deve ser algo como segue:

```
<leia dados>;
T = 0.0;
while (T < T_max) {
  <determine o próximo evento E que ocorrerá>;
  t = intervalo de tempo até o evento E;
  <avance o sistema até o momento T + t>;
  <execute o evento E>;
}
<imprima a saída>;
```

No esqueleto acima, "executar o evento E" basicamente significa determinar as novas velocidades dos blocos b e b' envolvidos na colisão representada pelo evento E. Note que "avançar o sistema até o momento $T + t$" é muito simples: entre o instante atual T e o instante $T + t$ não ocorre nenhum evento, isto é, não ocorre nenhuma colisão entre blocos, de forma que é muito simples determinar onde cada bloco estará no instante $T + t$.

Algumas funções que serão úteis para implementar o algoritmo acima de simulação orientado a eventos são detalhados a seguir.

6.2 Detalhes de implementação

Você deve, **obrigatoriamente**, implementar e usar em seu programa as funções descritas a seguir. Para cada função, é dado seu protótipo e uma descrição de seu comportamento.

Leitura da entrada. Implemente e use a função de protótipo

```
void read_data(double *x0, double *x1, double *x2, double *x3,
               double *v0, double *v1, double *v2, double *v3,
               double *m0, double *m1, double *m2, double *m3,
               double *T_max);
```

para a leitura dos dados fornecidos pelo usuário.

Tempo até a próxima colisão entre dois blocos. Suponha que em um dado instante T os blocos b e b' estejam nas posições x e x' e que eles estejam com velocidades v e v', respectivamente. A chamada $t(x, x', v, v')$ da função de protótipo

```
double t(double x, double xx, double v, double vv);
```

deve ter valor igual a ΔT, onde ΔT é o intervalo de tempo entre o instante atual T e o momento em que a colisão entre b e b' ocorrerá, *supondo que não há outros blocos no sistema.*

Importante. Pode ocorrer de b e b' estarem em uma situação no instante T que faz com que eles não colidam em nenhum momento após T. Nesse caso, $t(x, x', v, v')$ deve valer `HUGE_VAL`, uma constante definida em `math.h` que pode ser usada para representar ∞. (Veja o programa `huge_val.c` disponibilizado.)

Mínimo entre três valores. A função de protótipo

```
double min(double a, double b, double c, int *i);
```

deve devolver o menor valor m entre a, b e c. Ademais, ela deve colocar o valor 1 em `*i` se $a = m$. Se $a \neq m$, ela deve colocar o valor 2 em `*i` se $b = m$. Se $a \neq m$ e $b \neq m$, ela deve colocar o valor 3 em `*i`.

Movimento uniforme. A função de protótipo

```
double adv(double *x, double v, double t);
```

deve receber a posição `*x` e a velocidade `v` de um bloco e um intervalo de tempo `t`, deve atualizar o valor de `*x` para a posição desse bloco após o tempo `t`, supondo que o bloco executou um movimento retilíneo uniforme com velocidade `v`. Além disso, essa função deve devolver o deslocamento que o bloco sofreu durante esse movimento, em valor absoluto.

Execução de uma colisão. Suponha que dois blocos b e b' estão em um instante imediatamente anterior a uma colisão. Suponha que eles tenham velocidades `*v` e `*vv` e que tenham massas `m` e `mm`. A chamada da função de protótipo

```
void resolve(double *v, double *vv, double m, double mm);
```

deve atualizar os valores de `*v` e `*vv` para as velocidades de b e b' imediatamente após a colisão (veja Seção 5).

Impressão da saída. Implemente e use a função de protótipo

```
void print_data(long c1, long c2, long c3,
                double x0, double x1, double x2, double x3,
                double v0, double v1, double v2, double v3,
                double d0, double d1, double d2, double d3);
```

para a impressão da situação do sistema no instante atual. Durante a depuração do seu programa, esta função pode ser usada dentro do laço da simulação para ajudar a identificar eventuais erros.

7 Testes para seu programa

No e-Disciplinas disponibilizamos vários arquivos adicionais de entrada e também as saídas correspondentes a cada um deles. Para testar seu programa com o arquivo de entrada `entrada.txt`, compile seu programa e produza um arquivo de saída como abaixo:

```
$ gcc ep2.c -Wall -ansi -pedantic -O2 -o ep2
$ ./ep2 < entrada.txt > saida_minha.txt
```

Compare a saída produzida pelo seu programa (`saida_minha.txt`) com o arquivo de saída disponibilizado para a entrada correspondente. Espera-se que os números de colisões em seu arquivo `saida_minha.txt` sejam iguais aos números correspondentes nas saídas disponibilizadas. Os x_i, v_i e d_i podem diferir, no caso de simulações que envolvam um grande número de colisões.