Geração de números aleatórios

Números aleatórios são muito importantes em várias áreas da informática, como jogos, simulações e na geração de chaves criptográficas. Um gerador de **números pseudoaleatórios** (PRNG - Pseudo Random Number Generator) é um algoritmo que gera uma sequência de valores aparentemente aleatórios. A sequência de números gerada por um PRNG não é realmente aleatória, porque pode ser reproduzida caso seja usado o mesmo valor inicial, usualmente chamado de "semente" (*seed*).[1)]

Congruência linear

Uma das abordagens mais simples para a geração de números pseudoaleatórios são os **geradores congruentes lineares** (LCG), que são definidos através da seguinte fórmula:

> $x_{i+1} = (A \cdot x_i + C) \bmod M$

onde "x_0" é o valor inicial (semente), A e C são parâmetros inteiros, M é o tamanho do espaço de valores aleatórios possíveis e "mod" é o resto da divisão inteira.

Por exemplo, escolhendo-se A=40.692, C=127, M=10.000.000 e x_0=0, será gerada a seguinte sequência de números pseudoaleatórios:

> $x_0 = 0$
> $x_1 = 127$
> $x_2 = 5168011$
> $x_3 = 6703739$
> $x_4 = 8547515$
> $x_5 = 5480507$
> $x_6 = 2790971$
> $x_7 = 192059$
> $x_8 = 5264955$
> $x_9 = 1548987$
> ...

Os geradores congruentes lineares são considerados fracos, por serem facilmente previsíveis. Por isso, **não devem ser usados** em aplicações críticas, como a criptografia.

Atividade

a) Implementação da biblioteca

Construa uma biblioteca *LC-Random* para gerar números inteiros pseudoaleatórios usando o algoritmo de congruência linear; a interface dessa biblioteca é definida pelo seguinte arquivo de cabeçalho (que **não deve ser alterado**):

lcrandom.h

```c
// ATENCAO: ESTE ARQUIVO NAO DEVE SER ALTERADO!

#ifndef __LCRANDOM__
#define __LCRANDOM__

// calcula e devolve um valor pseudoaleatório
unsigned long lcrandom () ;

// define o valor inicial (semente) da sequência de aleatórios
// (inicialmente zero (0) por default
void lcrandom_seed (unsigned long s) ;

// informa o valor máximo que pode ser gerado (o mínimo é sempre zero)
unsigned long lcrandom_max () ;

// modifica os parâmetros da equação do gerador; por default
// devem ser usados: A=40.692, C=127 e M=10.000.000
void lcrandom_parms (unsigned long A, unsigned long C, unsigned long M) ;

#endif
```

Caso a função `lcrandom_parms` não seja invocada, a implementação deve usar os valores *default* dos parâmetros A, C e M definidos no exemplo acima.

Dicas para a implementação:

- Os valores da semente, de x e dos parâmetros A, C e M são parte do **estado interno** da biblioteca, por isso devem ser implementados como variáveis globais dentro do arquivo `lcrandom.c`.
- A biblioteca apenas calcula, atualiza e devolve valores; por isso, **nenhuma função de entrada/saída** (`printf`, `scanf`, etc) deve ser chamada dentro da implementação de suas funções (a não ser para produzir eventuais mensagens de erro).

b) Geração de histograma

A distribuição dos valores produzidos por um gerador de números aleatórios pode ser avaliada visualmente através de um histograma[2].

Escreva um programa C para gerar um histograma dos valores aleatórios produzidos por sua biblioteca.

Para isso:

1. gere 10^6 valores aleatórios no intervalo [0...99] (use `lcrandom() % 100`);
2. conte quantas vezes cada número foi gerado;
3. normalize (i.e. ajuste) as contagens para intervalo [0..100], aplicando uma regra de três às contagens obtidas, na qual a maior contagem corresponde a 100;
4. plote o histograma resultante em um gráfico em modo texto; no eixo vertical estão os valores possível (0 a 99, neste exemplo); no eixo horizontal estão as frequências de ocorrência normalizadas.

Exemplo simplificado de histograma em modo texto (cada * vale 2 unidades):

```
  0   10   20   30   40   50   60   70   80   90   100
```

```
   +----+----+----+----+----+----+----+----+----+----+
 0 |****************************
 1 |************************************************
 2 |****************************************
 3 |*****************************************
 4 |*********************************
 5 |******************************
 6 |***************************
 7 |*******************************
...| (linhas omitidas)
98 |**********************************
99 |************************************************
   +----+----+----+----+----+----+----+----+----+----+
```

c) Cálculo de período

O **período** de um PRNG corresponde à quantidade de valores gerados por ele antes de começar a repeti-los, para uma dada semente. Por exemplo, para uma semente x_0=0, um PRNG gera a seguinte sequência, cujo período é 16 (pois são gerados 16 valores distintos antes de repetir o 78):

> 0 2233 491 11 7219 **7801** 31 446 9876 831 6159 43259 63 38321 1126 43674 **7801** 31 446 9876 831 ...

Escreva um programa C para calcular o período do gerador implementado em sua biblioteca, a partir de x_0=0.

Para os valores default de (A=40.692, C=127 e M=10.000.000), o período do gerador vale 62.503.

d) Tudo de novo, com novos parâmetros

Repita o cálculo do histograma e do período usando valores melhores para os parâmetros A, C e M do gerador: A = 1.103.515.245, C = 12.345 e ~~M = 2.147.483.648 (2^{31})~~ M = 1.073.741.824 (2^{30}) - valores inspirados de LCG.

Para esses valores de A, C e M, o período do gerador deve ser igual a M.

Caso encontre erros do tipo `relocation truncated to fit: R_X86_64_PC32 against symbol …` ao compilar seu código, experimente usar as opções `-fPIC` ou `-mcmodel=medium` na linha de compilação. Esses erros podem ocorrer no cálculo do período, caso um vetor muito grande.

Entregáveis

Arquivos a entregar ao professor:

1. `lcrandom.c` : implementação das funções do gerador de aleatórios;
2. `histograma-1.c` : implementação do histograma do gerador com os valores default de A, C e M;
3. `histograma-2.c` : idem, com os valores melhores de A, C e M;
4. `periodo-1.c` : implementação do cálculo de período do gerador com os valores default de A, C e M;

5. `periodo-2.c` : idem, com os valores melhores de A, C e M.

Lembre-se:

- Identificar **TODOS** os seus arquivos (comentário com nome completo e GRR na primeira linha de cada arquivo).
- Compilar com a opção `-Wall` e resolver todos os *warnings*, pois eles penalizam a nota.
- Comentar seu código adequadamente (código sem comentários é penalizado).

[1)]
A reprodução da sequência pseudoaleatória não é um *bug*, mas uma característica útil, caso se deseje repetir os valores usados em uma simulação, ou repetir os passos que levaram o software a um erro, por exemplo.
[2)]
existem diversas outras métricas melhores para avaliar a qualidade de um PRNG, algumas delas são descritas nesta página