UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

Laboratório 2

Recursividade

Objetivo

O objetivo deste exercício é colocar em prática a implementação de programas recursivos na linguagem de programação C++.

Orientações gerais

Você deverá observar as seguintes observações gerais na implementação deste exercício:

1) Apesar da completa compatibilidade entre as linguagens de programação C e C++, seu código fonte não deverá conter recursos da linguagem C nem ser resultante de mescla entre as duas linguagens, o que é uma má prática de programação. Dessa forma, deverão ser utilizados estritamente recursos da linguagem C++.
2) Você deverá utilizar apenas um editor de texto simples (tais como o Gedit ou o Sublime) e o compilador em linha de comando, por meio do terminal do sistema operacional Linux.
3) Durante a compilação do seu código fonte, você deverá habilitar a exibição de mensagens de aviso (*warnings*), pois elas podem dar indícios de que o programa potencialmente possui problemas em sua implementação que podem se manifestar durante a sua execução.
4) Aplique boas práticas de programação. Codifique o programa de maneira legível (com indentação de código fonte, nomes consistentes, etc.) e documente-o adequadamente na forma de comentários.
5) Busque desenvolver o seu programa com qualidade, garantindo que ele funcione de forma correta e eficiente. Pense também nas possíveis entradas que poderão ser utilizadas para testar apropriadamente o seu programa e trate adequadamente possíveis entradas consideradas inválidas.
6) Para melhor organização do seu código, implemente diferentes funções e faça a separação da implementação do programa entre arquivos cabeçalho (`.h`) e corpo (`.cpp`) quando necessário.

Programa 1

Inventadas pelo matemático francês Édouard Lucas em 1883, as *Torres de Hanoi* são uma espécie de jogo que consiste de três pinos, sendo um de origem, um de destino e um auxiliar, e n discos de tamanhos diferentes. Inicialmente, todos os discos encontram-se empilhados no pino de origem em ordem decrescente de tamanho, de baixo para cima, como mostra a Figura 1. O objetivo do jogo é empilhar todos os discos no pino de destino atendendo às seguintes regras: (1) apenas um disco pode ser movido por vez; (2) cada movimento consiste em levar um disco que esteja no topo de uma das pilhas e colocá-lo no topo de outra pilha, ou seja, um disco só pode ser movido se ele estiver no topo de uma pilha; (3) um disco não pode ser colocado sobre outro disco de tamanho menor.

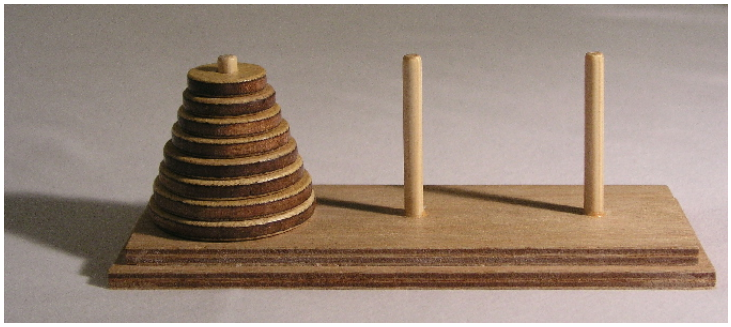

Figura 1: Ilustração das Torres de Hanoi

Implemente um programa que resolve, de forma recursiva, o problema das Torres de Hanoi. O programa deve receber um número natural positivo correspondente ao número de discos e deve exibir como resultado a lista de movimentos de disco necessários para encontrar a resposta seguido do número de movimentos requeridos. A função recursiva que resolve o problema das Torres de Hanoi deverá ter o seguinte protótipo:

```
void hanoi(int n, char origem, char destino, char aux)
```

em que `n` é o número de discos, `origem` é o identificador do pino de origem, `destino` é o identificador do pino de destino, e `aux` é o identificador do pino auxiliar. Por questões de simplicidade, os pinos serão identificados pelos caracteres `A`, `B` e `C`

Exemplo de execução do programa considerando `A` como pino de origem e `C` como pino de destino:

```
$ ./hanoi
Digite o numero de discos: 3

Movimentos de disco necessarios:
A -> C
A -> B
C -> B
A -> C
B -> A
B -> C
A -> C

Total de movimentos: 7
```

Nota: O algoritmo recursivo para resolver uma Torre de Hanoi com n discos a serem movidos de um pino de origem A para um pino de destino C utilizando um pino auxiliar B funciona basicamente da seguinte forma:

1) Mover $n-1$ discos do pino de origem A para o pino auxiliar B
2) Mover o disco n do pino de origem A para o pino de destino C
3) Mover $n-1$ discos do pino auxiliar B para o pino de destino C

Em algum ponto da execução do algoritmo, restará apenas um disco no pino de origem A, de modo que movê-lo para o pino de destino C é um passo trivial. É fato provado matematicamente que o número mínimo de movimentos necessários para resolver uma Torre de Hanoi com n discos é $2^n - 1$.

Programa 2

Na Matemática, o *fatorial* de um número natural não-negativo n, denotado por $n!$, é o produto de todos os números naturais positivos menores ou iguais a n, sendo definido como

$$n! = \prod_{k=1}^{n} k \qquad \forall n \in \mathbb{N}$$

Por exemplo, o fatorial de 3, escrito 3!, é igual a

$$3! = 1 \times 2 \times 3 = 6$$

De forma geral, pode-se ainda escrever

$$n! = n \times (n-1)$$

sendo convencionalmente $0! = 1$.

Com base na descrição acima, implemente um programa recursivo que calcule o fatorial de um número natural recebido como entrada.

Exemplo de execução do programa:

```
$ ./fatorial
Digite um numero natural: 7
O fatorial de 7 e 5040
```

Nota: A maioria das aplicações realiza a computação de fatoriais de números pequenos por meio de sucessivas multiplicações, como descrito anteriormente, sem potenciais problemas. Contudo, a principal dificuldade prática na computação de fatoriais diz respeito ao tamanho do resultado. Dependendo do compilador e da arquitetura do computador utilizados, exceder o valor máximo que pode ser representado em memória faz com que o programa realize cálculos de forma incorreta e retorne um resultado diferente do esperado, um erro conhecido como *overflow*. Por essa razão, deve-se utilizar outros tipos de dados com maior capacidade de representação em memória na implementação.

Programa 3

O *máximo divisor comum* (abreviadamente, MDC) entre dois ou mais números naturais positivos é o maior número natural que é fator desses números, isto é, o maior número natural positivo que divide todos os números em questão sem deixar resto. Por exemplo, os divisores comuns de 12 e 18 são 1, 2, 3 e 6, de modo que o MDC entre esses números é 6.

Dentre as diversas formas de se determinar o MDC de dois ou mais números naturais positivos, uma das mais simples e eficientes é o chamado *algoritmo de Euclides*, proposto por volta de 300 a.C. Dados dois números naturais positivos m e n tais que $0 \leq n \leq m$, o algoritmo de Euclides é um procedimento recursivo que consiste em efetuar divisões sucessivas de m e n até se chegar a uma divisão exata, de modo que o resto de uma divisão é utilizado como entrada para a próxima etapa de cálculo do MDC. Com isso, o MDC entre m e n seria o último resto diferente de zero obtido.

A título de exemplo, considere o cálculo do MDC entre 48 e 30. O algoritmo de Euclides calcula MDC(48, 30) da seguinte forma:

1) Divide-se o número maior pelo menor
   48 / 30 = 1 (com resto 18)
2) Divide-se sucessivamente o divisor da divisão anterior pelo o resto da divisão anterior:
   30 / 18 = 1 (com resto 12)
   18 / 12 = 1 (com resto **6**)
   12 / **6** = 0 (com resto zero)
3) O divisor da última divisão exata (com resto zero) será o MDC entre os dois números. Logo: MDC(48, 30) = 6.

Com base na descrição acima, implemente um programa recursivo que retorne o MDC de dois números naturais positivos.

Exemplo de execução do programa:

```
$ ./mdc
Digite dois numeros naturais positivos: 348 156
MDC(348,156) = 6
```

Programa 4

Um problema típico em Computação consiste em converter um número da sua forma decimal para a forma binária. Por exemplo, o número 10 tem a sua representação binária igual a 1010. A forma mais simples de se fazer isso é dividir sucessivamente o número em questão por 2, de modo que o resto da i-ésima divisão será o dígito i do número binário da direita para a esquerda. Implemente um programa recursivo que recebe como entrada um número inteiro não-negativo (isto é, número positivo incluindo o zero) e retorna a representação nesse número na forma binária.

Exemplo de execução do programa:

```
$ ./dec2bin
Digite um numero: 8
Representacao de 8 na forma binaria: 1000
```

Programa 5

Um *palíndromo* é uma palavra, frase, número ou qualquer outra sequência de unidades que pode ser lida e compreendida tanto da esquerda para a direita como da direita para a esquerda, indiferentemente. Em um palíndromo, normalmente são desconsiderados os sinais ortográficos assim como os espaços entre palavras. As palavras *radar* e *ovo* são exemplos de palíndromos.

Implemente um programa que receba uma palavra (na forma de uma *string*) e determine, de forma recursiva, se tal palavra é ou não um palíndromo. Para a entrada do seu programa, considere que serão fornecidas apenas *strings* sem acentos gráficos como `à`, `é`, `õ`, etc.

Exemplo de execuções do programa:

```
$ ./palindromo
Digite uma palavra: osso
"osso" e um palindromo

$ ./palindromo
Digite uma palavra: crescer
"crescer" nao e um palindromo
```

Extra: Como anteriormente mencionado, a verificação de palíndromos não se resume a palavras simples, mas pode também se aplicar a frases inteiras. Por exemplo, as frases *A rara arara* e *Socorrame, subi no ônibus em Marrocos* são exemplos de palíndromos. Para verificar se uma frase inteira é um palíndromo, o programa feito por você anteriormente deverá ser modificado: poderão ser criadas funções que removem todos os espaços em branco e separadores como hifens, vírgulas, ponto-e-vírgula, etc., e em seguida realizar o procedimento de verificação propriamente dito. Para tal, você poderá fazer uso de funções disponíveis na biblioteca `<string>`. Maiores detalhes e exemplos podem ser encontrados no endereço http://www.cplusplus.com/reference/string/string/.

Programa 6

O quadrado de um número natural pode ser calculado como a soma de todos os números ímpares inferiores ao dobro do número, como mostrado na tabela a seguir:

n	$1 + \ldots + (2n-1)$	n^2
1	1	1
2	$1 + 3$	4
3	$1 + 3 + 5$	9
6	$1 + 3 + 5 + 7 + 9 + 11$	36

Implemente uma função (e cada função dará origem a um programa distinto, como informado a seguir) que calcule o quadrado de um número natural, informado via linha de comando, utilizando:

a) recursão de cauda;
b) uma versão iterativa da recursão anteriormente desenvolvida.

Exemplo de execução dos programas:

```
$ ./quadrado_recursivo 5
quadrado(5) => 1 + 3 + 5 + 7 + 9 = 25

$ ./quadrado_iterativo 3
quadrado(3) => 1 + 3 + 5 = 9
```

Nota: O uso de parâmetros via linha de comando é possibilitado pelo uso das variáveis `argc` e `argv` passadas como parâmetro à função `main`. Dessa forma, o protótipo da função `main` deve passar a ser

```
int main (int argc, char* argv [])
```

em que o parâmetro `argc` (*argument count*) é um valor inteiro que informa a quantidade de parâmetros passados na linha de comando e o parâmetro `argv` (*argument vector*) refere-se a um vetor de *strings* que armazena cada parâmetro informado. Como o nome do programa é contabilizado, logo ele é armazenado na primeira posição do vetor `argv`; com isso, pode-se concluir que o valor do parâmetro `argc` é sempre pelo menos igual a 1 e que o tamanho do vetor `argv` é igual ao valor de `argc`. A título de exemplo, considere o seguinte programa em C++ que conta e mostra os argumentos recebidos na linha de comando:

```
#include <iostream>
using std::cout;
using std::endl;

int main(int argc, char* argv []) {
   cout << "Numero de parametros: " << argc << endl;
   for (int cont = 0; cont < argc; cont++) {
      cout << "Parametro " << cont << ":" << argv[cont] << endl;
   }
   return 0;
}
```

Posto em execução, o programa acima exibe a seguinte saída:

```
$ ./cont_arg x y z
Numero de parametros: 4
Parametro 0: cont_arg
Parametro 1: x
Parametro 2: y
Parametro 3: z
```