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. Anote ainda o código fonte para dar suporte à geração automática de documentação utilizando a ferramenta Doxygen (http://www.doxygen.org/). Consulte o documento extra disponibilizado na Turma Virtual do SIGAA com algumas instruções acerca do padrão de documentação e uso do Doxygen.
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.

Autoria e política de colaboração

O trabalho deverá ser feito **individualmente**. O trabalho em cooperação entre estudantes da turma é estimulado, sendo admissível a discussão de ideias e estratégias. Contudo, tal interação não deve

ser entendida como permissão para utilização de (parte de) código fonte de colegas, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outros colegas ou da Internet serão sumariamente rejeitados e receberão nota zero.

Entrega

Você deverá submeter um único arquivo compactado no formato `.zip` contendo todos os códigos fonte resultantes da implementação das soluções às questões deste exercício, sem erros de compilação e devidamente testados e documentados na forma de comentários, **até às 23h59 do dia 19 de março de 2017** através da opção *Tarefas* na Turma Virtual do SIGAA.

Avaliação

O trabalho será avaliado sob os seguintes critérios: (i) utilização correta dos conteúdos vistos anteriormente e nas aulas presenciais da disciplina; (ii) a corretude da execução dos programas implementados, que devem apresentar saída em conformidade com a especificação e as entradas de dados fornecidas, e; (iii) a aplicação correta de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte. A presença de mensagens de aviso (*warnings*) ou de erros de compilação e/ou de execução, a modularização inapropriada e a ausência de documentação são faltas que serão penalizadas. Este trabalho contabilizará nota de até 1,0 ponto na 1ª Unidade da disciplina.

Questão 1

Etapa 1. Desenvolva funções recursivas como solução para os seguintes problemas e indique qual tipo de recursão está sendo utilizada em cada um dos algoritmos:

a) Dado um valor *N*, calcular o valor da sequência *A*, definida como

$$A = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{N}$$

b) Dado um valor *n*, calcular o valor da sequência *B*, definida como

$$B = \frac{2}{4} + \frac{5}{5} + \frac{10}{6} + \frac{17}{7} + \frac{26}{8} + \ldots + \frac{(N^2 + 1)}{(N + 3)}$$

Etapa 2. Converta as funções recursivas anteriores, criadas por você, para suas respectivas versões iterativas.

Etapa 3. Desenvolva ainda um programa chamado `sequencia` para testar as suas funções. Seu programa deverá receber via linha de comando três parâmetros:

- a sequência a ser resolvida (A ou B);
- o tipo de função a ser utilizada para resolver a sequência, se recursiva (R) ou iterativa (I), e;
- o valor de *N* a ser usado no cálculo da sequência.

Exemplos de execução do programa seriam:

```
$ ./sequencia A R 4
O valor da sequencia A para N = 4 e 2.08 (a versão recursiva foi usada)

$ ./sequencia B I 3
O valor da sequencia B para N = 3 e 3.17 (a versão iterativa foi usada
```

Questão 2

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 chamado `mdc` que calcule de forma recursiva o MDC de dois números naturais positivos. Um exemplo de execução do programa seria:

```
$ ./mdc
Digite dois numeros naturais positivos: 348 156
MDC(348,156) = 12
```

Questão 3

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 chamado `dec2bin` 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, determinada de forma recursiva. Um exemplo de execução do programa seria:

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

Questão 4

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 chamado `palindromo` 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. Seu programa deve ser *case insensitive*, ou seja, não deve fazer distinção entre letras maiúsculas e minúsculas. Exemplos de execuções do programa seriam:

```
$ ./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 *Socorram-me, 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/.

Questão 5

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

Implemente um programa chamado `quadrado_recursivo` que calcule, utilizando uma **recursão de cauda**, o quadrado de um número natural informado via linha de comando. Feito isso, você deverá implementar um segundo programa distinto, chamado `quadrado_iterativo`, como uma versão iterativa do primeiro programa. Exemplos de execução desses programas seriam:

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

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

Questão 6

Escreva uma função recursiva que efetue uma busca ternária de um valor `x` sobre um vetor de inteiros. Tal função deverá ter a seguinte assinatura:

```
bool busca_ternaria(int v, int ini, int fim, int x)
```

em que `v` é o vetor de inteiros sobre o qual a busca será realizada, `ini` e `fim` são os limites inferior e superior dos índices do vetor entre os quais será feita a busca, e `x` o valor a ser buscado. A função deverá retornar o valor `true` (verdadeiro) se `x` for encontrado no vetor `v`, ou `false` (falso), caso contrário.

Para implementar o algoritmo de busca ternária, você poderá inspirar-se no algoritmo de busca binária aprendido na disciplina IMD0029 – Estruturas de Dados Básicas I ou equivalente. A ideia básica da busca ternária é dividir o vetor em três partes (em vez de duas) e comparar o valor a ser buscado com os dois elementos separadores dessas três partes. Caso o valor não seja encontrado nesses separadores, o algoritmo decide em qual das partes ele pode estar e continua, de forma recursiva, o procedimento de busca.

Por fim, desenvolva um programa chamado `ternaria` para testar a sua função. O programa deverá operar sobre o seguinte vetor:

```
[ 2 5 9 11 13 17 22 24 33 38 39 40 45 56 71 99 110 113 132 155 166 203 211 212 230 233]
```

Exemplos de execução do programa seriam:

```
$ ./ternaria 5
O elemento 5 faz parte do vetor.

$ ./ternaria 88
O elemento 88 nao faz parte do vetor.
```