Primeiro Exercício-Programa

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 19/06/2022

1 Implementação de diversas funções matemáticas

Nesse exercício prático você deverá implementar cinco funções, muitas delas relacionadas a operações matemáticas/aritméticas, mas não poderá usar funções prontas de outras bibliotecas.

Deverá, dessa reforma, implementar suas funções utilizando os operadores aritméticos (por exemplo, $\%, /, *, +, -$), condicionais, laços etc.

Função 1 - *void separaNumero(int valor)* : esta função deverá receber um número inteiro como parâmetro (*valor*). Se este número for menor que zero ou maior que 9999, a função deverá colocar o valor -1 em quatro variáveis globais existentes no programa: *unidades, dezenas, centenas e milhares*. Caso contrário, a função deverá separar o número de acordo com suas unidades, dezenas, centenas e milhares e colocar esses valores nas respectivas variáveis globais. Isto é, para o número 2345, deve atribuir o valor 2 na variável *milhares*, o valor 3 na variável *centenas*, o valor 4 na variável *dezenas* e o valor 5 na variável *unidades*.

Função 2 - *double potenciacao(double base, int expoente)* : esta função deverá receber um número real (*valor*) e um número inteiro (*expoente*). Se os dois parâmetros valerem zero, a função deverá retornar 0 (apenas como uma exceção e não para representar o resultado da operação); se o valor de *expoente* for negativo, a função também deverá retornar 0 (zero); se o valor de *expoente* for igual a zero e o valor de *base* for diferente de zero, deverá retornar 1 (um). Caso contrário, a função deverá retornar o resultado de $base^{expoente}$. Este resultado deverá ser obtido a partir de sucessivas multiplicações. Por exemplo, para *base*=3 e *expoente*=4, a função deverá retornar 81, valor obtido pela seguinte conta: 3*3*3*3.

Função 3 - *double somaIntervalo(double inicio, double fim, double passo)* : esta função deverá receber três números reais (*inicio, fim e passo*). Se qualquer desses números for negativo ou nulo a função deverá retornar 0 (apenas como uma exceção e não para representar o resultado da operação); se o valor de *fim* for menor do que o valor de *inicio*, a função também deverá retornar 0 (zero). Caso contrário, a função deverá retornar a soma de todos os valores começando por *inicio*, variando de acordo com o *passo* e enquanto forem menores do que *fim*. Por exemplo [valores decimais indicados com . (ponto) ao invés de , (vírgula)],

para *inicio*=3.2 e *fim*=4 e *passo*=0.2, a função deverá retornar 14, valor obtido pela soma dos seguintes valores: 3.2 + 3.4 + 3.6 + 3.8. Já para *inicio*=3.2, *fim*=4, *passo*=0.3, a função deverá retornar 10.5, valor obtido pela soma dos seguintes valores: 3.2 + 3.5 + 3.8.

Função 4 - *int passosDaDivisao(double dividendo, double divisor, double limiar)*: esta função deverá receber três números reais (*dividendo, divisor e limiar*). Se qualquer desses números for negativo ou nulo a função deverá retornar -1 (apenas como uma exceção e não para representar o resultado da operação); se o valor de *divisor* for menor ou igual a 1, a função deverá retornar -1; se o valor de *limiar* for maior do que o valor de *dividendo*, a função deverá retornar 0 (zero). Caso contrário, a função deverá retornar o número de vezes que é possível dividir o dividendo pelo divisor enquanto o resultado da operação for maior ou igual ao valor de *limiar*, dada a regra a seguir. A cada divisão, o valor do dividendo deve ser atualizado com o resultado da divisão realizada. Por exemplo [valores decimais indicados com . (ponto) ao invés de , (vírgula)], para *dividendo*=3.2 e *divisor*=1.6 e *limiar*=1.1, a função deverá retornar 3, pois haverá três passos até que o resultado sucessivo da divisão seja menor do que o limiar: 3.2/1.6=2 [primeiro passo (note que 2 é maior ou igual ao limiar)], 2/1.6 = 1.25 [segundo passo (note que 1.25 é maior ou igual ao limiar)], 1.25/1.6 = 0.78125 [terceiro passo (note que 0.78125 é menor do que o limiar)], assim, ocorreram três passos e a função deverá retornar 3. Já para *dividendo*=5 e *divisor*=8 e *limiar*=1.0, a função deverá retornar 1, pois 5/8 = 0.625 [primeiro passo (note que 0.625 é menor do que o limiar)], assim, ocorreu apenas um passo e a função deverá retornar 1.

Função 5 - *long fibonacci(int n)* : esta função deverá receber um número inteiro como parâmetro (n). Se este número for menor do que zero, a função deverá retornar -1 (apenas como uma exceção e não para representar o resultado da operação). Caso contrário, a função deverá retornar o n-ésimo número da sequência de Fibonacci, conforme descrito a seguir. O 'zerézimo' número de Fibonacci (isto é, para n=0) é igual a 0 (zero) [ou seja, fibonacci(0)=0], o número seguinte de Fibonacci (isto é, para n=1) é igual a 1 [ou seja, fibonacci(1)=1]. Para os demais valores de n, temos $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$, por exemplo fibonacci(2) = fibonacci(1) + fibonacci(0) = 1 + 0 = 1; já fibonacci(3) = fibonacci(2) + fibonacci(1) = 1 + 1 = 2; e fibonacci(4) = fibonacci(3) + fibonacci(2) = 2 + 1 = 3. Esta definição da função da sequência de Fibonacci (que usa a própria função para sua resolução) é chamada de definição recorrente e poderia ser resolvida por uma implementação recursiva (mas este será o assunto de outra aula!). O que se pede, neste EP, é que você implemente a função de forma iterativa. Isto é, para calcular o valor no 'n-ésimo' número da sequência de Fibonacci você calculará o segundo, terceiro, quarto ... até atingir o valor do n-ésimo. Já que cada número da sequência é calculado pela soma dos dois anteriores, você pode ter duas variáveis auxiliares, por exemplo *ultimo* e *penultimo* e a cada iteração o valor atual (por exemplo, guardado em uma terceira variável chamada *atual*) será igual a soma dessas duas variáveis e a variável *penultimo* deverá ser atualizada com o valor de *ultimo* e a variável *ultimo* deverá ser atualizada com o valor de *atual*. Por exemplo, para n=4 e sabendo que

os primeiros números de Fibonacci são 0 (zero) e 1, podemos ter três variáveis do tipo long (*penultimo, ultimo e atual*) e sempre iniciarmos *penultimo* com o valor 0 (zero) e *ultimo* com o valor 1. Dentro de um laço que deverá iterar, neste exemplo, três vezes (correspondendo a n=2, n=3 e n=4) teremos, na primeira iteração: $atual = ultimo + penultimo$ (ou seja, $1 + 0 = 1$); penultimo = ultimo (ou seja, 1); e $ultimo = atual$ (ou seja, 1). Na segunda iteração: $atual = ultimo + penultimo$ (ou seja, $1 + 1 = 2$); penultimo = ultimo (ou seja, 1); e $ultimo = atual$ (ou seja, 2). Na terceira iteração: $atual = ultimo + penultimo$ (ou seja, $2 + 1 = 3$); penultimo = ultimo (ou seja, 2); e $ultimo = atual$ (ou seja, 3). Assim, para n=4 a função deverá retornar 3 que é o valor da variável atual (após três iterações), conforme exemplificado.

2 Material a Ser Entregue

Um arquivo, denominado `NUSP.c` (sendo NUSP o seu número USP, por exemplo: 123456789.c), contendo o código das cinco funções solicitadas e as quatro variáveis globais definidas (e qualquer outra função auxiliar que você ache necessário). Para sua conveniência, `completeERenomeie.c` será fornecido, cabendo a você então completá-lo e renomeá-lo para a submissão.

Atenção!

1. Não modifique a assinatura das cinco funções solicitadas neste EP!
2. Para avaliação, apenas estas cinco funções serão invocadas diretamente (e, dependendo do teste, os valores das variáveis globais serão consultados). Em especial, qualquer código dentro da função `main()` será ignorado. Então certifique-se de que o problema seja resolvido chamando-se diretamente as funções implementadas.

3 Entrega

A entrega será feita única e exclusivamente via sistema e-Disciplinas, até a data final marcada. Deverá ser postado no sistema um arquivo .c, tendo como nome seu número USP:

```
seuNumeroUSP.c   (por exemplo, 123456789.c)
```

Não esqueça de preencher o cabeçalho constante do arquivo .c, com seu nome, número USP, turma etc.

A responsabilidade da postagem é exclusivamente sua. Por isso, submeta e certifique-se de que o arquivo submetido é o correto (fazendo seu *download*, por exemplo). Problemas referentes ao uso do sistema devem ser resolvidos com antecedência. Se, na data limite da submissão, o sistema e-Disciplinas estiver fora do ar, então serão aceitas submissões por e-mail (apenas se o sistema estiver fora do ar e dentro do prazo máximo de submissão).

4 Avaliação

A nota atribuída ao EP será focada nas funcionalidades solicitadas, porém não esqueça de se atentar aos seguintes aspectos:

1. Documentação: se há comentários explicando o que se faz nos passos mais importantes e para que serve o programa (tanto a função quanto o programa em que está inserido);
2. Apresentação visual: se o código está legível, identado etc;
3. Corretude: se o programa funciona.

Além disso, algumas observações pertinentes ao trabalho, que influenciam em sua nota, são:

- Este exercício-programa deve ser elaborado individualmente;
- Não será tolerado plágio, em hipótese alguma;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.