CREDIT SUISSE

Processo Seletivo para programador/desenvolvedor de Electronic Trading e Quantitative Strategies
Prova Específica de TI

Nome completo: ***Mauricio Lima***

Data : 04 / 12 / 2024

Informação: As questões podem ser respondidas utilizando qualquer linguagem/sintaxe de sua escolha.

Tempo de duração: 50 minutos.

1) Faça uma função que retorne TRUE, caso um determinado número for ímpar.

```
bool impar(long numero)
{
   return (numero % 2 != 0);
}
```

Repositório com código e testes:
https://github.com/mauriciolimaps/24-credit-suisse/tree/main/sources/tests/even-odd

CREDIT SUISSE

2) Sabe-se que a sequência Fibonacci cresce da seguinte maneira: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...
Podemos dizer que:
f(0) = 1
f(1) = 1
f(n) = f(n-1) + f(n-2)
Implemente uma função utilizando recursividade, que recebe o parâmetro n e retorna o número da sequência de Fibonacci na posição n.
Obs.: Não é necessário imprimir a sequência inteira.

Respondendo sinteticamente, seria a função a seguir,

```
unsigned long long fibonacci(unsigned int n)
{
    return (n < 2) ? 1 : fibonacci(n - 1) + fibonacci(n - 2);
}
```

mas cabem algumas observações. Primeiramente que a definição está equivocada, porque f(0) = 0, f(1) = 1, então estaria equivocado que f(0) = 1, como pode ser verificado em

https://pt.wikipedia.org/wiki/Sequ%C3%AAncia_de_Fibonacci

mas a implementação inicial não considera isso e segue o especificado no enunciado.

Além disso, testando observou-se que para o parâmetro até n = 34 é bem rápido, mas para números maiores, o aumento exponencial de recursões vai tornando a retorno cada vez mais lento, a ponto de ser inviável de utilizar na prática, portanto propõe-se a solução a seguir, que resguarda de alguns erros com algumas proteções de limite e reaproveitando alguns cálculos desnecessariamente repetidos que podem ser reaproveitados, então sugere-se uma implementação, ainda recursiva e mais eficiente, por reutilizar o que já foi calculado. Nessa solução também é corrigido o equívoco quando ao valor para n = 0, citado anteriormente.

```
#define FIBONACCI_MAX_INPUT    95

unsigned long long fibonacci(int n, bool reuse = true)
{
    if (n > 47 && sizeof(unsigned long long) == 4)
          throw std::overflow_error("O maior Fibonnaci calculável para 32
bits é o 47");

    if (n > 93 && sizeof(unsigned long long) == 8)
          throw std::overflow_error("O maior Fibonnaci calculável para 64
bits é o 93");
```

```
    if (reuse)
    {
        static unsigned long long previous[FIBONACCI_MAX_INPUT + 1];
        static bool initialized = false;

        if (!initialized)
        {
            previous[0] = 0;
            previous[1] = 1;
            for (int index = 2; index < FIBONACCI_MAX_INPUT; index++)
            {
                previous[index] = ULLONG_MAX;
            }

            initialized = true;
        }

        if (n < 2)
            return previous[n];

         previous[n - 1] = (previous[n - 1] == ULLONG_MAX) ? fibonacci(n -
1, true) : previous[n - 1];
         previous[n - 2] = (previous[n - 2] == ULLONG_MAX) ? fibonacci(n -
2, true) : previous[n - 2];

        return previous[n - 1] + previous[n - 2];
    }

    if (n == 0)
        return 0;

    return (n < 1) ? 1 : fibonacci(n - 1, false) + fibonacci(n - 2,
false);
}
```

CREDIT SUISSE

3) A estratégia Percentage of Volume (POV) é utilizada quando o cliente quer executar uma quantidade que acompanhe um determinado percentual do volume negociado de um papel no mercado. Por exemplo, o cliente quer fazer 10% do volume do papel PETR4. Ao serem negociadas 1000 ações no total do papel no mercado, é esperado que 100 ações desse total tenham sido executadas por essa estratégia. Faça uma função que, com a porcentagem e o número de ações negociadas, ela retorne o valor de ações que devem ser negociadas pela estratégia POV para atingir o volume esperado.

Considere:
decimal **funcaoRetornaQuantidade** (decimal porcentagem, **int totalNegociado**)
funcaoRetornaQuantidade (0.1, 900) = 100.

Ou seja, quando a função receber um **totalNegociado** no mercado de 900 ações e o percentual definido pelo cliente for 10%, a estratégia terá de executar 100 ações para totalizar 1000 ações negociadas no mercado. Implemente o corpo dessa função.

```
decimal funcaoRetornaQuantidade(decimal porcentagem, int totalNegociado)
{
      if ((porcentagem <  0) || (porcentagem >= 1))
            throw new Exception("O parâmetro de porcentagem deve ser positivo e
menor que 1");

      decimal quantity = totalNegociado * porcentagem / (1 - porcentagem);
      return quantity;
}
```

Repositório com código e testes:
https://github.com/mauriciolimaps/24-credit-suisse/tree/main/sources/percentage-volume-test

CREDIT SUISSE

4) Explique com suas palavras as seguintes estruturas de dados:
- Pilha
- Fila
- Fila de prioridade

Cada um desses é uma lista de elementos, em geral com alguma semelhança entre si, como a mesma estrutura de dados, sendo que estão encadeados em uma mesma lista de forma que possam ser adicionados e retirados elementos, mas cada um segue uma regra diferente, a saber:

a - A pilha segue o princípio LIFO, que é um sigla para “Last In, First Out”, que significa que o último elemento que entrou na fila, será o primeiro a ser retirado através de uma função ou método que lida com a piha.

b - A fila segue o princípio FIFO, que é uma sigla para “First In, First Out”, que significa que o primeiro que entrou, será o primeiro a sair, ou seja, por ordem de entrada na fila, que também é manipulado por funções ou métodos específicos para acessar os elementos da fila.

c - A “Fila com prioridade” é similar ao anterior, um FIFO, mas ao retirar observará se os elementos subsequentes tem prioridade sobre o primeiro que entrou. E faz isso sucessivamente até o final da fila, e pode retirar um elemento em qualquer ordem, seguindo um critério de prioridade. Quando há igualdade nas prioridades, vale o princípio FIFO.

5) Descreva um caso de uso em que você deveria usar uma lista e outro em que deveria usar um Dicionário (Map). Explique sua resposta.

Uma lista tem acesso aleatório baseado em um índice numérico que é a posição sequencial do item na lista. O elemento pode estar na lista de forma ordenada ou não, dependendo da forma que foi inserido, mas sempre será acessado por sua posição, enquanto em um dicionário, o elemento é localizado por uma chave (que pode ser um número também), mas pode ter um texto e identifica unicamente o elemento.

Um exemplo prático seria uma fila de atendimento a clientes, onde são chamados por ordem de chegada, então o próximo que será atendido por alguém disponível seguirá a ordem de chegada, portanto a posição do elemento na fila.

Um exemplo de dicionário seria uma busca de informações dos clientes baseados no CPF, onde o conjunto de informações é localizado usando o CPF como chave de busca.