BCC202 – Estruturas de Dados I (2023-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)

AULA PRÁTICA 4

- **Data de entrega: Até 04 de junho às 23:59:59.**

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Fibonacci

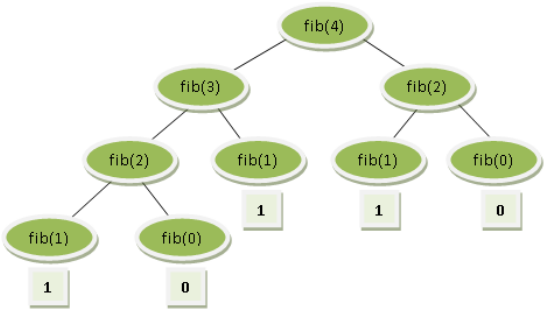
Quase todo estudante de Ciência da Computação recebe em algum momento no início de seu curso de graduação algum problema envolvendo a sequência de Fibonacci. Tal sequência tem como os dois primeiros valores 0 (zero) e 1 (um) e cada próximo valor será sempre a soma dos dois valores imediatamente anteriores. Por definição, podemos apresentar a seguinte fórmula para encontrar qualquer número da sequência de Fibonacci:

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n - 1) + fib(n - 2)
```

Uma das formas de encontrar o número de Fibonacci é por meio de chamadas recursivas. Isto é ilustrado a seguir, apresentando a árvore de derivação ao calcularmos o valor `fib(4)`, ou seja, o quinto valor desta sequência:

Desta forma:

- `fib(4) = 1 + 0 + 1 + 1 + 0 = 3`.
- Foram feitas 8 chamadas recursivas.



Considerações

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. Você deve criar um TAD `Fib` que tem uma função recursiva para efetuar o cálculo desejado. O TAD não precisa ser alocado e desalocado dinamicamente.

A informação lida da entrada deve ser armazenada no TAD `Fib` que contém os campos *i* (para armazenar o número fornecido na entrada), *resultado* (para armazenar o número de Fibonacci calculado) e *chamadas* (para armazenar o número de chamadas recursivas realizadas).

Note que os dois últimos valores podem ser números muito grandes e, portanto, um tipo adequado deve ser utilizado.

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes.

Especificação da Entrada e da saída

A primeira linha da entrada contém um único inteiro *n*, o qual indica o número de casos de teste. Cada caso de teste contém um número inteiro *i*, a partir do qual deve ser calculado o Fibonacci.

Para cada caso de teste de entrada deverá ser apresentada uma linha de saída, no seguinte formato:
`fib(i) = x chamadas = y`
onde **x** é o número de chamadas recursivas e **y** representa o número Fibonacci encontrado. Há sempre um espaço antes e depois do sinal de igualdade, conforme o exemplo abaixo:

Entrada	Saída
2 5 4	fib(5) = 14 chamadas = 5 fib(4) = 8 chamadas = 3

Diretivas de Compilação

```
$ gcc -c fib.c -Wall
$ gcc -c pratica.c -Wall
$ gcc fib.o pratica.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed:  0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.