Laboratório 5: recursão

Atenção: de agora em diante está terminantemente **proibido** usar arrays estáticos de tamanho variado como os exibidos abaixo:

```
int n;
int vetor[n]; // nunca definir a dimensão de um array com uma variável
```

Essa proibição aplica-se a tudo: listas, trabalho e avaliação. Qualquer programa usando o tipo de construção acima receberá nota **zero**. Todo array dinâmico deverá ser construído usando alocação dinâmica de memória:

```
int* vetor = calloc(n, sizeof(int)); // ou
int* vetor2 = malloc(n * sizeof(int));
```

INSTRUÇÕES

- Em todos os seus programas você deve gerenciar corretamente a memória, liberando toda a memória requerida pelo seu programa após o término do uso. Programas com vazamento de memória receberão uma penalização de 25% do valor da nota total do exercício. Você pode verificar se o seu programa possui vazamento de memória com o comando

  ```
  valgrind --leak-check=full /caminho/para/o/seu/programa
  ```

  Para que o comando acima funcione, você deve habilitar a *flag* de *debug* do seu compilador. O exemplo a seguir ilustra como deve ser feito caso você use o *gcc* para compilar o seu programa

  ```
  gcc -Wall -Wextra -Wvla -g -std=c99 arquivo.c
  ```

- Em vários exercícios, eu peço a vocês para que escrevam uma função de um determinado tipo. Além de escrever essa função, vocês também devem escrever uma função `main()` que irá usar essa função com dados fornecidos pelo usuário. Ou seja, a sua `main()` deverá pedir a entrada para o usuário e passar esses dados como parâmetro para a função que você desenvolveu. Requisite esses dados imprimindo mensagens na tela, para que o professor saiba o que digitar quando estiver corrigindo o seu trabalho.

Questão 1. Implemente a função recursiva

```
void count_down(int n);
```

Essa é uma função recursiva que faz uma contagem regressiva. O texto abaixo ilustra uma possível saída para a chamada `count_down(5)`.

```
5
4
3
2
1
```

```
ACABOU!
```

Questão 2. Implemente a função recursiva

```
double power(double a, int b);
```

Essa função recursiva retorna o valor de `a` elevado a potência `b`, isto é, o valor retornado é a^b.

Questão 3. Implemente a função recursiva

```
int soma(int v[], int n);
```

Essa função recursiva retorna a soma de todos os elementos do vetor `v`.

Questão 4. Implemente a função recursiva chamada `inverte`. Um dos parâmetros que essa função irá receber é `char word[]`. Você pode passar outros parâmetros se sentir necessidade. Como resultado, essa função deve inverter a palavra recebida, i.e., se a `word` armazenar a palavra "abobora", após a execução da função, ela armazenará a palavra "aroboba".

Questão 5. O *maior divisor comum* de dois inteiros a e b, denotado por $gcd(a,b)$, é o maior inteiro c que divide a e b. O Algoritmo de Euclides nos apresenta uma forma simples de computar $gcd(a,b)$:

$$gcd(a,b) = \begin{cases} a, & \text{se } b = 0; \\ gcd(b, a\%b), & \text{caso contrário.} \end{cases}$$

Implemente o algoritmo de Euclides em C.

Questão 6. Adapte o programa que resolve o problema da Torre de Hanoi visto em sala de aula. Ao invés de exibir os movimentos, imprima o número de movimentos necessários para mover os n discos.

Questão 7. Escreva a função recursiva abaixo:

```
void convert_to_binary(int n, char output[]);
```

Essa função recebe um número inteiro `n` como entrada e converte-o em uma representação binária na forma de string. O resultado da conversão é armazenado na variável `output`.

Questão 8. Implemente uma função recursiva para gerar a sequência de Collatz para um dado número inteiro positivo. A sequência de Collatz é gerada a partir das seguintes regras:

- Se o número atual for par, divida-o por 2.
- Se o número atual for ímpar, multiplique-o por 3 e some 1.

A função deve imprimir cada termo da sequência até que o número atual seja 1, momento em que a recursão deve parar.

Como exemplo, temos que a sequência de Collatz para o número 7 é:

```
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

Curiosidade Acredita-se que a sequência de Collatz converge para 1 para qualquer inteiro, mas, até os dias atuais, ninguém conseguiu provar essa afirmação. Em outras palavras, pode ser que o seu programa rode para sempre e que você não tenha nenhuma culpa disso :)

Questão 9. Escreva uma função recursiva que calcule $\binom{n}{k}$, para $n \geq 0$ e $k \geq 0$. Para isso, observe a seguinte propriedade:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Questão 10. Escreva a seguinte função recursiva:

```
void list_all_bit_strings(int n);
```

Essa função imprime todas as sequências com `n` bits (sem repetição). A ordem em que as cadeias de bits são impressas não importa, apenas tome cuidado para não imprimir a mesma cadeia mais de uma vez. O exemplo abaixo mostra uma das saídas possíveis para quando $n = 3$.

```
000
001
010
011
100
101
110
111
```

Questão 11. Escreva um programa recursivo capaz de encontrar a saída de um labirinto. O mapa do labirinto segue o seguinte formato:

- A entrada do labirinto está no canto superior esquerdo.
- A saída do labirinto está no canto inferior direito.
- As paredes do labirinto são marcadas com o caractere `#`.

Veja um exemplo de instância abaixo.

```
  ###########
          # #
# ##### ### #
# # #       #
# # # # ### #
# #   # #   #
### # ### ###
#   #   # # #
# # ##### # #
# #   #     #
##### # ### #
#     # #
###########
```

Você pode criar outros labirintos para o seu programa no *Maze Generator*[1].

O seu programa recursivo deve imprimir o percurso da entrada até a saída do labirinto. Veja o exemplo abaixo.

[1] https://www.dcode.fr/maze-generator

```
. ###########
........  # #
# #####.### #
# # #  .....#
# # # # ###.#
# #   # #...#
### # ###.###
#   #   #.# #
# # #####.# #
# #   #  ...#
##### # ###.#
#     # #  .
###########..
```

Questão 12. O **Merge Sort** é um algoritmo de ordenação que segue o paradigma *dividir para conquistar*. O algoritmo funciona da seguinte forma:

1. **Dividir:** O vetor original com n elementos é divido em dois subvetores, um de tamanho $\lceil n/2 \rceil$ e outro de tamanho $\lfloor n/2 \rfloor$.
2. **Conquistar:** Os subvetores são ordenados recursivamente.
3. **Combinar:** Os subvetores ordenados são combinados (intercalados) em um único vetor ordenado.

Exemplo

Considere o vetor

$$[38, 27, 43, 3, 9, 82, 10]$$

- Divida o vetor ao meio:

$$[38, 27, 43, 3] \quad \text{e} \quad [9, 82, 10]$$

- Ordene os subvetores de forma recursiva:

$$[3, 27, 38, 43] \quad \text{e} \quad [9, 10, 82]$$

- Combine os subvetores ordenados. Note que podemos fazer isso de uma forma esperta (percorrendo cada subvetor apenas uma vez).

$$[3, 9, 10, 27, 38, 43, 82]$$

Implemente o algoritmo **Merge Sort** em C para ordenar um vetor de números inteiros.

Questão 13. Considere uma sequência formada pelos caracteres ‘(‘ e ‘)‘. Dizemos que uma sequência de parênteses é **válida** se:

1. Cada parêntese de abertura ‘(‘ possui um parêntese de fechamento ‘)‘ correspondente.
2. Em qualquer prefixo da sequência, o número de parênteses de fechamento ‘)‘ nunca excede o número de parênteses de abertura ‘(‘.

Exemplos de Sequências Válidas

- `"()()"`
- `"((()))"`
- `"(()())"`

Exemplos de Sequências Inválidas

- `"(()"` *Falta um parêntese de fechamento*
- `"())"` *Parêntese de fechamento extra*
- `")()("` *Parêntese de fechamento antes do parêntese de abertura*

Implemente uma função recursiva em C que determine se uma sequência de parênteses é válida.

Questão 14. Escreva um programa recursivo que, dados n números positivos e um inteiro k, onde $1 \leq k \leq n$, seja capaz de imprimir todas as combinações de k elementos dos valores fornecidos. Um exemplo de entrada possível seria

```
4 3
1
7
2
8
```

Na entrada acima, o usuário informa que entrará com 4 valores e que quer uma combinação de 3 em 3. Uma possível saída seria:

```
1 7 2
1 7 8
1 2 8
7 2 8
```

Questão 15. Escreva um programa recursivo que, dado um número positivo n fornecido pelo usuário, seja capaz de imprimir todas as combinações dos n primeiros naturais. Para $n = 3$, um exemplo de saída seria:

```
1
2
3
12
13
23
123
```