UTFPR - Universidade Tecnológica Federal do Paraná
DACOM - Departamento de Computação
BCC3001 :: Algoritmos e Estruturas de Dados 1

Exercícios 01 :: Ponteiros & Alocação de Memória

Instruções Gerais

- Faça todos os exercícios em um único arquivo .c. Utilize a função main() para fazer chamadas de testes às funções solicitadas pelos exercícios.
- Utilize a extensão .c, o compilador gcc e o editor de sua preferência: VS Code, Dev C++, etc.
    - Alternativamente, utilize https://replit.com/languages/c.

OBS: Lembre-se de, ao testar cada função na main(), liberar a memória alocada pela(s) função(ões) para evitar vazamentos de memória.

1. Escreva uma função que aloca na heap um array de inteiros de tamanho **n**, preenchido com zeros, e retorna seu endereço.
   `int* create_vector(int n)`

   Exemplo de uso da função:
   ```
   int* v1 = create_vector(7);   // aloca array de 7 inteiros na heap
   print_vector(7, v1);          // imprime 0,0,0,0,0,0,0
   free(v);                      // libera array alocado na heap
   ```

2. Escreva uma função que receba um vetor de inteiros e seu tamanho **n**. A função deve retornar o endereço de um novo vetor, alocado na heap, contendo os **max** maiores elementos contidos no vetor original. OBS: você pode modificar o conteúdo do vetor original, se necessário.
   `int* get_largest(int n, int* v, int max)`

   Exemplo de uso da função:
   ```
   int v0[] = {6,10,2,1,2,3,9}
   int* v1 = get_largest(7, v0, 2); // retorna vetor com os 2 maiores valores de v0
   print_vector(2, v1);             // imprime 10,9
   free(v1);                        // libera array alocado na heap
   ```

3. Escreva uma função que receba o endereço de uma string. A função deve retornar o endereço de uma cópia da string alocada na heap.
   ```
   char* copy_string(const char* str)  // 'const' indica que o vetor original não
                                       // pode ser modificado dentro da função
   ```

4. Escreva uma função que receba o endereço de uma string. A função deve retornar o endereço de uma cópia reversa da string alocada na heap.
   `char* copy_reverse(const char* str)`

5. Escreva uma função que concatene duas strings e retorne o resultado em uma nova string, alocada na heap.
   `char* concat_string(const char* str1, const char* str2)`

6. Escreva uma função que realize a união entre dois vetores, retornado o resultado em um vetor alocado na heap. Considere que não há elementos repetidos dentro de um mesmo vetor.
   `int* array_union(int n1, const int* v1, int n2, const int* v2)`

7. Escreva uma função que realize a intersecção entre dois vetores, retornado o resultado em um vetor alocado na heap. Considere que não há elementos repetidos dentro de um mesmo vetor.
   `int* array_intersection(int n1, const int* v1, int n2, const int* v2)`

8. Escreva uma função que recebe o endereço e as dimensões de uma matriz alocada na heap. A função deve imprimir seu conteúdo. Lembre-se: dentro da função, o acesso aos elementos da matriz **v** ocorre normalmente por meio do operador de índice. Ex: v[0][2].
   `void print_vector2D(int m, int n, const int** v)`

9. Escreva uma função que aloca na heap uma matriz de inteiros de dimensões mxn e retorna o seu endereço. A matriz deve ser preenchida com uma sequência crescente de inteiros.
   `int** create_vector2D(int m, int n)`

10. Escreva uma função que receba uma matriz na stack como entrada e retorne o endereço de sua transposta, alocada na heap.
    `int** transpose(int m, int n, int v[m][n])`

11. Escreva uma função que receba o endereço de uma string contendo palavras separadas por espaços. A função deve retornar o endereço de um vetor de strings (matriz de caracteres) contendo as palavras separadas. Além disso, deve retornar, via parâmetro, o número de palavras encontradas.
    `char** split(const char* str, int* n)`

    Exemplo de uso da função:

```
char str[] = "  Texto de   teste  com várias palavras  ";
int size = 0;
char** tokens = split(str, &size);  // devolve vetor de strings com as palavras

for (int i = 0; i < size; i++)
    printf("%s\n", tokens[i]);      // imprime as palavras encontradas

// libera vetor de strings alocado na heap
for (int i = 0; i < size; i++)
    free(tokens[i]);
free(tokens);
```