UTFPR - Universidade Tecnológica Federal do Paraná
DACOM - Departamento de Computação
BCC2001 :: Algoritmos

Lista de Exercícios no. 05 :: Vetores

Instruções Gerais

- Os exercícios são de resolução individual.
- Crie um único arquivo .c e faça cada exercício em uma ou mais funções. Teste-as na main().
- Utilize o compilador gcc e o editor VS Code (ou outro editor de sua preferência).
- **Não é permitido o uso de recursos que ainda não foram abordados na disciplina até o momento da publicação desta lista. Esta lista considera até:**
    - **Vetores (unidimensionais alocados em pilha).**

1. Escreva uma função que recebe um vetor **vet** de tamanho **n** e imprime apenas os valores pares.
   `void print_even(int n, int vet[])`

2. Escreva uma função que recebe um vetor **vet** de tamanho **n** contendo números inteiros positivos e negativos. A função deve inverter o sinal dos números negativos, passando-os para positivo.
   `void set_positive(int n, int vet[])`

   `Entrada:{1, -5, 67, -45, -1, -1, 0, 48} → Saída:{1, 5, 67, 45, 1, 1, 0, 48}`

3. Escreva uma função que recebe um vetor **vet** de tamanho **n** e devolve o maior valor contido no mesmo.
   `int find_max(int n, int vet[])`

4. Escreva uma função que recebe um vetor **vet** de tamanho **n**, bem como, um elemento **elem** a ser procurado. A função deve substituir todas as ocorrência de **elem** por -1.
   `void replace_all(int n, int vet[], int elem)`

5. Escreva uma função que faz a leitura de **n** números inteiros e os coloca no vetor **vet** fornecido. Considere que o **vet** possui tamanho **n**.
   `void read_vector(int n, int vet[])`

6. Escreva uma função que verifica se os elementos de um vetor, passado como parâmetro, estão em ordem crescente. A função deve retornar 1 (true), caso os elementos estejam dispostos em ordem crescente, ou 0 (false), em caso contrário.
   `int is_sorted(int n, int vet[])`

   ```
   Exemplo de uso da função:
   int v[] = {1,4,7,9,15,22,48,512};
   int res = is_sorted(8, v); // neste caso, res = 1 (true)
   ```

7. Escreva uma função que recebe pontos X,Y em um vetor **points** de tamanho **n**. O vetor conterá os pontos sequencialmente: [X1, Y1, X2, Y2, X3, Y3,...]. A função deve informar a distância entre cada par de pontos consecutivos. Distância: sqrt((x2 - x1)*(x2 - x1) + (y2 - y1)*(y2 - y1)). Função **square root**: https://www.tutorialspoint.com/c_standard_library/c_function_sqrt.htm).
   `void distances(int n, int points[])`

8. Escreva uma função que recebe um vetor e inverte os seus elementos. OBS: o código não deve apenas imprimir o vetor ao contrário, mas sim, inverter os elementos no próprio vetor.
   `void reverse(int n, int vet[])`

9. Escreva uma função que recebe dois vetores (**vet1** e **vet2**) de mesmo tamanho **n**. A função deve copiar todos os elementos de **vet1** para **vet2**.
   `void vector_copy(int n, int vet2[], int vet1[])`

10. Um vetor pode ser utilizado como uma lista que permite inserções e remoções de números inteiros positivos. No vetor, o valor -1 indica posição vazia (disponível). Escreva uma função que recebe um **vet1** de tamanho **n** e um novo elemento. A função deve inserir o novo elemento na primeira posição válida disponível, isto é, marcada com -1 (se houver). A função deve retornar 1, caso a inserção ocorra, ou 0, em caso contrário.
    `int vector_insert(int n, int vet1[], int elem)`

    ```
    int v[9] = {1, 6, -1, 9, 4, -1, -1, 2, -1}
    vector_insert(9, v, 18);   // v = {1, 6, 18, 9, 4, -1, -1, 2, -1}
    vector_insert(9, v, 7);    // v = {1, 6, 18, 9, 4, 7, -1, 2, -1}
    ```

11. Considerando o mesmo formato de lista do exercício anterior, escreva uma função que recebe um **vet1** de tamanho **n** e uma posição. A função deve "remover" o elemento da lista, marcando a posição com -1. Caso a posição seja inválida, nenhuma operação deve ser realizada.
    `void vector_remove(int n, int vet1[], int pos)`

12. Considere as operações dos dois exercícios anteriores. Após algumas operações de inserção e remoção, o vetor ficará repleto de "buracos". Escreva uma função que recebe um vetor **vet** de tamanho **n**. O vetor contém inteiros positivos e posições livres, marcadas com -1. A função deve desfragmentar o vetor, colocando todos os valores válidos à esquerda.
    `void vector_defrag(int n, int vet[n])`

    ```
    Exemplo:
    int v[9] = {1, 6, -1, 9, 4, -1, -1, 2, -1} // vetor original
    vector_defrag(9, v); // v = {1, 6, 9, 4, 2, -1, -1, -1, -1}
    ```

13. Escreva uma função que recebe três vetores e seus tamanhos. A função deve concatenar (juntar) o conteúdo de **v1** e **v2** em **v3**. Considere que **v3** tem tamanho **n1** + **n2**. Os vetores contém apenas inteiros positivos.
    `void vector_concat(int n1, int v1[n1], int n2, int v2[n2], int v3[])`

    ```
    Exemplo:
    int v1[5] = {1,2,3,4,5};
    int v2[3] = {2,3,8};
    int v3[8];
    vecor_concat(5, v1, 3, v2, v3);     // v3 = {1,2,3,4,5,2,3,8}
    ```

14. Escreva uma função que recebe dois vetores **v1** e **v2** ordenados crescentemente. Ela deve mesclar ordenadamente os conteúdos de **v1** e **v2**, colocando em **v3**. Considere que **v3** tem tamanho **n1** + **n2**. Note que é possível se beneficiar do fato dos vetores estarem ordenados. OBS: você não deve concatenar e ordenar.
    `void merge(int n1, int v1[], int n2, int v2[], int v3[])`

```
Entrada:    v1 = {1, 3, 4, 7, 9, 10}
            v2 = {2, 3, 5, 7, 7, 14}
Saída:      v3 = {1, 2, 3, 3, 4, 5, 7, 7, 7, 9, 10, 14}
```

15. Escreva uma função que recebe um vetor contendo números inteiros. Ela deve determinar o segmento de soma máxima, retornando a soma.
    `int max_sum_slice2(int n, int v[])`

```
Exemplo: No vetor {5, 2,-2,-7, 3,14,10,-3, 9,-6, 4, 1}, a soma do segmento é 33.
```

16. Escreva uma função que recebe um vetor preenchido com inteiros positivos. A função deve imprimir as ocorrências (contagem) de cada número no vetor. Dica: utilize um vetor **count** para armazenar a contagem de cada elemento no vetor **vet**, relacionando as posições de **count** aos valores em **vet**.
    `void count_elements(int n, int vet[])`

17. Escreva uma função que recebe um valor inteiro **value** e um vetor **notes** com as possíveis cédulas. A função deve imprimir a quantidade mínima de cédulas equivalente ao valor.
    Dica: use um vetor auxiliar **count** para armazenar a contagem de cada cédula.
    `void min_bills(int value, int n, int bills[]) // n é o tamanho de bills`

    ```
    Exemplo:
    min_bills(209, 5, (int[]){1,5,10,50,100}); // lista de tipos de cédulas
    2 x R$100,00
    1 x R$5,00
    4 x R$1,00
    ```

18. Escreva uma função que recebe três vetores e seus tamanhos. A função deve realizar a união entre os vetores **v1** e **v2**, colocando os valores em **v3**. Considere que **v3** tem tamanho **n1** + **n2**. Os vetores contém apenas inteiros positivos e **v3** deve ser iniciado com 0.
    `void vector_union(int n1, int v1[n1], int n2, int v2[n2], int v3[])`

    ```
    Exemplo:
    int v1[5] = {1,2,3,4,5};
    int v2[3] = {2,3,8};
    int v3[8] = {0};
    vector_union(5, v1, 3, v2, v3);     // v3 = {1,2,3,4,5,8,0,0}
    ```

19. Escreva uma função que recebe três vetores e seus tamanhos. A função deve realizar a intersecção entre os vetores **v1** e **v2**, colocando os valores em **v3**. Considere que **v3** tem tamanho min(**n1**, **n2**). Os vetores contém apenas números naturais (inteiros positivos) e **v3** deve ser iniciado com 0.
    `void vector_intersection(int n1, int v1[n1], int n2, int v2[n2], int v3[])`

    Exemplo:

```
int v1[5] = {1,2,3,4,5};
int v2[3] = {2,3,8};
int v3[3] = {0};
vector_intersection(5, v1, 3, v2, v3);    // v3 = {2,3,0}
```