# Universidade do Estado de Santa Catarina
# Centro de Ciências Tecnológicas - CCT
# Departamento de Ciência da Computação - DCC
# Processo Seletivo 07/2026 - PROVA ESCRITA - PADRÃO DE RESPOSTA
# ÁREA: Algoritmos e Linguagem de Programação

1 -) (3,0) Utilizando a linguagem de programação C, é possível implementar uma matriz genérica utilizando um descritor e um único bloco de memória alocado dinamicamente, independentemente do tipo de dado armazenado. Para isso, os elementos são organizados sequencialmente na memória deste bloco, linha a linha, e cada elemento ocupa uma quantidade fixa de bytes.

Considere a seguinte estrutura para representar o descritor de uma matriz genérica:

```c
typedef struct {
    int linhas;
    int colunas;
    size_t tamanho_dados;
    void *dados;
} matriz_t;
```

O campo `tamanho_dados` indica a quantidade de bytes ocupados por cada elemento, enquanto `dados` aponta para o bloco de memória que armazena todos os elementos da matriz. Os campos `linhas` e `colunas` indicam o número de linhas e colunas da matriz, respectivamente.

Implemente as seguintes funções em C:

(a) (0,5) `matriz_t* criar_matriz(int linhas, int colunas, size_t tamanho_dados)`

Recebe o número de linhas, o número de colunas e o tamanho, em bytes, de cada elemento. Aloca dinamicamente a estrutura do descritor e o bloco de memória para armazenar os elementos e retorna um ponteiro para o descritor da matriz criada. Retorna `NULL` caso algum parâmetro seja inválido ou ocorra uma falha de alocação.

(b) (0,5) `int get(matriz_t *matriz, int linha, int coluna, void *dado)`

Recebe um ponteiro para o descritor da matriz, os índices de um elemento e um ponteiro para uma região de memória na qual deverá copiar o valor desse elemento. Retorna 0 em caso de sucesso e 1 em caso de parâmetros inválidos ou índices fora dos limites da matriz.

(c) (0,5) `int set(matriz_t *matriz, int linha, int coluna, void *dado)`

Recebe um ponteiro para o descritor da matriz, os índices de um elemento e um ponteiro para o valor a ser armazenado. Copia o valor para a posição correspondente da matriz. Retorna 0 em caso de sucesso e 1 em caso de parâmetros inválidos ou índices fora dos limites da matriz.

(d) (1,5) `matriz_t* transposta(matriz_t *matriz)`

Recebe um ponteiro para o descritor da matriz e retorna um ponteiro para um novo descritor da matriz que corresponde à sua transposta. A nova matriz deve manter o mesmo tipo de dado, representado por `tamanho_dados`, e ser alocada dinamicamente. A função deve retornar NULL caso ocorra uma falha de alocação. Não utilize as funções `set` e nem `get`.

Exemplo da transposta:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Pág. 01 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

**RESPOSTA**:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdint.h>

typedef struct {
    int linhas;
    int colunas;
    size_t tamanho_dados;
    void *dados;
} matriz_t;

//a)
matriz_t *criar_matriz(int linhas, int colunas, size_t tamanho_dados) {
    if (linhas <= 0 || colunas <= 0 || tamanho_dados == 0)
        return NULL;

    size_t elementos = (size_t)linhas * (size_t)colunas;

    matriz_t *matriz = malloc(sizeof(matriz_t));
    if (matriz == NULL)
        return NULL;

    matriz->dados = malloc(elementos * tamanho_dados);
    if (matriz->dados == NULL) {
        free(matriz);
        return NULL;
    }

    matriz->linhas = linhas;
    matriz->colunas = colunas;
    matriz->tamanho_dados = tamanho_dados;

    return matriz;
}

// b)
int get(matriz_t *matriz, int linha, int coluna, void *dado) {
    if (matriz == NULL || matriz->dados == NULL || dado == NULL || linha < 0 ||
        linha >= matriz->linhas || coluna < 0 || coluna >= matriz->colunas)
        return 1;

    size_t indice = (size_t)linha * matriz->colunas + coluna;
    unsigned char *origem = (unsigned char *)matriz->dados +
        indice * matriz->tamanho_dados;

    memcpy(dado, origem, matriz->tamanho_dados);

    return 0;
}

// c)
int set(matriz_t *matriz, int linha, int coluna, void *dado) {
    if (matriz == NULL || matriz->dados == NULL || dado == NULL || linha < 0 ||
        linha >= matriz->linhas || coluna < 0 || coluna >= matriz->colunas)
        return 1;

    size_t indice = (size_t)linha * matriz->colunas + coluna;
    unsigned char *destino = (unsigned char *)matriz->dados +
        indice * matriz->tamanho_dados;

    memcpy(destino, dado, matriz->tamanho_dados);
```

Pág. 02 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

```c

    return 0;
}

// d)
matriz_t *transposta(matriz_t *matriz) {
    if (matriz == NULL || matriz->dados == NULL || matriz->linhas <= 0 ||
        matriz->colunas <= 0 || matriz->tamanho_dados == 0)
        return NULL;

    matriz_t *T = criar_matriz(matriz->colunas, matriz->linhas,
        matriz->tamanho_dados);

    if (T == NULL)
        return NULL;

    for (int i = 0; i < matriz->linhas; i++) {
        for (int j = 0; j < matriz->colunas; j++) {
            size_t indice_origem = (size_t)i * matriz->colunas + j;
            size_t indice_destino = (size_t)j * T->colunas + i;
            unsigned char *origem = (unsigned char *)matriz->dados +
                indice_origem * matriz->tamanho_dados;
            unsigned char *destino = (unsigned char *)T->dados +
                indice_destino * T->tamanho_dados;
            memcpy(destino, origem, matriz->tamanho_dados);
        }
    }

    return T;
}
```

Pág. 03 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

2 -) (3,0) Utilizando a linguagem de programação C, desenvolva um programa que leia $n$ números de CPF informados pelo usuário, contendo apenas os 11 dígitos numéricos, sem pontos ou traços. Após a leitura de todos os CPFs, o programa deverá realizar a validação e apresentar, na mesma ordem em que foram informados, cada CPF acompanhado dos dígitos verificadores calculados e da indicação de que é válido ou inválido. Os CPFs deverão ser armazenados como números inteiros. O programa deverá utilizar alocação dinâmica de memória para armazenar todos os CPFs informados antes de realizar a validação.

**ENTRADA**

A entrada consiste em uma linha contendo o número $n$, seguida de $n$ linhas, cada uma contendo um número de CPF com 11 dígitos.

**SAÍDA**

A saída deverá apresentar, para cada CPF informado, o número do CPF, os dígitos verificadores calculados e uma mensagem indicando se o CPF é válido ou inválido. Os resultados deverão respeitar a ordem de entrada e seguir o formato apresentado no exemplo de saída abaixo.

**EXEMPLO**

| Entrada | Saída |
|---|---|
| 3<br>11761988408<br>11144477735<br>42317695837 | CPF: 117.619.884-08<br>d1 = 0<br>d2 = 9<br>> CPF inválido!!!<br>CPF: 111.444.777-35<br>d1 = 3<br>d2 = 5<br>CPF válido!!!<br>CPF: 423.176.958-37<br>d1 = 3<br>d2 = 7<br>CPF válido!!! |

**VALIDAÇÃO DO CPF**

**a - Cálculo do primeiro dígito**

O primeiro passo é calcular o primeiro dígito verificador, e para isso, separamos os primeiros 9 dígitos do CPF (**111.444.777**) e multiplicamos cada um dos números, **da direita para a esquerda** por números crescentes a partir do número 2, como no exemplo abaixo:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 7 | 7 | 7 |
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 10 | 9 | 8 | 28 | 24 | 20 | 28 | 21 | 14 |

Multiplicamos cada dígito do CPF pelo respectivo número e somamos cada um dos resultados:

**10+9+8+28+24+20+28+21+14 = 162**

Pegamos o resultado obtido **162 e dividimos por 11**. Consideramos como quociente apenas o valor inteiro.

**162 / 11 = 14 com resto 8**

- Se o resto da divisão for menor que 2, então o dígito é igual a **0 (Zero)**.
- Se o resto da divisão for maior ou igual a 2, então o dígito verificador é igual a 11 menos o resto da divisão **(11 - resto)**.

Pág. 04 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

No nosso exemplo temos que o resto é 8 então faremos **11-8 = 3**

Logo o primeiro dígito verificador é 3. Então podemos escrever o CPF com os dois dígitos calculados: **111.444.777-3X**

**b - Cálculo do segundo dígito**

Para calcular o segundo dígito vamos usar o primeiro dígito já calculado. Vamos montar a mesma tabela de multiplicação usada no cálculo do primeiro dígito. Só que desta vez usaremos na segunda linha os valores 11,10,9,8,7,6,5,4,3,2 já que estamos incluindo mais um dígito no cálculo (*o primeiro dígito calculado*):

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 4 | 4 | 4 | 7 | 7 | 7 | 3 |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| 11 | 10 | 9 | 32 | 28 | 24 | 35 | 28 | 21 | 6 |

Novamente, efetuamos somamos o resultado da multiplicação: **11 + 10 + 9 + 32 + 28 + 24 + 35 + 28 + 21 + 6 = 204**

Dividimos o total do somatório por 11 e consideramos o resto da divisão.

**204 / 11 = 18 e resto 6**

Após obter o resto da divisão, precisamos aplicar a mesma regra que utilizamos para obter o primeiro dígito:

- Se o resto da divisão for menor que 2, então o dígito é igual a **0 (Zero)**.
- Se o resto da divisão for maior ou igual a 2, então o dígito é igual a 11 menos o resto da divisão **(11 - resto)**.

**11-6 = 5** Logo 5 é o nosso segundo dígito verificador.

Logo o nosso CPF fictício será igual a: **111.444.777-35**.

**RESPOSTA**:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> //pow(x,y)

void valida_cpfs(long long int *cpf) {
  int i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11;
  int r, resto, d1, d2;
  //i1 = cpf/10000000000; // igual a cpf/pow(10,10);
  i1 =   *cpf/pow(10,10);
  i2 =  (*cpf/pow(10,9))-(i1*pow(10,1));
  i3 =  (*cpf/pow(10,8))-(i1*pow(10,2) + i2*pow(10,1));
  i4 =  (*cpf/pow(10,7))-(i1*pow(10,3) + i2*pow(10,2) + i3*pow(10,1));
  i5 =  (*cpf/pow(10,6))-(i1*pow(10,4) + i2*pow(10,3) + i3*pow(10,2) +
    i4*10);
  i6 =  (*cpf/pow(10,5))-(i1*pow(10,5) + i2*pow(10,4) + i3*pow(10,3) +
    i4*pow(10,2) + i5*pow(10,1));
  i7 =  (*cpf/pow(10,4))-(i1*pow(10,6) + i2*pow(10,5) + i3*pow(10,4) +
    i4*pow(10,3) + i5*pow(10,2) + i6*pow(10,1));
  i8 =  (*cpf/pow(10,3))-(i1*pow(10,7) + i2*pow(10,6) + i3*pow(10,5) +
    i4*pow(10,4) + i5*pow(10,3) + i6*pow(10,2) + i7*pow(10,1));
  i9 =  (*cpf/pow(10,2))-(i1*pow(10,8) + i2*pow(10,7) + i3*pow(10,6) +
    i4*pow(10,5) + i5*pow(10,4) + i6*pow(10,3) + i7*pow(10,2) + i8*pow(10,1));
  i10 = (*cpf/pow(10,1))-(i1*pow(10,9) + i2*pow(10,8) + i3*pow(10,7) +
    i4*pow(10,6) + i5*pow(10,5) + i6*pow(10,4) + i7*pow(10,3) +
    i8*pow(10,2) + i9*pow(10,1));
```

Pág. 05 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

```
  i11 = (*cpf/pow(10,0))-(i1*pow(10,10) + i2*pow(10,9) + i3*pow(10,8) +
  i4*pow(10,7) + i5*pow(10,6) + i6*pow(10,5) + i7*pow(10,4) + i8*1000 +
  i9*100 + i10*10);
  r = (i1*10)+(i2*9)+(i3*8)+(i4*7)+(i5*6)+(i6*5)+(i7*4)+(i8*3)+(i9*2);
  resto = r % 11;
  if(resto < 2)
    d1 = 0;
  else
    d1 = 11 - resto;

  r = (i1*11)+(i2*10)+(i3*9)+(i4*8)+(i5*7)+(i6*6)+(i7*5)+(i8*4)+(i9*3)+(d1*2);
  resto = r % 11;
  if(resto < 2)
    d2 = 0;
  else
    d2 = 11 - resto;

  printf("CPF: %d%d%d.%d%d%d.%d%d%d-%d%d\n",
    i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11);
  printf("d1 = %d\n", d1);
  printf("d2 = %d\n", d2);
  if((d1 == i10) && (d2 == i11))
    printf("CPF válido!!!\n");
  else
    printf("> CPF inválido!!!\n");
}

int main(void) {
  long long int *cpfs=NULL;
  int n, i;

  scanf("%d", &n);

  cpfs = malloc(sizeof(long long int) * n);
  for(i=0; i<n; i++)
    scanf("%lld", cpfs+i);

  for(i=0; i<n; i++)
    valida_cpfs(cpfs+i);

  return 0;
}
```

Pág. 06 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

3 -) (3,0) Utilizando a linguagem C, escreva um programa que leia um arquivo texto, uma linha por vez, e imprima cada linha com os caracteres em ordem invertida. Por exemplo, se o arquivo contiver

```
abc
defghijkl

mnopq
```

a saída deverá ser

```
cba
lkjihgfed

qponm
```

Considere que:

- todas as linhas do arquivo de entrada são terminadas por `\n`;
- cada linha contém no máximo 255 caracteres; e
- o nome do arquivo deve ser passado como argumento na linha de comando.

**RESPOSTA**:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>             /* strlen() */

#define MAXLEN 255

char s[MAXLEN + 1];

int main(int argc, char *argv[])
{
    FILE *fp;
    char *rev;

    if (argc > 1) {
        fp = fopen(argv[1], "r");
        if (fp == NULL) {
            /* precisa tratar NULL, mas exibir o erro é opcional */
            fprintf(stderr, "%s: erro ao abrir %s\n", argv[0], argv[1]);
            exit(1);
        }
    }

    while (!feof(fp)) {
        (void) fgets(s, MAXLEN, fp);
        if (!feof(fp)) {
            rev = s + strlen(s) - 2;
            for (; rev >= s; rev--)
                putchar(*rev);
            putchar('\n');
        }
    }
    fclose(fp);
    return 0;
}
```

Pág. 07 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

4 -) (1,0) Um aluno apresenta o código abaixo, que implementa uma função na linguagem C, mas contém um erro. Com base na análise do código, responda aos itens a seguir.

```
1 int str42(char *s1, const char *s2) {
2     if ((s1 == NULL) || (s2 == NULL))
3         return 1;
4
5     while(*s2 != '\0') {
6         *s1++ = *s2++;
7     }
8
9     return 0;
10 }
```

(a) Descreva a finalidade da função apresentada, descrevendo a lógica utilizada para produzir o resultado esperado.

(b) Identifique o erro presente no código, apresente a correção necessária e justifique sua resposta.

**RESPOSTA**:

a) A função copia os caracteres de `s2` para `s1` até encontrar o terminador nulo de `s2`. Retorna 1 caso algum dos ponteiros seja NULL e 0 caso contrário. A função percorre a string `s2`, copiando cada caractere para a posição correspondente em `s1`, até encontrar o terminador nulo. O qualificador `const` indica que a função não deve modificar os caracteres da string apontada por `s2`, pois ela é utilizada apenas como origem da cópia.

b) A função não copia o terminador nulo `'\0'`, fazendo com que a string resultante não seja devidamente terminada. Para corrigir o erro, deve-se adicionar, após o laço (na Linha 8), a seguinte instrução:

```
*s1 = '\0';
```

Presidente da Banca Examinadora

Pág. 08 de 08 - Documento assinado digitalmente. Para conferência, acesse o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo UDESC 00044704/2026 e o código 8ZFJ145P.

# Assinaturas do documento

Código para verificação: **8ZFJ145P**

Este documento foi assinado digitalmente pelos seguintes signatários nas datas indicadas:

**YURI KASZUBOWSKI LOPES** (CPF: 063.XXX.259-XX) em 09/11/2026 às 08:52:24
Emitido por: "SGP-e", emitido em 02/06/2021 - 13:17:43 e válido até 02/06/2121 - 13:17:43.
(Assinatura do sistema)

Para verificar a autenticidade desta cópia, acesse o link https://portal.sgpe.sea.sc.gov.br/portal-externo/conferencia-documento/VURFU0NfMTIwMjJfMDAwNDQ3MDRfNDQ3MTRfMjAyNl84WkZKMTQ1UA== ou o site https://portal.sgpe.sea.sc.gov.br/portal-externo e informe o processo **UDESC 00044704/2026** e o código **8ZFJ145P** ou aponte a câmera para o QR Code presente nesta página para realizar a conferência.