

*PROVA DE RECUPERAÇÃO*  
**MAC 2166 – INTRODUÇÃO À COMPUTAÇÃO (ELÉTRICA E COMPUTAÇÃO)**  
***PRIMEIRO SEMESTRE DE 2026***

Nome completo: \_\_\_\_\_

NUSP: \_\_\_\_\_ Turma: \_\_\_\_\_

Assinatura: \_\_\_\_\_

**Instruções:**

1. Não destaque as folhas deste caderno.
2. A prova pode ser feita a lápis. Cuidado com a legibilidade.
3. A prova consta de **10 páginas** (incluindo a atual), totalizando **3 questões**. Verifique, antes de começar a prova, se o seu caderno de questões está completo.
4. Preencha o cabeçalho acima e os campos NUSP e Turma das folhas seguintes.
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é permitido o uso de artefatos eletrônicos de qualquer natureza.
7. Não é permitida a consulta a livros, apontamentos ou colegas.
8. Não é necessário apagar rascunhos no caderno de questões.

**DURAÇÃO DA PROVA: 2 horas**

| Questão | Nota |
|---------|------|
| 1       |      |
| 2       |      |
| 3       |      |
| Total   |      |

**Q1. [3 pontos]** Considere o seguinte código.

```
#include <stdio.h>

void digitos(int N, int d, int digs[d], int *n_digs) {
    int i = 0;
    while (N > 0) {
        digs[i] = N % 10;
        N /= 10;
        i++;
    }
    *n_digs = i;
}

void start(int N, int k, int *i, int *j) {
    if (k < N) {
        *i = 0;
        *j = k;
    } else {
        *i = k - N + 1;
        *j = N - 1;
    }
}

int n_elementos(int N, int k) {
    if (k < N)
        return k + 1;
    return 2 * N - k - 1;
}

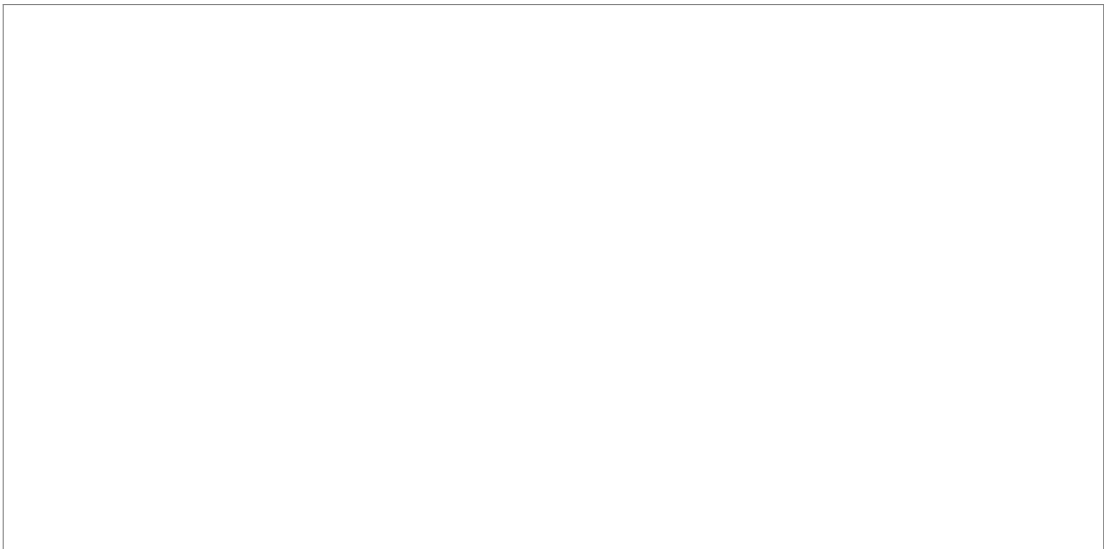
int main(int argc, char *argv[]) {
    int i, j;
    for (int k = 0; k < 2 * 3 - 1; k++) {
        start(3, k, &i, &j);
        printf("(%d, %d; %d) ", i, j, n_elementos(3, k));
    }
    printf("\n");
    int NUSP, N;
    scanf("%d %d", &NUSP, &N);
    int digs[10], D;
    digitos(NUSP, 10, digs, &D);
    int M[N][N], p = 0;
    for (int k = 0; k < 2 * N - 1; k++) {
        start(N, k, &i, &j);
        for (int t = 0; t < n_elementos(N, k); t++) {
            M[i][j] = digs[p];
            i++;
            j--;
            p = (p + 1) % D;
        }
    }
}
```

```
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < N; j++)  
            printf("%d ", M[i][j]);  
        printf("\n");  
    }  
    return 0;  
}
```

Determine a saída impressa pelo programa acima, supondo que a entrada é seu número USP, seguido do número 4. Por exemplo, se seu número fosse 13579246, a entrada seria

13579246 4

*Saída:*



*Rascunho:*

**NUSP:** \_\_\_\_\_ **Turma:** \_\_\_\_\_

**Q2. [3 pontos]** Nesta questão, dizemos que dois números inteiros positivos são *anagramas* um do outro se um deles pode ser transformado no outro permutando seus dígitos. Por exemplo, 112233 e 123123 são anagramas um do outro e 246533 e 326435 são anagramas um do outro, mas 10230 e 2031 não são anagramas um do outro.

(a) Escreva inicialmente uma função

```
void conta(int N, int c[10])
```

que recebe um inteiro positivo  $N$  e um vetor  $c$  com dez posições, e que preenche  $c[d]$  com o número de vezes que o dígito  $d$  ocorre em  $N$ .

(b) Escreva agora uma função

```
bool anagrama(int x, int y)
```

que recebe dois números inteiros positivos  $x$  e  $y$ , e que devolve `true` se  $x$  e  $y$  são anagramas um do outro e devolve `false` caso contrário.

- (c) Com as funções dos itens (a) e (b) em mãos, escreva um programa que executa a seguinte tarefa. Seu programa deve ler da entrada padrão uma sequência de inteiros positivos  $x_0, x_1, \dots$  até encontrar o término da sequência (EOF). É garantido que a sequência terá no máximo 10000 elementos. Seu programa deve imprimir alguns dos  $x_i$  lidos e deve omitir outros: (1) os  $x_i$  impressos não podem conter dois números que são anagramas um do outro, e (2) *todo* elemento da sequência original deve ou ele mesmo aparecer na saída, ou algum anagrama dele deve aparecer na saída.

**Exemplo.** Se a entrada de seu programa for

```
222 220 101 221 12 211 111 212 10 102 220 210 112 120 112 222 22
```

a saída de seu programa poderia ser

```
222 220 101 221 12 211 111 10 102 22
```

Escreva aqui a função `main` de seu programa. Não é necessário repetir as funções que você escreveu em (a) e (b), mas seu programa deve, obrigatoriamente, basear-se nelas.

NUSP: \_\_\_\_\_ Turma: \_\_\_\_\_

**Q3. [4 pontos]** Prediolândia é uma cidade com muitos prédios, dispostos no formato de uma grade  $N \times N$ , descrita por uma matriz  $A$  com tais dimensões. O prédio na linha  $i$  e coluna  $j$  possui  $a_{ij}$  andares, onde  $a_{ij}$  é um inteiro positivo. A linha de índice  $N - 1$  da grade consiste dos prédios mais ao sul, e a coluna de índice 0 consiste dos prédios mais ao oeste.

O  $k$ -ésimo andar do prédio  $P$  é *visível pelo sul* se não existe nenhum outro prédio que tem pelo menos  $k$  andares, está na mesma coluna que  $P$  e está mais ao sul que  $P$ . Similarmente, o  $k$ -ésimo andar do prédio  $P$  é *visível pelo oeste* se nenhum outro prédio que está na mesma linha que  $P$  e está mais ao oeste que  $P$  tem  $k$  ou mais andares. Um andar de  $P$  é *visível* se é visível pelo sul ou pelo oeste. Por exemplo, considere a cidade dada por  $N = 3$  e pela matriz

$$A = \begin{pmatrix} 6 & 8 & 2 \\ 4 & 3 & 9 \\ 1 & 5 & 2 \end{pmatrix}.$$

O prédio de coordenadas  $(0, 1)$  tem 8 andares. Desses,

- os três andares mais altos são visíveis pelo sul, e o prédio de altura 5 (que está mais ao sul e na mesma coluna) bloqueia os demais andares.
- os dois andares mais altos são visíveis pelo oeste, e o prédio de altura 6 (que está mais ao oeste e na mesma linha) bloqueia todos os demais.
- ou seja, três andares do prédio de altura 8 são visíveis.

Nesse exemplo, o prédio que tem mais andares visíveis é o que está na posição  $(1, 2)$ . Ele tem 9 andares, 7 dos quais visíveis.

(a) Escreva uma função

```
int num_andares(int N, int A[N][N], int pi, int pj, char dir)
```

que recebe  $N$ , a matriz  $A$  que descreve a cidade, e as coordenadas  $(pi, pj)$  de um prédio. Se o caractere `dir` for igual a `S`, você deve devolver quantos andares desse prédio são visíveis pelo sul; se o caractere `dir` for igual a `O`, você deve devolver quantos andares desse prédio são visíveis pelo oeste.

(b) Escreva uma função

```
int mais_visivel(int N, int A[N][N], int *pi, int *pj)
```

que calcula a posição  $(p_i, p_j)$  do prédio que tem mais andares visíveis. Sua função deve também devolver tal número máximo de andares visíveis. Caso haja vários prédios com o máximo número de andares visíveis, você pode calcular a posição de qualquer um deles.

NUSP: \_\_\_\_\_ Turma: \_\_\_\_\_

- (c) Escreva a função `main` do seu programa. Ela deve ler o valor de  $N$  e as entradas da matriz  $N \times N$  que representa a cidade, e deve imprimir uma linha com três inteiros `pi pj M`, onde  $(p_i, p_j)$  é a posição de um prédio com o maior número de andares visíveis, e  $M$  é tal número máximo de andares visíveis. Caso haja vários prédios com  $M$  andares visíveis, você pode imprimir a posição de qualquer um deles.

Use, obrigatoriamente, as funções dos itens (a) e (b) acima, mesmo que você não tenha resolvido aqueles itens.

(Rascunho)

**NUSP:** \_\_\_\_\_ **Turma:** \_\_\_\_\_

(Rascunho)