MAC2166 – Introdução à Computação
Escola Politécnica
Prova Substutiva – 30 de junho de 2015

Questão 1 (valor: 3 pontos)

Uma sequência de n números inteiros não nulos é dita piramidal m-alternante se é constituída por m segmentos: o primeiro com um elemento, o segundo com dois elementos e assim por diante até o m-ésimo, com m elementos. Além disso, os elementos de um mesmo segmento devem ser todos pares ou todos ímpares e para cada segmento, se seus elementos forem todos pares (ímpares), os elementos do segmento seguinte devem ser todos ímpares (pares).

Por exemplo, a sequência com $n = 10$ elementos:

12 3 7 2 10 4 5 13 5 11 é piramidal 4-alternante.

A sequência com $n = 3$ elementos:

7 10 2 é piramidal 2-alternante.

A sequência com $n = 8$ elementos:

1 12 4 3 13 5 12 6 não é piramidal alternante pois o último segmento não tem tamanho 4.

a) Escreva uma função `bloco` que recebe como parâmetro um inteiro n e lê n inteiros do teclado, devolvendo um dos seguintes valores:

- 0, se os n números lidos forem pares;
- 1, se os n números lidos forem ímpares;
- -1, se entre os n números lidos há números com paridades diferentes.

Solução

```
int bloco(int n)
{
    int num, tem_par = 0, tem_impar = 0;

    while (n > 0)
    {
        scanf("%d", &num);
        if (num % 2 == 0)
            tem_par = 1;
        else
            tem_impar = 1;

        n--;
    }

    if (tem_par && !tem_impar)
        return 0;
    if (tem_impar && !tem_par)
        return 1;

    return -1;
}
```

b) Usando a função do item anterior, escreva um programa que, dados um inteiro $n > 1$ e uma sequência de n números inteiros, verifica se ela é piramidal m-alternante. O programa deve imprimir o valor de m ou dar a resposta não.

Solução

```
int main()
{
    int n, m = 1, paridade_ant, paridade, eh_piramidal_alternante = TRUE;

    scanf("%d", &n);
    paridade_ant = bloco(1);
    n--;

    while (n > 0)
    {
        m++;
        if (m > n) {
            m = n;
            eh_piramidal_alternante = FALSE;
        }

        paridade = bloco(m);
        if (paridade == -1 || paridade == paridade_ant)
            eh_piramidal_alternante = FALSE;

        paridade_ant = paridade;

        n = n - m;
    }

    if (eh_piramidal_alternante == TRUE)
        printf("A sequencia e' piramidal %d-alternante\n", m);
    else
        printf("A sequencia nao e' piramidal m-alternante.\n");

    return 0;
}
```

Questão 2 (valor: 3 pontos)

Nesta questão você deve escrever uma função que recebe uma *string* e retorna como resultado uma nova *string* contendo a maior palavra da *string* original. Por exemplo, a frase abaixo contém 12 palavras, onde a maior delas é ‘‘`enobrecem`’’ :

‘‘`Os EPs enobrecem os alunos assim como o trabalho enobrece os homens`’’

Sua função deve receber apenas **UM** parâmetro, `frase`, que deve ser uma *string* (não será dado o tamanho da *string*!). A função deve devolver uma nova *string* **alocada dinamicamente** contendo a maior palavra encontrada em `frase`. Para facilitar, considere que `frase` contém apenas caracteres de letras e espaço, portanto as palavras aparecem sempre separadas por um espaço em branco.

Sugestão: Faça uma função auxiliar `acha_proxima`, que recebe como parâmetro um índice i e uma *string* e retorna o início e o tamanho da primeira palavra que aparece a partir da posição i da *string*.

Lembre-se de que uma *string* é um vetor de caracteres que contém 0 (= ‘\0’) na última posição.

Resolução

```
void acha_proxima(int i, char *frase, int *inicio, int *tam)
{   /* pula os espacos em branco ate encontrar o inicio da proxima palavra */
    while (frase[i] != '\0' && frase[i] == ' ')
        i++;

    *inicio = i;
    *tam = 0;

    while (frase[i] != '\0' &&  frase[i] != ' ')
    {   /* conta o numero de caracteres na palavra */
        i++;
        (*tam)++;
    }
}

char *maior_palavra(char *frase)
{
    char *palavra;
    int i = 0, inicio_prox_palavra, tam_prox_palavra,
        inicio_maior_palavra = 0, tam_maior_palavra = 0;

    while (frase[i] != '\0')
    {
        acha_proxima(i, frase, &inicio_prox_palavra, &tam_prox_palavra);
        if (tam_prox_palavra > tam_maior_palavra)
        {
            tam_maior_palavra = tam_prox_palavra;
            inicio_maior_palavra = inicio_prox_palavra;
        }
        i = inicio_prox_palavra + tam_prox_palavra;
    }

    /* Copia para a string a maior palavra da frase */
    palavra = (char *) malloc ((tam_maior_palavra+1) * sizeof(char));
    for (i = 0; i < tam_maior_palavra; i++)
        palavra[i] = frase[inicio_maior_palavra+i];

    palavra[i] = '\0';
    return palavra;
}
```

Questão 3 (valor: 4 pontos)

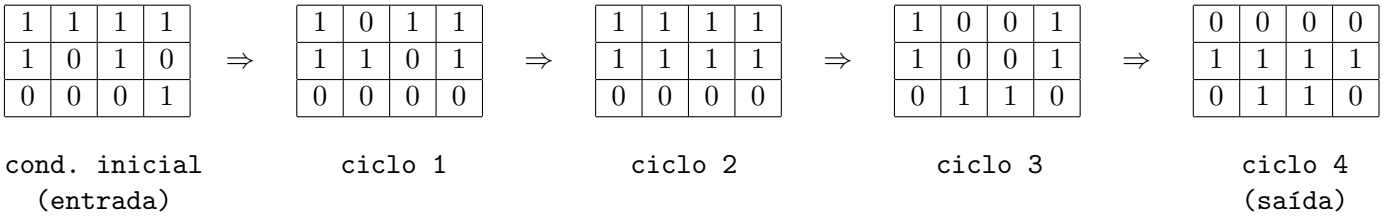
O *Jogo da Vida* é uma simulação clássica de uma colônia de bactérias. O meio de cultura é representado por uma matriz A $m \times n$, em que cada posição contém um valor indicando se ali há uma bactéria (1) ou não (0).

Você deve fazer um programa que lê a condição inicial do meio de cultura (ou seja, a matriz A) e um número inteiro C para em seguida calcular e exibir o resultado final de C ciclos do *Jogo da Vida*. A cada ciclo, as posições da matriz devem ser atualizadas simultaneamente com base nos valores da matriz do jogo no ciclo anterior:

- Cada bactéria com menos de 2 bactérias vizinhas morre (de "solidão");
- Cada bactéria com 2 ou 3 bactérias vizinhas continua viva;
- Cada bactéria com mais de 3 bactérias vizinhas morre (de claustrofobia);
- Cada posição vazia com exatamente 3 bactérias vizinhas passa a ter uma bactéria (por reprodução).

Posições (i, j) e (i', j') distintas são vizinhas se $|i - i'| \leq 1$ e $|j - j'| \leq 1$. Logo, uma posição tem no máximo 8 vizinhos.

Exemplo de aplicação das regras de atualização para um meio de cultura em uma matriz 3×4, com $C = 4$:



Você deve implementar e usar funções para:

- Leitura dos dados de entrada;
- Cálculo de um novo ciclo;
- Contagem do número de bactérias vizinhas de uma posição (i, j);

Para a impressão do resultado, basta escrever o protótipo de uma função e usá-la (ou seja, não é necessário implementá-la).

Resolução

```
#include <stdio.h>
#define MAX 100
#define BACTERIA 1
#define VAZIO 0

void exibe_colonia(int mat[][MAX], int nlin, int ncol);

void le_condicao_inicial(int mat[][MAX], int *nlin, int *ncol)
{
    int i, j;

    scanf("%d %d", nlin, ncol);
    for (i = 0; i < *nlin; i++)
        for (j = 0; j < *ncol; j++)
        {
         printf("Digite o elemento da posicao (%d,%d) da matriz: ", i+1, j+1);
            scanf("%d", &mat[i][j]);
        }
}

int cont_bacterias_vizinhas(int mat[][MAX], int nlin, int ncol, int i, int j)
{
    int k, l, cont = 0;

    for (k = i - 1; k <= i + 1; k++)
        for (l = j - 1; l <= j+1; l++)
            if (k >= 0 && k < nlin && l >= 0 && l < ncol && mat[k][l] == BACTERIA)
                cont++;
    if (mat[i][j] == BACTERIA)
        cont--;

    return cont;
}

void calcula_novo_ciclo(int mat[][MAX], int nlin, int ncol)
{
    int copia[MAX][MAX], i, j, cont;

    for (i = 0; i < nlin; i++)
        for (j = 0; j < ncol; j++)
            copia[i][j] = mat[i][j];

    for (i = 0; i < nlin; i++)
        for (j = 0; j < ncol; j++)
        {
            cont = cont_bacterias_vizinhas(copia, nlin, ncol, i, j);
            if (mat[i][j] == BACTERIA && (cont < 2 || cont >= 4) )
                mat[i][j] = VAZIO;
            else if (mat[i][j] == VAZIO && cont >= 3)
                 mat[i][j] = BACTERIA;
        }
}
```

```
int main()
{
    int mat_colonia[MAX][MAX], nlinhas, ncolunas, C;

    printf("Digite a matriz com a condicao inicial da colonia: ");
    le_condicao_inicial(mat_colonia, &nlinhas, &ncolunas);

    printf("Digite o numero de ciclos: ");
    scanf("%d", &C);

    while (C > 0)
    {
        calcula_novo_ciclo(mat_colonia, nlinhas, ncolunas);
        C--;
    }

    printf("Condicao final: \n");
    exibe_colonia(mat_colonia, nlinhas, ncolunas);

    return 0;
}
```