Instituto Federal do Espírito Santo — IFES (Campus Serra)
Coordenadoria de Informática — Cin

Aluno(a): _____
Disciplina: Estrutura de dados
Valor: 25 pontos
Professor: Thiago Meireles Paixão (`thiago.paixao@ifes.edu.br`)
Data: 16 de outubro de 2023

Prova 1

Responder as questões nas folhas de papel almaço. Você têm 0,5 pontos extras na prova graças ao Marlon. Há 3 pontos extras nessa prova, porém a nota máxima considerada será de 25 pontos.

Pontuação					
1 (5 pts.)	2 (3 pts.)	3 (3 pts.)	4 (6 pts.)	5 (8 pts.)	6 (3 pts.)

1. (5 pontos) Programação em C – Uma matriz triangular inferior é um tipo especial de matriz quadrada em álgebra linear e matemática que possui todos os elementos acima da diagonal principal iguais a zero. A diagonal principal e todos os elementos abaixo dela podem ter valores distintos de zero, conforme exemplo seguir:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ -4 & 5 & 6 & 0 \\ 7 & 8 & 0 & 10 \end{bmatrix}.$$

   Complete o Código 1 (em C) cujo objetivo é alocar memória para uma matriz triangular inferior de inteiros de dimensões $n \times n$. Para economia de memória, seu programa DEVE alocar espaço apenas para os valores que não estão acima da diagonal principal, ou seja, 1 elemento para a primeira linha, 2 elementos para a segunda, 3 elementos para a terceira linha e assim em sucessivamente.

   Observação: Não se preocupe em preencher os valores, apenas em alocar o espaço.

Código 1: Código base (questão 1).

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

int main() {
   int n;

   /**
    * Seu código aqui.
    */

   return 0;
}
```

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

int main() {
    int n;
    printf("Digite n: ");
    scanf("%d", &n);

    // Aloca a matriz dinamicamente
    int **matriz = (int **)malloc(m * sizeof(int *));
    for (int i = 0; i < m; i++) {
        matriz[i] = (int *)malloc((i+1)) * sizeof(int));
    }

    return 0;
}
```

2. (3 pontos) Programação em C – Qual o tamanho (em bytes) da matriz alocada na questão anterior? Forneça sua resposta em termos das variáveis que descrevem as dimensões da matriz.

   Observação: Pode ser útil o seguinte resultado sobre a soma dos elementos de uma progressão aritmética $(a_1, a_2, a_3, \ldots, a_n)$:

$$S_n = a_1 + a_2 + a_3 + \ldots + a_n = \frac{n(a_1 + a_n)}{2}.$$

$$\#\text{bytes} = \frac{n(1+n)}{2} \times 4 = 2n^2 + 2n$$

3. (3 pontos) Vetores dinâmicos – No uso do TAD vetor dinâmico (Código 2), deve-se escolher a capacidade inicial do vetor (`n_max`). Eventualmente, será requisitado a inserção de um elemento quando a ocupação do vetor está no limite (isto é, `n = n_max`). O que deve ocorrer para que a inserção possa ser realizada? Sobre a escolha da capacidade inicial do vetor dinâmico: comente o problema em se escolher um valor muito baixo ou valor muito alto.

Código 2: TAD Dynvec.

```c
typedef struct dynvec DynVec;
struct dynvec
{
  int n; // número de elementos
  int n_max; // capacidade do vetor
  float *v; // dados
};
```

Nessa situação, a capacidade do vetor deve ser aumentada utilizando realocação de memória, em geral, multiplicando-se por um fator inteiro α. Um valor inicial baixo de capacidade do vetor (i.e., valor inicial de `n_max`) pode levar a frequentes realocações de memória, sendo este processo custoso em termos de desempenho. Por outro lado, se a capacidade inicial for muito alta, o vetor ocupará mais memória desde o início, o que pode resultar em desperdício de recursos.

4. (6 pontos) TADs – Nesta questão, o objetivo é implementar um TAD em C que representa um círculo contendo as informações de seu raio e centro no plano cartesiano. Para isso, implemente

a estrutura `circle` e mais duas funções (Código 3): uma para inicializar a estrutura e outra para verificar se há interseção entre dois círculos.

Observação: Dois círculos se instersectam se a distância entre os seus centros é menor ou igual à soma dos seus raios.

Código 3: circle.h.

```c
// Definição do tipo
typedef struct circle Circle;

// Inicializa círculo alocando espaço e inicializando seus campos internos.
Circle* init_circle(float x, float y, float radius);

// Verifica interseção entre dois círculos (1 - sim, 0 - não)
int intersect(Circle* c1, Circle* c2);
```

Código 4: circle.c.

```c
#include <circle.h>
#include <math.h>

// Implementação do tipo
struct circle {
/**
 * Seu código aqui.
 */
};

// Inicializa círculo alocando espaço e inicializando seus campos internos
Circle* init_circle(float x, float y, float radius) {
/**
 * Seu código aqui.
 */
}

// Verifica interseção entre dois círculos (1 - sim, 0 - não)
int intersect(Circle* c1, Circle* c2) {
/**
 * Seu código aqui.
 */
}
```

```c
// Implementação do tipo
struct circle {
    float x; // coordenada x do centro
    float y; // coordenada y do centro
    float radius; // raio do círculo
};

// Inicializa círculo alocando espaço e inicializando seus campos internos
Circle* init_circle(float x, float y, float radius) {
    Circle* circle = malloc(sizeof(Circle));
    circle->x = x;
```

```
    circle->y = y;
    circle->radius = radius;
    return circle;
}

// Verifica interseção entre dois círculos (1 - sim, 0 - não)
int intersect(Circle* c1, Circle* c2) {
    float distance = sqrt(pow(c2->x - c1->x, 2) + pow(c2->y - c1->y, 2));
    return distance <= c1->radius + c2->radius;
}
```

5. (8 pontos) Listas encadeadas – Considere os tipos `List` e `ListNode` descritos no Código 5. Os valores `w` e `h` da estrutura `ListNode` denotam a base e a altura de um retângulo. Escreva uma função que percorre a lista e retorna um ponteiro para um nó da lista com o retângulo de maior área. Se a lista estiver vazia, sua função deve retornar `NULL`.

Código 5: TAD List.

```
typedef struct list {
   ListNode *first;
} List;

typedef struct list_node {
   float w, h;
   ListNode *next;
} ListNode;
```

```
ListNode* max_area_rectangle(List *l) {
    float area, max_area = l->first->h * l->first->w;
    ListNode *ans = l->first;
    for(ListNode *p = l->first->next; p != NULL; p = p->next) {
        area = p->h * p->w;
        if(area > max_area) {
            max_area = area;
            ans = p;
        }
    }
    return ans;
}
```

6. (3 pontos) Listas encadeadas – Considere o tipo lista de inteiros definido em nossos estudos. Complete as funções descritas no Código 6 cujo objetivo é calcular RECURSIVAMENTE o primeiro elemento par da lista. Assuma que a lista possui pelo menos 1 elemento par.

Código 6: Função first_even.

```
// Retorna o primeiro elemento par da lista
int first_even(List *l) {
   return _first_even(...);
}

// Função recursiva associada a first_even
static int _first_even(ListNode *p) {
/**
 * Seu código aqui.
```

```
 */
}
```

```
int first_even(List *l) {
    return _first_even(l->first);
}

// Função recursiva associada a largest
static int _first_even(ListNode *p) {
    // caso a sublista estiver vazia, retorne -1 (na presente questão, isso não ocorrerá p
    if(p == NULL)
        return -1;

    // se o primeiro elemento for par, retorne ele
    if (p->info % 2 == 0)
        return p->info;

    // senão, retorne o primeiro par da sublista iniciada no próximo nó
    return _first_even(p->next);
}
```