Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
CSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini

(*) 1.

a) Escreva uma função que recebe 3 `float`'s e retorna uma `struct` que tem 3 campos do tipo `float`: menor, maior e meio. A função deve analisar os valores preencher a `struct`.

```
MaMeMe classifica (float n1, float n2, float n3);
```

b) Reescreva a função anterior para que ela ordene os valores já colocados em uma variável passada por referência. Os valores são dados em uma ordem arbitrária, e reordenados pela função.

```
void classifica (MaMeMe* valores);
```

(***) 2. Considere a seguinte declaração do tipo `GSImage`:

```
typedef struct
{
    int largura;
    int altura;
    unsigned char** dados;
} GSImage;
```

Uma variável do tipo `GSImage` representa uma imagem em escala de cinza, onde cada pixel é representado por um byte em uma matriz de dados. O valor do byte indica a intensidade do pixel: quanto mais alto o valor, mais claro é o pixel, de forma que o valor 0 representa um pixel preto e o valor 255 representa um pixel branco.

Suponha que a função abaixo é usada para desalocar toda a memória associada a uma variável do tipo `GSImage`:

```
void destroiGSImage (GSImage* img)
{
    int i;
    for (i = 0; i < img->altura; i++)
        free (img->dados [i]);
    free (img->dados);
    free (img);
}
```

a) Escreva uma função que aloca a memória para uma variável do tipo `GSImage`. Esta função deve alocar dinamicamente a memória para a variável e sua matriz de dados, preencher os campos da `struct` adequadamente, e retornar um ponteiro para o espaço alocado. O protótipo da função deve ser:

```
GSImage* criaGSImage (int largura, int altura);
```

Nota: as funções `destroiGSImage` e `criaGSImage` têm papel análogo a um destruidor e um construtor no paradigma orientado a objetos.

b) Escreva uma função que cria uma versão reduzida de uma imagem dada. A função deve criar uma nova imagem (usando a função `criaGSImage`), com largura e altura reduzidas pela metade. Cada

grupo de 4 pixels na imagem original é mapeado como 1 pixel na imagem reduzida. O valor do pixel é igual à média dos 4 pixels na imagem original (não se preocupe com arredondamentos). Por exemplo, a posição (0, 0) da imagem reduzida receberá a média dos valores das posições (0, 0), (0, 1), (1, 0) e (1, 1) da imagem original; e a posição (0, 1) da imagem reduzida receberá a média dos valores das posições (0, 2), (0, 3), (1, 2) e (1, 3) da imagem original. A função tem o seguinte protótipo:

```
GSImage* reduzPelaMetade (GSImage* img);
```

A função retorna um ponteiro para a imagem reduzida criada. Para evitar erros com acessos a posições inválidas de memória, a largura e a altura da imagem original devem ser ambas divisíveis por 2. Se isso não ocorrer, a função deve retornar `NULL`, sem realizar quaisquer outras ações.

c) Escreva uma função sem parâmetros nem valor de retorno, que cria uma imagem (usando a função `criaGSImage`) e preenche a sua matriz de dados de forma que cada pixel em uma posição (i, j) receba o valor de i*j. As dimensões da imagem são dadas por duas macros `LARG` e `ALT`, que você pode pressupor que já foram definidas. A função deve então criar uma versão reduzida da imagem (usando a função `reduzPelaMetade`). Depois disso, a função deve computar e mostrar as médias dos valores dos pixels de cada imagem, separadamente. Para as médias, use variáveis do tipo `float`, evitando que seus valores sejam truncados. Lembre-se de evitar vazamentos de memória!

(***) 3. Suponha que um concurso público adotou o modelo de prova conhecido como “verdadeiro ou falso da morte”. Uma prova deste tipo contém um certo número de questões, sendo que cada questão aborda um tópico diferente. Todas as questões têm o mesmo valor, e a soma total dos valores é igual a 100. Cada questão tem um número de itens, e cada item deve ser respondido com V ou F. Todos os acertos têm o mesmo valor, mas um item com resposta incorreta anula uma resposta certa. Desta forma, se um candidato errar a metade dos itens de uma questão ou mais, aquela questão terá valor 0. Entretanto, o candidato tem a opção de deixar de responder um item – neste caso, ele não ganha o valor daquele item, mas não ocorre qualquer outro desconto.

Por exemplo, se a prova tiver 4 questões, cada uma valerá 25 pontos. Se cada questão tiver 5 itens, cada item valerá 5 pontos. Suponha que os resultados de um candidato foram:

- Questão 1: os 5 itens foram marcados com a resposta correta.
- Questão 2: o candidato marcou corretamente 3 itens, deixando os outros 2 em branco.
- Questão 3: o candidato marcou corretamente 3 itens, deixou 1 em branco, e errou 1 item.
- Questão 4: o candidato marcou corretamente 2 itens, e errou 3.

Neste exemplo, o candidato ganharia 25 pontos pela questão 1, 15 pontos pela questão 2, 10 pontos pela questão 3, e 0 ponto pela questão 4. Sua nota final é, portanto, 50.

Suponha que as provas são corrigidas por um sistema automático, que lê os cartões de resposta e os associa a variáveis de um tipo que representa um candidato. O tipo é declarado da seguinte forma:

```
typedef struct
{
    int numero;
    char* nome;
    char* email;
    char* endereco;
    int** prova;
    float nota;

} Candidato;
```

A matriz `prova` de uma variável do tipo `Candidato` contém as respostas dadas pelo candidato. Cada linha corresponde a uma questão, e cada coluna a um item. Cada posição tem o valor 0, 1 ou -1, representando respectivamente as respostas F, V e “vazio” (quando o candidato opta por não responder à questão).

a) Escreva uma função que corrige a prova de um candidato. A função deve comparar as respostas dadas pelo candidato com um gabarito (codificado da mesma forma que a prova), e calcular sua nota. Suponha que todos os campos do parâmetro dado possuem valores válidos, exceto a nota, que deve ser calculada pela função. A função também recebe como parâmetros o número de questões da prova e o número de itens em cada questão. O seu protótipo é:

```
void corrigeProva (Candidato* c, int** gabarito, int n_questoes, int n_itens);
```

b) A função abaixo recebe como parâmetros um vetor do tipo `Candidato` e o seu tamanho, além do gabarito de uma prova:

```
void computaResultados (Candidato* candidatos, int n, int** gabarito)
{
    int i;
    for (i = 0; i < A; i++)
    {
        corrigeProva (B, C, N_QUESTOES, N_ITENS);
        if (D >= NOTA_FASE2)
            convocaParaFase2 (E, F, G);
    }
}
```

A função percorre o vetor – que já está devidamente preenchido – e invoca a função `corrigeProva` para cada candidato. Quando a nota obtida pelo candidato for maior ou igual a `NOTA_FASE2`, o sistema envia um e-mail para ele, convocando-o para a segunda fase do concurso. O e-mail é enviado por uma função, que tem o protótipo abaixo:

```
void convocaParaFase2 (char* nome, char* email, float nota);
```

Substitua no código da função as letras ***A*** a ***G*** pelo trecho de código apropriado.

(***) 4. Em alguns *games* de estratégia por turnos, cada participante (país, facção, reino, etc.) tem sob seu domínio um certo número de territórios, de forma similar ao jogo de tabuleiro War (Risk). A cada rodada, cada território gera ou consome uma certa quantidade de recursos financeiros. Suponha que em um determinado *game*, um território é descrito pelo seguinte tipo:

```
typedef struct
{
    float lucro_base;
    Benfeitoria* benfeitorias [MAX_BENFEITORIAS];
    int n_benfeitorias;
    Exercito* exercitos [MAX_EXERCITOS];
    int n_exercitos;
    /* ... (outros dados, irrelevantes para a questão) */

} Territorio;
```

O campo `lucro_base` indica a quantidade de recursos gerada pelo território a cada rodada. Este valor pode ser modificado pela construção de benfeitorias (portos, estradas, minas, etc.). Cada território possui um vetor de ponteiros para instâncias do tipo `Benfeitoria`. O tamanho deste vetor é `MAX_BENFEITORIAS`, mas apenas `n_benfeitorias` posições são usadas em um dado momento. Uma instância do tipo `Benfeitoria` é descrita pelo seguinte tipo:

```
typedef struct
{
    float lucro_mult;
    float lucro_soma;
    /* ... (outros dados, irrelevantes para a questão) */

} Benfeitoria;
```

Os dois campos mostrados afetam o lucro total dos territórios aos quais uma benfeitoria está associada. O campo `lucro_mult` é um multiplicador, que é aplicado sobre o lucro base e sobre os multiplicadores das outras benfeitorias no mesmo território (i.e. os multiplicadores das benfeitorias se afetam mutuamente). O seu valor é sempre maior ou igual 1. O campo `lucro_soma` é uma quantia fixa adicionada ao lucro do território, que não é afetada pelos multiplicadores. Este campo pode ter um valor negativo.

Cada território tem também associado um vetor de ponteiros para instâncias do tipo `Exercito`. Novamente, o vetor tem tamanho `MAX_EXERCITOS`, mas apenas `n_exercitos` posições são usadas. Uma instância do tipo `Exercito` é descrita pelo seguinte tipo:

```
typedef struct
{
    float custo;
    /* ... (outros dados, irrelevantes para a questão) */

} Exercito;
```

Para esta questão, o único dado relevante sobre um exército é o seu `custo`, que deve ser subtraído do lucro total a cada rodada.

Suponha que cada participante do jogo é representado pelo tipo abaixo:

```
typedef struct
{
    Territorio* territorios [MAX_TERRITORIOS];
    int n_territorios;
    /* ... (outros dados, irrelevantes para a questão) */

} Faccao;
```

Novamente, temos um vetor de ponteiros para instâncias do tipo `Territorio`, com `MAX_TERRITORIOS` posições, mas das quais apenas `n_territorios` posições são usadas.

A cada rodada de uma partida, o jogo deve calcular qual é o lucro (ou prejuízo) total de cada participante. Apesar da estrutura aparentemente complicada do sistema, com várias relações entre ponteiros, o algoritmo para calcular o lucro total de um participante é simples: basta percorrer o vetor `territorios`, computando o lucro gerado por cada território. Para calcular o lucro gerado por um território, toma-se o seu `lucro_base`, que é multiplicado por todos os `lucro_mult` associados às benfeitorias daquele território. Depois, adicionam-se ao lucro os valores no campo `lucro_soma` das benfeitorias. Por fim, percorre-se o vetor `exercitos`, subtraindo-se do lucro do território o `custo` de cada exército. Se o custo dos exércitos ou das benfeitorias for muito grande, o lucro do território será negativo, indicando que aquele território não gera lucro, mas prejuízo!

Escreva uma função que calcule e retorne o lucro (ou prejuízo) total para um participante do jogo. A função deve ter o seguinte protótipo:

```
float lucroTotal (Faccao* f);
```