1.

a)

```
#include <stdio.h>

typedef struct
{
    float maior;
    float menor;
    float meio;
} MaMeMe;

void troca (float* a, float* b)
{
    float aux = *a;
    *a = *b;
    *b = aux;
}

MaMeMe classifica (float n1, float n2, float n3)
{
    MaMeMe resultado;

    resultado.menor = n1;
    resultado.maior = n2;
    resultado.meio = n3;

    /* Garante que o meio e o maior estão em ordem. */
    if (resultado.meio > resultado.maior)
        troca (&resultado.meio, &resultado.maior);

    /* Garante que o menor e o meio estão em ordem. */
    if (resultado.menor > resultado.meio)
    {
        troca (&resultado.menor, &resultado.meio);
        /* Garante que o meio e o maior estão em ordem (de novo). */
        if (resultado.meio > resultado.maior)
            troca (&resultado.meio, &resultado.maior);
    }

    return (resultado);
}

int main ()
{
    MaMeMe numeros;

    numeros = classifica (30, 20, 10);

    printf ("%.2f, %.2f, %.2f\n", numeros.menor, numeros.meio, numeros.maior);

    return (0);
}
```

b)

```
#include <stdio.h>

typedef struct
{
    float maior;
    float menor;
    float meio;
} MaMeMe;

void troca (float* a, float* b)
{
    float aux = *a;
    *a = *b;
    *b = aux;
}

void classifica (MaMeMe* valores)
{
    // Garante que o meio e o maior estão em ordem.
    if (valores->meio > valores->maior)
        troca (&valores->meio, &valores->maior);

    // Garante que o menor e o meio estão em ordem.
    if (valores->menor > valores->meio)
    {
        troca (&valores->menor, &valores->meio);
        // Garante que b e c estão em ordem (de novo).
        if (valores->meio > valores->maior)
            troca (&valores->meio, &valores->maior);
    }
}

int main ()
{
    MaMeMe numeros;

    numeros.menor = 30;
    numeros.meio = 20;
    numeros.maior = 10;

    classifica (&numeros);

    printf ("%.2f, %.2f, %.2f\n", numeros.menor, numeros.meio, numeros.maior);

    return (0);
}
```

2.

a)

```
GSImage* criaGSImage (int largura, int altura)
{
    int i;
    GSImage* img;

    /* Aloca espaço para a imagem. */
    img = (GSImage*) malloc (sizeof (GSImage));

    /* Precisa lembrar da largura e da altura. */
    img->largura = largura;
    img->altura = altura;

    /* Aloca a matriz de dados. */
    img->dados = (unsigned char**) malloc (sizeof (unsigned char*) * altura);
    for (i = 0; i < altura; i++)
        img->dados[i] = (unsigned char*) malloc (sizeof(unsigned char)*largura);
    return (img);
}
```

b)

```
GSImage* reduzPelaMetade (GSImage* img)
{
    int i, j;
    GSImage* reduzida;

    /* As dimensões precisam ser divisíveis por 2. */
    if (img->altura % 2 || img->largura % 2)
        return (NULL);

    /* Cria a imagem reduzida. */
    reduzida = criaGSImage (img->largura/2, img->altura/2);

    for (i = 0; i < reduzida->altura; i++)
        for (j = 0; j < reduzida->largura; j++)
            reduzida->dados [i][j] = (img->dados [i*2][j*2] +
                                      img->dados [i*2][j*2+1] +
                                      img->dados [i*2+1][j*2] +
                                      img->dados [i*2+1][j*2+1])/4;

    return (reduzida);
}
```

c)

```
void f ()
{
    int i, j;
    GSImage *img, *reduzida;
    float media1, media2;

    img = criaGSImage (LARG, ALT);

    for (i = 0; i < ALT; i++)
        for (j = 0; j < LARG; j++)
            img->dados [i][j] = i*j;

    reduzida = reduzPelaMetade (img);

    media1 = 0;
    for (i = 0; i < img->altura; i++)
        for (j = 0; j < img->largura; j++)
            media1 += img->dados [i][j];
    media1 /= (img->altura * img->largura);

    media2 = 0;
    for (i = 0; i < reduzida->altura; i++)
        for (j = 0; j < reduzida->largura; j++)
            media2 += reduzida->dados [i][j];
    media2 /= (reduzida->altura * reduzida->largura);

    printf ("m1: %.2f, m2: %.2f\n", media1, media2);

    destroiGSImage (img);
    destroiGSImage (reduzida);
}
```

3.

a)

```
void corrigeProva (Candidato* c, int** gabarito, int n_questoes, int n_itens)
{
    int i, j, acertos, erros;
    float peso_por_item = 100/((float) n_questoes * n_itens);

    /* A nota total começa em 0, e vai aumentando (ou não) para cada questão. */
    c->nota = 0;

    /* Para cada questão, conta o número de erros e acertos. */
    for (i = 0; i < n_questoes; i++)
    {
        acertos = 0;
        erros = 0;

        /* Verifica cada item. */
        for (j = 0; j < n_itens; j++)
        {
            /* Se o valor não for nem 0 nem 1, ignora o item. Do contrário,
              verifica se está igual ou diferente do gabarito. */
            if (c->prova [i][j] == 0 || c->prova [i][j] == 1)
            {
                if (c->prova [i][j] == gabarito [i][j])
                    acertos++;
                else
                    erros++;
            }
        }

        /* O teste é porque uma questão não pode ter pontuação negativa! */
        if (erros < acertos)
            c->nota += (peso_por_item * (acertos - erros));
    }
}
```

b)

A. n
B. &candidatos [i]
C. gabarito
D. candidatos[i].nota
E. candidatos[i].nome
F. candidatos[i].email
G. candidatos[i].nota

4.

```
float lucroTotal (Faccao* f)
{
    int i, j;
    float total = 0;
    float total_t = 0;

    /* Para cada territorio... */
    for (i = 0; i < f->n_territorios; i++)
    {
        Territorio* t = f->territorios [i];

        total_t = t->lucro_base;

        /* Para cada benfeitoria... */
        for (j = 0; j < t->n_benfeitorias; j++)
            total_t *= t->benfeitorias [j]->lucro_mult;

        for (j = 0; j < t->n_benfeitorias; j++)
            total_t += t->benfeitorias [j]->lucro_soma;

        /* Para cada exército... */
        for (j = 0; j < t->n_exercitos; j++)
            total_t -= t->exercitos [j]->custo;

        total += total_t;
    }

    return (total);
}
```