1.

```
char* empacotaString (char* string)
{
    int i;
    char* string_empacotada;
    int tam_string = strlen (string); /* Assume uma string bem formada. */

    /* Precisa de uma posição extra para o '\0'. */
    string_empacotada = (char*) malloc (sizeof (char) * (tam_string + 1));

    /* Copia a string para o "pacote". */
    for (i = 0; i < tam_string; i++)
        string_empacotada [i] = string [i];
     string_empacotada [tam_string] = '\0';

    return (string_empacotada);
}
```

2.

```
char* concatenaStrings (char* s1, char* s2)
{
    int i, pos_concatenada;
    char* concatenada;
    int tam_s1, tam_s2;

    /* Assume strings bem formadas. */
    tam_s1 = strlen (s1);
    tam_s2 = strlen (s2);

    /* Precisa de uma posição extra para o '\0'. */
    concatenada = (char*) malloc (sizeof (char) * (tam_s1 + tam_s2 + 1));

    /* Copia. */
    pos_concatenada = 0;
    for (i = 0; i < tam_s1; i++)
        concatenada [pos_concatenada++] = s1 [i];
    for (i = 0; i < tam_s2; i++)
        concatenada [pos_concatenada++] = s2 [i];
    concatenada [pos_concatenada++] = '\0';

    return (concatenada);
}
```

3. Vou mostrar 3 jeitos diferentes de resolver o problema.

a) Um algoritmo simples:

```
void balanceamentoDeParenteses (char* str, int* parenteses) {
    int i, j;

    for (i = 0; str [i] != '\0'; i++) {
        if (str [i] == '(') /* Parêntese recém-aberto. */
            parenteses [i] = -1;
        else if (str [i] == ')') {
            /* Volta procurando o último parêntese aberto. */
            j = i-1;
            while (j >= 0 && (parenteses [j] != -1 || str [j] != '('))
                j--;

            if (j < 0)
                parenteses [i] = -1;
            else {
                parenteses [i] = j;
                parenteses [j] = i;
            }
        }
        else
            parenteses [i] = 0;
    }
}
```

b) Um algoritmo bem mais complicado:

```
void balanceamentoDeParenteses (char* str, int* parenteses) {
    int i, j, n_abertos;

    for (i = 0; str [i] != '\0'; i++)
        if (str [i] == ')') /* Começa com -1 em todos os ')'. */
            parenteses [i] = -1;

    for (i = 0; str [i] != '\0'; i++) {
        if (str [i] != '(' && str [i] != ')')
            parenteses [i] = 0; /* Não é parêntese. */
        else if (str [i] == '(') {
            /* Procura o ')' correspondente.
             Para isso, conta quantos parênteses abertos tem adiante. */
            n_abertos = 1;
            for (j = i+1; str [j] != '\0'; j++) {
                if (str [j] == '(')
                    n_abertos++;
                else if (str [j] == ')')
                    n_abertos--;

                if (n_abertos == 0) /* Achou. */
                    break;
            }

            if (str [j] != '\0') {
                parenteses [i] = j;
                parenteses [j] = i;
            }
            else
                parenteses [i] = -1;
        }
    }
}
```

c) Um algoritmo computacionalmente mais eficiente. Usamos aqui um conceito muito útil: uma pilha. A ideia é sempre guardar a posição de cada parêntese aberto, e quando achamos um ')', sabemos que ele fecha o último ')' que tínhamos visto (e cuja posição foi a última guardada).

```
void balanceamentoDeParenteses (char* str, int* parenteses)
{
    int i;
    int tam_str; /* Tamanho da string. */
    int n_abertos = 0; /* Número de parenteses abertos não fechados. */
    int* abertos; /* Um vetor onde colocaremos as posições dos parênteses
                     abertos que não foram fechados. */

    /* Descobre o tamanho da string e aloca o vetor. */
    tam_str = 0;
    for (i = 0; str [i] != '\0'; i++)
        tam_str++;
    abertos = (int*) malloc (sizeof (int) * tam_str);

    /* Percorre a string... */
    for (i = 0; i < tam_str; i++)
    {
        /* Parêntese recém-aberto. Lembra da posição dele no vetor. */
        if (str [i] == '(')
            abertos [n_abertos++] = i;
        else if (str [i] == ')')
        {
            if (!n_abertos) /* Não tem nada para fechar!!! */
                parenteses [i] = -1;
            else
            {
                /* Pega a posição do último parêntese aberto e atualiza o
                   vetor de parênteses. */
                n_abertos--;
                parenteses [i] = abertos [n_abertos];
                parenteses [abertos [n_abertos]] = i;
            }
        }
        else
            parenteses [i] = 0;
    }

    /* Ops, tem parênteses que não foram fechados! */
    while (n_abertos > 0)
        parenteses [abertos [--n_abertos]] = -1;

    free (abertos);
}
```

4.

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

#define BUFLEN 1024

int main ()
{
    int termina = 0; /* Flag que diz se é para terminar. */

    /* 3 buffers */
    char string1 [BUFLEN];
    char string2 [BUFLEN];
    char string3 [BUFLEN];

    /* Usamos estes ponteiros para decidir qual buffer aparece onde. */
    char* string_cima = string1;
    char* string_meio = string2;
    char* string_baixo = string3;

    /* Lê as 3 primeiras strings.
       Importante lembrar que a fgets mantém o \n na string. */
    fgets (string1, BUFLEN, stdin);
    if (strlen (string1) == 1) /* Só o \n. */
        return (1);
    fgets (string2, BUFLEN, stdin);
    if (strlen (string2) == 1) /* Só o \n. */
        return (1);
    fgets (string3, BUFLEN, stdin);
    if (strlen (string3) == 1) /* Só o \n. */
        return (1);

    while (!termina)
    {
        system ("cls"); /* Limpa a tela. */

        /* Mostra as strings em ordem. */
        printf ("%s", string_cima);
        printf ("%s", string_meio);
        printf ("%s", string_baixo);

        /* Lé uma nova string. Reaproveita o buffer da string mais velha. */
        fgets (string_cima, BUFLEN, stdin);
        if (strlen (string_cima) == 1) /* Só o \n. */
            termina = 1;
        else
        {
            /* "Desliza" as strings - a do meio vai para cima, a de baixo vai
               para o meio, a de cima vai para baixo. */
            char* aux_ptr = string_cima;
            string_cima = string_meio;
            string_meio = string_baixo;
            string_baixo = aux_ptr;
        }
    }

    return (0);
}
```