Exercício 1: Bubble Sort (com Sentinela)

Este programa ordena as letras de uma string em ordem crescente usando o algoritmo da bolha com sentinela.

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

// Função para remover a quebra de linha do fgets
void remover_newline(char *str) {
    str[strcspn(str, "\n")] = 0;
}

void bubble_sort_sentinela(char *str) {
    int n = strlen(str);
    bool trocou; // Sentinela

    for (int i = 0; i < n - 1; i++) {
        trocou = false;
        for (int j = 0; j < n - 1 - i; j++) {
            if (str[j] > str[j + 1]) {
                // Troca os caracteres
                char temp = str[j];
                str[j] = str[j + 1];
                str[j + 1] = temp;
                trocou = true;
            }
        }
        // Se nenhuma troca ocorreu nesta passada, a string está ordenada
        if (!trocou) {
            break;
        }
    }
}

int main() {
    char string[101]; // Limite de 100 caracteres + \0

    printf("Digite uma string: ");
    fgets(string, sizeof(string), stdin);
    remover_newline(string);

    bubble_sort_sentinela(string);

    printf("String ordenada: %s\n", string);
    return 0;
}
```

Exercício 2: Inserção Ordenada em Lista

Este programa lê N nomes e os insere em uma lista (array 2D de char) de forma que a lista se mantenha sempre ordenada alfabeticamente.

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

#define MAX_NOMES 100
#define MAX_LEN 101

// Função para remover a quebra de linha do fgets
void remover_newline(char *str) {
    str[strcspn(str, "\n")] = 0;
}

int main() {
    char nomes[MAX_NOMES][MAX_LEN];
    char nome_atual[MAX_LEN];
    int n, i, j, qtd_nomes = 0;

    printf("Quantos nomes deseja inserir? ");
    scanf("%d", &n);
    getchar(); // Consome o \n deixado pelo scanf

    for (i = 0; i < n; i++) {
        printf("Digite o %dº nome: ", i + 1);
        fgets(nome_atual, sizeof(nome_atual), stdin);
        remover_newline(nome_atual);

        // Encontra a posição correta para inserção
        int pos = 0;
        while (pos < qtd_nomes && strcmp(nomes[pos], nome_atual) < 0) {
            pos++;
        }

        // Desloca os elementos para a direita para abrir espaço
        for (j = qtd_nomes; j > pos; j--) {
            strcpy(nomes[j], nomes[j - 1]);
        }

        // Insere o nome na posição correta
        strcpy(nomes[pos], nome_atual);
        qtd_nomes++;
    }
```

```
    printf("\n--- Nomes ordenados alfabeticamente ---\n");
    for (i = 0; i < qtd_nomes; i++) {
        printf("%s\n", nomes[i]);
    }

    return 0;
}
```

Exercício 3: Selection Sort (por Tamanho)

Este programa lê N nomes e os ordena pelo tamanho de cada nome (número de caracteres) usando o algoritmo da seleção.

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

#define MAX_NOMES 100
#define MAX_LEN 101

// Função para remover a quebra de linha do fgets
void remover_newline(char *str) {
    str[strcspn(str, "\n")] = 0;
}

void selection_sort_por_tamanho(char nomes[][MAX_LEN], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Encontra o índice da string de menor tamanho
        int indice_menor = i;
        for (int j = i + 1; j < n; j++) {
            if (strlen(nomes[j]) < strlen(nomes[indice_menor])) {
                indice_menor = j;
            }
        }

        // Troca a string da posição 'i' com a de menor tamanho
        if (indice_menor != i) {
            char temp[MAX_LEN];
            strcpy(temp, nomes[i]);
            strcpy(nomes[i], nomes[indice_menor]);
            strcpy(nomes[indice_menor], temp);
        }
    }
}

int main() {
    char nomes[MAX_NOMES][MAX_LEN];
    int n;
```

```
    printf("Quantos nomes deseja inserir? ");
    scanf("%d", &n);
    getchar(); // Consome o \n deixado pelo scanf

    printf("Digite os %d nomes:\n", n);
    for (int i = 0; i < n; i++) {
        fgets(nomes[i], MAX_LEN, stdin);
        remover_newline(nomes[i]);
    }

    selection_sort_por_tamanho(nomes, n);

    printf("\n--- Nomes ordenados por tamanho ---\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", nomes[i]);
    }

    return 0;
}
```

Exercício 4: Ordenação Lexicográfica

Este programa lê N strings e as ordena lexicograficamente (ordem de dicionário), onde maiúsculas antecedem minúsculas. Usamos a função qsort da biblioteca padrão para eficiência.

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

#define MAX_STRINGS 100
#define MAX_LEN 101

// Função de comparação para o qsort
// Compara duas strings lexicograficamente
int comparar_strings(const void *a, const void *b) {
    // A ordem lexicográfica padrão (strcmp) já trata
    // maiúsculas como vindo antes de minúsculas (pela tabela ASCII)
    return strcmp(*(const char **)a, *(const char **)b);
}

// Função para remover a quebra de linha do fgets
void remover_newline(char *str) {
    str[strcspn(str, "\n")] = 0;
}
```

```
int main() {
    int n;
    // Usamos um array de ponteiros para facilitar o qsort
    char *strings[MAX_STRINGS];
    char buffer[MAX_STRINGS][MAX_LEN]; // Buffer para armazenar as strings

    printf("Digite a quantidade de strings (n): ");
    scanf("%d", &n);
    getchar(); // Consome o \n

    printf("Digite as %d strings (uma por linha):\n", n);
    for (int i = 0; i < n; i++) {
        fgets(buffer[i], MAX_LEN, stdin);
        remover_newline(buffer[i]);
        strings[i] = buffer[i]; // Ponteiro aponta para a string no buffer
    }

    // Ordena o array de ponteiros
    qsort(strings, n, sizeof(char *), comparar_strings);

    printf("\n--- Strings ordenadas ---\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", strings[i]);
    }

    return 0;
}
```

Exercício 5: Ordenação por Múltiplas Chaves

Este programa lê N entradas, cada uma com 3 strings. Ele ordena as entradas usando string1 como chave primária, string2 como secundária e string3 como terciária.

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

#define MAX_ENTRADAS 100
#define MAX_LEN 101

// Estrutura para agrupar as três strings de uma entrada
typedef struct {
    char string1[MAX_LEN];
    char string2[MAX_LEN];
    char string3[MAX_LEN];
} Entrada;
```

```
// Função de comparação para o qsort
int comparar_entradas(const void *a, const void *b) {
    const Entrada *entradaA = (const Entrada *)a;
    const Entrada *entradaB = (const Entrada *)b;

    // 1. Compara pela string1
    int cmp1 = strcmp(entradaA->string1, entradaB->string1);
    if (cmp1 != 0) {
        return cmp1; // Retorna se forem diferentes
    }

    // 2. Se string1 for igual, compara pela string2
    int cmp2 = strcmp(entradaA->string2, entradaB->string2);
    if (cmp2 != 0) {
        return cmp2; // Retorna se forem diferentes
    }

    // 3. Se string1 e string2 forem iguais, compara pela string3
    return strcmp(entradaA->string3, entradaB->string3);
}

// Função para remover a quebra de linha do fgets
void remover_newline(char *str) {
    str[strcspn(str, "\n")] = 0;
}

int main() {
    Entrada entradas[MAX_ENTRADAS];
    int n;

    printf("Digite a quantidade de entradas (n): ");
    scanf("%d", &n);
    getchar(); // Consome o \n

    printf("Digite as %d entradas (3 linhas por entrada):\n", n);
    for (int i = 0; i < n; i++) {
        // printf("Entrada %d, String 1: ", i + 1);
        fgets(entradas[i].string1, MAX_LEN, stdin);
        remover_newline(entradas[i].string1);

        // printf("Entrada %d, String 2: ", i + 1);
        fgets(entradas[i].string2, MAX_LEN, stdin);
        remover_newline(entradas[i].string2);

        // printf("Entrada %d, String 3: ", i + 1);
        fgets(entradas[i].string3, MAX_LEN, stdin);
        remover_newline(entradas[i].string3);
    }
```

```
    // Ordena o array de structs usando a função de comparação
    qsort(entradas, n, sizeof(Entrada), comparar_entradas);

    printf("\n--- Saida ordenada ---\n");
    for (int i = 0; i < n; i++) {
        // Imprime no formato "s1, s2, s3"
        printf("%s, %s, %s\n",
               entradas[i].string1,
               entradas[i].string2,
               entradas[i].string3);
    }

    return 0;
}
```