1. Um exemplo simples de CWEB. Este programa recebe um arquivo texto e ordena as linhas em ordem alfabética (para ser mais preciso, na ordem lexicográfica induzida pela tabela ASCII). Por simplicidade, *assumimos* que as linhas têm no máximo COMP_MAX caracteres e que o arquivo tem no máximo L_MAX linhas.

Observação. A versão do CWEB que estamos usando neste programa não mescla muito bem com o português: um projeto interessante seria encontrar uma solução simples para este problema.

```
#define COMP_MAX 256
#define L_MAX 8192
```

2. Usaremos a variável *status* para passar ao sistema operacional informação sobre o sucesso, ou não, da execução deste programa. A variável *nome_do_prog* é usada para guardar o nome do programa.

```
#define OK 0    /* valor de status quando tudo correu bem */
#define erro_na_alocacao_de_memoria 1
        /* valor de status para quando ocorre erro na alocação de memória */
```

⟨ Variáveis globais 2 ⟩ ≡

```
  char *nome_do_prog;
  int status = OK;
```

This code is used in section 3.

3. Aqui está a estrutura geral do nosso programa `ex12.w`.

```
  ⟨ Inclusão de arquivos sobre as bibliotecas 4 ⟩
  ⟨ Variáveis globais 2 ⟩
  ⟨ Função de ordenação 11 ⟩
  ⟨ O programa principal 5 ⟩
```

4. Precisamos incluir os arquivos `.h` para as funções de entrada e saída e para as funções de alocação de memória.

⟨ Inclusão de arquivos sobre as bibliotecas 4 ⟩ ≡

```
#include <stdio.h>    /* para as funções de entrada e saída */
#include <stdlib.h>    /* para calloc(), malloc(), e free() */
#include <string.h>    /* para as funções sobre cadeias de caracteres */
```

This code is used in section 3.

5. Vamos começar com a função *main*.

⟨ O programa principal 5 ⟩ ≡

```
  main(argc, argv)
      int argc;    /* o número argumentos na linha de comando */
      char **argv;    /* um vetor de strings, contendo os argumentos */
  { ⟨ Variáveis locais a main 7 ⟩
    nome_do_prog = argv[0];
    ⟨ Leia a entrada 6 ⟩;
    ⟨ Ordene 10 ⟩;
    ⟨ Imprima o resultado 9 ⟩;
    exit(status);
  }
```

This code is used in section 3.

6. Façamos agora a leitura. Assumimos que a entrada vem em *stdin*, por simplicidade. Usamos a função *gets*( ) de *stdio.h* para ler as linhas da entrada. Esta função lê toda a linha, substitui o símbolo de fim de linha `'\n'` por `'\0'`, que, na convenção de `C`, marca o fim de cadeias. As funções *strlen*( ) e *strcpy*( ) são de *string.h*.

```
⟨ Leia a entrada 6 ⟩ ≡
  ⟨ Inicialize o vetor texto 8 ⟩;
  i = 0;
  while (gets(linha)) {
    tmp = strlen(linha);
    s = (char *) malloc(tmp + 2);    /* +2 para o '\n' e '\0' a serem acrescentado depois */
    if (s ≡ Λ) {    /* estourou a memória... */
      status = erro_na_alocacao_de_memoria;
      fprintf(stderr, "%s:␣Faltou␣memoria!\n", nome_do_prog);
      exit(status);
    }
    strcpy(s, linha);
    s[tmp] = '\n';
      /* acrescentamos um '\n' explicitamente porque gets( ) substitui este caracter por '\0' */
    s[++tmp] = '\0';    /* marcação de fim de cadeia */
    texto[i++] = s;
  }
  n_linhas = i;
```

This code is used in section 5.

7. ⟨ Variáveis locais a *main* 7 ⟩ ≡

```
  char linha[COMP_MAX];    /* para a leitura de uma linha */
  char *s;    /* um ponteiro para uma cópia da linha lida */
  int i;    /* índice da linha sendo lida */
  int n_linhas;    /* número de linhas lidas */
  int tmp;
  char *texto[L_MAX];    /* vetor de apontadores para strings, cada string contendo uma linha do texto */
```

This code is used in section 5.

8. ⟨ Inicialize o vetor *texto* 8 ⟩ ≡

```
  for (i = 0; i < L_MAX; i++) texto[i] = Λ;
```

This code is used in section 6.

9. ⟨ Imprima o resultado 9 ⟩ ≡

```
  for (i = 0; texto[i]; i++) printf("%s", texto[i]);
```

This code is used in section 5.

10. A rotina de ordenacao. Usamos o algoritmo QuickSort, devido a Hoare [veja, por exemplo, o Volume 3 de Knuth, D. E., *Sorting and Searching*, Second Edition (Reading, Massachusetts: Addison-Wesley, 1998), *xiv* + 780pp. + foldout. ISBN 0-201-89685-0]. Esté é um algoritmo genérico de ordenação que é muito popular pela sua eficiência. Ele é bastante sutil, e no momento não vamos nos preocupar muito com o funcionamento dele.

```
⟨ Ordene 10 ⟩ ≡
  quicksort(texto, 0, n_linhas − 1);
```

This code is used in section 5.

11.
```
⟨ Função de ordenação 11 ⟩ ≡
  void quicksort(char **tab, int l, int r)
  { int j;
    if (l < r) {
      ⟨ Determine o índice j para a partição 12 ⟩;
      quicksort(tab, l, j);
      quicksort(tab, j + 1, r);
    }
  }
```

This code is used in section 3.

12.
```
⟨ Determine o índice j para a partição 12 ⟩ ≡
  { char *pivot;
    char *ptr_aux;
    int i;
    pivot = tab[l];
    i = l − 1;
    j = r + 1;
    while (1) {
      while (strcmp(tab[−−j], pivot) > 0) ;
      while (strcmp(tab[++i], pivot) < 0) ;
      if (i ≥ j) break;
      ptr_aux = tab[i], tab[i] = tab[j], tab[j] = ptr_aux;
    }
  }
```

This code is used in section 11.

13. Indice. Aqui está uma lista de identificadores que são usados, com indicação de onde eles aparecem. As entradas sublinhadas indicam onde eles aparecem.