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.

Faremos uma contagem do número de chamadas a *strcmp*( ) que fazemos no processo de ordenação ao executarmos este programa. Poderemos assim usar este programa para fazer alguns testes sobre o desempenho do "heapsort", que é o algoritmos que usamos para o processo de ordenação. Usaremos a variável global *n_comp* para esta contagem.

```
#define COMP_MAX 256
#define L_MAX 200000
#define o n_comp++
```

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;
unsigned long n_comp;
```

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 12 ⟩
⟨ 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 11 ⟩;
  ⟨ Imprima o resultado 9 ⟩;
  ⟨ Imprima o número de comparações feitas 10 ⟩;
  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. ⟨ Imprima o número de comparações feitas 10 ⟩ ≡

```
  printf("\nFiz %lu comparacoes!\n", n_comp);
```

This code is used in section 5.

11. A ordenação. Usaremos o "heapsort" neste programa. A chamada da função de ordenação é simples.

```
⟨ Ordene 11 ⟩ ≡
  heapsort(texto, n_linhas);
```

This code is used in section 5.

12. Implementamos nesta e na seção seguinte o algoritmo *heapsort* de ordenação. Vale a pena lembrarmos que *texto* começa indexado com 0. Desta forma, por exemplo, os filhos de um nó de índice k tem filhos de índices $2k+1$ e $2k+2$. A função que escrevemos nesta seção recebe um *heap* cuja raiz é $tab[k]$, exceto que pode acontecer da propriedade de *heap* ser violada entre $tab[k]$ e seus filhos. A propriedade de *heap* deve valer para todos os *outros* pares pai–filho. A função que escrevemos aqui também recebe como parâmetro um inteiro n que especifica o número de elementos do *heap* inteiro.

```
⟨ Função de ordenação 12 ⟩ ≡
  void transforme_em_heap(char **tab, int n, int k)
  { int j;
    char *aux;
    aux = tab[k];
    while (k < n/2) {
      j = (k ≪ 1) + 1;       /* índice do filho esquerdo */
      if ((j < n − 1) ∧ (o, strcmp(tab[j], tab[j + 1]) < 0)) j++;
          /* precisamos saber qual dos filhos é maior (se existe mais de um) */
      if (o, strcmp(aux, tab[j]) ≥ 0) break;
      tab[k] = tab[j];
      k = j;
    }
    tab[k] = aux;
  }
```

See also section 13.

This code is used in section 3.

13. Aqui temos a função de ordenação propriamente dita. Precisamos primeiro montar o *heap* e então ordenar os elementos "extraindo" os maiores elementos um por um (pondo-os no fim do vetor).

Note que, pelo fato de termos o nosso vetor começando com o índice 0, o elemento do *heap* de índice mais alto que tem um filho é aquele de índice $\lfloor n/2 \rfloor - 1$, onde n é o número de elementos no *heap*.

```
⟨ Função de ordenação 12 ⟩ +≡
  void heapsort(char **tab, int n)
  {
    int k;
    char *tmp;
    for (k = n/2 − 1; k ≥ 0; k−−) transforme_em_heap(tab, n, k);
    while (n > 1) {
      tmp = tab[0];
      tab[0] = tab[n − 1];
      tab[n − 1] = tmp;
      transforme_em_heap(tab, −−n, 0);
    }
  }
```

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

argc: 5.
argv: 5.
aux: 12.
calloc: 4.
COMP_MAX: 1, 7.
erro_na_alocacao_de_memoria: 2, 6.
exit: 5, 6.
fprintf: 6.
free: 4.
gets: 6.
heapsort: 11, 13.
i: 7.
j: 12.
k: 12, 13.
L_MAX: 1, 7, 8.
linha: 6, 7.
main: 5.
malloc: 4, 6.
n: 12, 13.
n_comp: 1, 2, 10.
n_linhas: 6, 7, 11.
nome_do_prog: 2, 5, 6.
o: 1.
OK: 2.
printf: 9, 10.
s: 7.
status: 2, 5, 6.
stderr: 6.
stdin: 6.
stdio: 6.
strcmp: 1, 12.
strcpy: 6.
string: 6.
strlen: 6.
tab: 12, 13.
texto: 6, 7, 8, 9, 11, 12.
tmp: 6, 7, 13.
transforme_em_heap: 12, 13.

⟨ Função de ordenação 12, 13 ⟩ Used in section 3.
⟨ Imprima o número de comparações feitas 10 ⟩ Used in section 5.
⟨ Imprima o resultado 9 ⟩ Used in section 5.
⟨ Inclusão de arquivos sobre as bibliotecas 4 ⟩ Used in section 3.
⟨ Inicialize o vetor *texto* 8 ⟩ Used in section 6.
⟨ Leia a entrada 6 ⟩ Used in section 5.
⟨ O programa principal 5 ⟩ Used in section 3.
⟨ Ordene 11 ⟩ Used in section 5.
⟨ Variáveis globais 2 ⟩ Used in section 3.
⟨ Variáveis locais a *main* 7 ⟩ Used in section 5.