```
ctype.txt
========
int isalnum(int);
int isalpha(int);
int iscntrl(int);
int isdigit(int);
int isgraph(int);
int islower(int);
int isprint(int);
int ispunct(int);
int isspace(int);
int isupper(int);
int isxdigit(int);
int tolower(int);
int toupper(int);

vogais.c
========
/*
 * arquivo: vogais.c
 * -----------------
 * Este programa le uma sequencia de caracteres e determina
 * o numero de vogais, consoantes, caracteres nao alfanumericos,
 * e o total deles.
 */

#include <stdio.h>
#include <ctype.h>

int main()
{
  int ch;
  int v=0, conso=0, outros=0;


  while ((ch=getchar())!=EOF) {

    switch (tolower(ch)) {
    case 'a': case 'e': case 'i': case 'o': case 'u':
      v++;
      break;

    default:
      if (isalpha(ch))
        conso++;
      else
        outros++;
    }
  }

  printf("No. vogais: %d\nNo. consoantes: %d\n", v, conso);
  printf("No. outros: %d\nNo. caracteres total: %d\n", outros,
  v+conso+outros);
  return 0;
}

mais_longa.c
========
/*
 *  Este programa recebe como entrada uma cadeia de caracteres terminada
 *  por um '.' e determina o comprimento maximo de uma palavra nesta entrada.
```

```
 *  Entendemos como uma palavra uma cadeia contigua de letras (caracteres de
 *  'a'..'z' e 'A'..'Z').
 *
 *  Envie notificacao de erros para Y. Kohayakawa <yoshi@ime.usp.br>
 *  (responsavel pela Turma 10 de mac2166 - Poli, 1o. semestre de 2000)
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define TRUE  1
#define FALSE 0

int main()
{
  int ch;
  int max;
  int comp_atual;
  int dentro;     /* indicador de estado: 0 se fora de uma palavra e 1 se
                     dentro */
  max=0;
  dentro=0;

  while (TRUE) {
    ch=getchar();

    switch (dentro) {
    case 1:
      if (isalpha(ch)) {
        comp_atual++;
      } else {
        if (comp_atual>max) {
          max=comp_atual;
        }
        comp_atual=0;
        dentro=0;
      }
      break;

    case 0:
      if (isalpha(ch)) {
        comp_atual=1;
        dentro=1;
      }
      break;

    default: printf("Nao pode acontecer\n");
      exit(1);
    }

    if (ch=='.') break;

  }

  printf("Maior comprimento: %d\n", max);
  return 0;
}

string.txt
========
```

2.14 string.h

The string header provides many functions useful **for** manipulating strings (character arrays).

Macros:

    NULL

Variables:

    typedef size_t

Functions:

```
    memchr();
    memcmp();
    memcpy();
    memmove();
    memset();
    strcat();
    strncat();
    strchr();
    strcmp();
    strncmp();
    strcoll();
    strcpy();
    strncpy();
    strcspn();
    strerror();
    strlen();
    strpbrk();
    strrchr();
    strspn();
    strstr();
    strtok();
    strxfrm();
```

2.14.1 Variables and Definitions

size_t is the **unsigned** integer result of the **sizeof** keyword.
NULL is the value of a null pointer constant.

str.c
========

```c
/*-*/
/********************************************************
 * Name:                                                *
 *      string                                          *
 *                                                      *
 * Purpose:                                             *
 *      Demonstrate how to use strcpy (string copy)     *
 *                                                      *
 * Usage:                                               *
 *      Run it and find out the name of "Sam"           *
 ********************************************************/
/*+*/
#include <string.h>
#include <stdio.h>
int main()
{
```

```
    char name[30];    /* First name of someone */
    strcpy(name, "Sam");    /* Initialize the name */
    printf("The name is %s\n", name);
    return 0;
}


full.c
========
/*-*/
/********************************************************
 * Name:                                                *
 *      concat                                          *
 *                                                      *
 * Purpose:                                             *
 *      Demonstrates how to put strings together using  *
 *      strcat.                                         *
 *                                                      *
 * Usage:                                               *
 *      Run it and see my full name.                    *
 *                                                      *
 * Notes:                                               *
 *      Don't try to pronounce the last name.           *
 ********************************************************/
/*+*/
#include <string.h>
#include <stdio.h>

int main()
{
    char first[100];            /* first name */
    char last[100];             /* last name */
    char full_name[200];        /* full version of first and last name */

    strcpy(first, "Steve");     /* Initialize first name */
    strcpy(last, "Oualline");   /* Initialize last name */

    strcpy(full_name, first);   /* full = "Steve" */
    /* Note: strcat not strcpy */
    strcat(full_name, " ");     /* full = "Steve " */
    strcat(full_name, last);    /* full = "Steve Oualline" */

    printf("The full name is %s\n", full_name);
    return 0;
}

full1.c
========
/*-*/
/********************************************************
 * Name:                                                *
 *      full name (V1)                                  *
 *                                                      *
 * Purpose:                                             *
 *      Get first and last name and print them          *
 *      together.                                       *
 *                                                      *
 * Usage:                                               *
 *      Run the program.                                *
 *      Type in the first name.                         *
 *      Type in the second name.                        *
```

```
 *      Get the result.                               *
 *                                                    *
 * BUG: This program contains a bug that causes       *
 *      the names to appear on separate lines.        *
 ******************************************************/
/*+*/
#include <stdio.h>
#include <string.h>

/* First and last name of the person (computed) */
char full[200];

int main()
{
    char first[100];            /* first name of person we are working with */
    char last[100];             /* His last name */

    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);

    strcpy(full, first);
    strcat(full, " ");
    strcat(full, last);

    printf("The name is %s\n", full);
    return 0;
}

full2.c
========
/*-*/
/******************************************************
 * Name:                                              *
 *      full name (V2)                                *
 *                                                    *
 * Purpose:                                           *
 *      Get first and last name and print them        *
 *      together.                                     *
 *                                                    *
 * Usage:                                             *
 *      Run the program.                              *
 *      Type in the first name.                       *
 *      Type in the second name.                      *
 *      Get the result.                               *
 *                                                    *
 * Note: This version does not contain the bug found  *
 *      in program 4_6.c
 ******************************************************/
/*+*/
#include <stdio.h>
#include <string.h>

int main()
{
    char first[100];            /* first name of person we are working with */
    char last[100];             /* His last name */

/* First and last name of the person (computed) */
```

```
    char full[100];

    printf("Enter first name: ");
    fgets(first, sizeof(first), stdin);
    /* trim off last character */
    first[strlen(first) - 1] = '\0';

    printf("Enter last name: ");
    fgets(last, sizeof(last), stdin);
    /* trim off last character */
    last[strlen(last) - 1] = '\0';

    strcpy(full, first);
    strcat(full, " ");
    strcat(full, last);

    printf("The name is %s\n", full);
    return 0;
}

my_sort.c
========
/*
 * Adaptado de K&R
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXLINES 5000           /* Numero maximo de linhas que podemos manipular */
#define MAXLEN 1000

/* readlines(): le as linhas da entrada em lineptr */
int readlines(char *lineptr[], int nlines);

/* writelines(): imprime as linhas em lineptr */
void writelines(char *lineptr[], int nlines);

/* my_qsort(): funcao de ordenacao */
void my_qsort(char *lineptr[], int left, int right);

/************************************************************/
int main()
{
  char *lineptr[MAXLINES];
  int nlines;                   /* Numero de linhas na entrada */

  nlines = readlines(lineptr, MAXLINES);
  if (nlines > 0) {
    my_qsort(lineptr, 0, nlines - 1);
    writelines(lineptr, nlines);
    return 0;
  } else {
    printf("Error: entrada muito grande.\n");
    return 1;
  }
}

/************************************************************/

/* readlines(): le as linhas da entrada em lineptr */
```

```c
int readlines(char *lineptr[], int maxlines)
{
  int len, nlines;
  char *p, line[MAXLEN];
  nlines = 0;
  for (;;) {
    p = fgets(line, sizeof(line), stdin);
    if (p == NULL)
      break;
    len = strlen(line) + 1;
    p = (char *) malloc(len);
    if ((nlines >= maxlines) || (p == NULL))
      return -1;
    else {
      strcpy(p, line);
      lineptr[nlines++] = p;
    }
  }
  return nlines;
}

/* writelines(): imprime as linhas em lineptr */
void writelines(char *lineptr[], int nlines)
{
  int i;
  for (i = 0; i < nlines; i++)
    printf("%s", lineptr[i]);
}

/* Rotinas para ordenacao */
void swap(char *v[], int i, int j)
{
  char *temp;
  temp = v[i];
  v[i] = v[j];
  v[j] = temp;
}

/*
 * Esta implementacao usa uma versao do quicksort
 * para ordenar v[].
 */
void my_qsort(char *v[], int left, int right)
{
  int i, last;
  if (left >= right)              /* Do nothing if array has fewer than */
    return;                       /* two elements                       */
  swap(v, left, (left + right) / 2);    /* save the middle(pivot) position */
  last = left;
  for (i = left + 1; i <= right; i++) {
    if (strcmp(v[i], v[left]) < 0)      /* if the element is less than the pivot */
      swap(v, ++last, i);         /* move it to the left side               */
  }
  swap(v, left, last);            /* restore the pivot position */

  my_qsort(v, left, last - 1);
  my_qsort(v, last + 1, right);
}
```