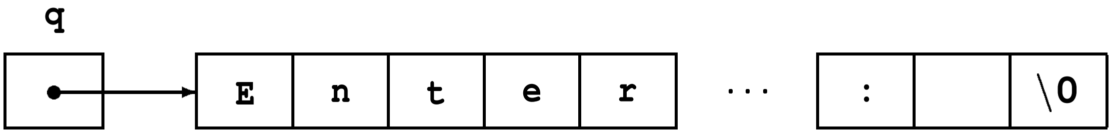
CHAPTER 10

STRINGS

WHAT A STRING LOOKS LIKE



yesorno.c

(Figure 10.1 – Page 233)

```
/*
 * Obtain a yes or no answer from the user, prompting for response.
 */
#include <stdio.h>
#include <ctype.h>

static const int YES = 1;        /* can be any nonzero value */
static const int NO = 0;         /* shouldn't be changed from zero */

int yesorno(const char *prompt)
{
  int answer, junk;

  while (printf(prompt), (answer = tolower(getchar())) != EOF)
  {
    for (junk = answer; junk != '\n' && junk != EOF; junk = getchar())
      ;                          /* skip extra characters on line */
    if (answer == 'y')
      return YES;
    if (answer == 'n')
      return NO;
    printf("That's not a reasonable answer.\n");
  }
  return NO;
}
```

ynmain.c

(Figure 10.2 – Page 233)

```
/*
 * Main program using our new "yesorno" function.
 */
#include <stdio.h>

int main()
{
  int   yesorno(const char *prompt);
  char *q;                                    /* question for user */

  q = "Enter YES or NO: ";
  printf("%s!\n", yesorno(q) ? "YES" : "NO");
  return 0;
}
```

lineno.c

(Figure 10.3 – Page 235)

```
/*
 * Line number its input, one line at a time.
 */
#include <stdio.h>

#define MAXLEN  80                      /* longest line */

int main()
{
  int           getline(char line[], int max);
  char          line[MAXLEN + 1];       /* input line (plus \0) */
  unsigned long lines = 0L;             /* line count */

  while (getline(line, MAXLEN) != -1)
    printf("%6lu %s\n", ++lines, line);
  return 0;
}
```

getline.c

(Figure 10.4 – Page 235)

```
/*
 * Read a single input line into an array.
 */
#include <stdio.h>

int getline(char line[], int max)
{
  int c;                                    /* current character */
  int i = 0;                                /* character count */

  while ((c = getchar()) != '\n' && c != EOF)
    if (i < max)
      line[i++] = c;
  line[i] = '\0';                           /* terminate with null */
  return (c == EOF) ? -1 : i;
}
```

STRING LIBRARY FUNCTIONS

NAME	WHAT IT DOES
`char *strcat(s1,s2)`	Concatenates **`s2`** to the end of **`s1`**, returning **`s1`**.
`char *strncat(s1,s2,n)`	Concatenates at most **`n`** characters from **`s2`** to the end of **`s1`**, returning **`s1`**.
`char *strcpy(s1,s2)`	Copies **`s2`** to **`s1`**, including the null character, returning **`s1`**.
`char *strncpy(s1,s2,n)`	Copies at most **`n`** characters from **`s2`** to **`s1`**, returning **`s1`**. If it stops copying before the null, the null isn't added to **`s2`**.
`int strcmp(s1,s2)`	Compares **`s1`** and **`s2`**, returning less than 0, 0, or greater than 0, depending on whether **`s1`** is less than, equal to, or greater than **`s2`**, respectively.
`int strncmp(s1,s2,n)`	Compares at most n characters; same return as **`strcmp`**.
`size_t strlen(s)`	Returns number of characters in **`s`**, *not* counting trailing null.
`char *strchr(s,c)`	Returns a pointer to the first occurrence of **`c`** in **`s`**, or **`NULL`**.
`char *strrchr(s,c)`	Returns a pointer to the last occurrence of **`c`** in **`s`**, or **`NULL`**.
`char *strstr(s1, s2)`	Returns a pointer to the first occurrence of the string **`s2`** in **`s1`**.
`size_t strspn(s1,s2)`	Returns length of prefix of **`s1`** consisting of characters in **`s2`**.
`size_t strcspn(s1,s2)`	Returns length of prefix of **`s1`** consisting characters not in **`s2`**.
`char *strpbrk(s1,s2)`	Returns pointer to first occurrence in **`s1`** of any character in **`s2`**, or **`NULL`** if it's in the string.

Sample I/O – convnam

```
Alex Quilici
Larry Miller
_ _ _ _ _
```

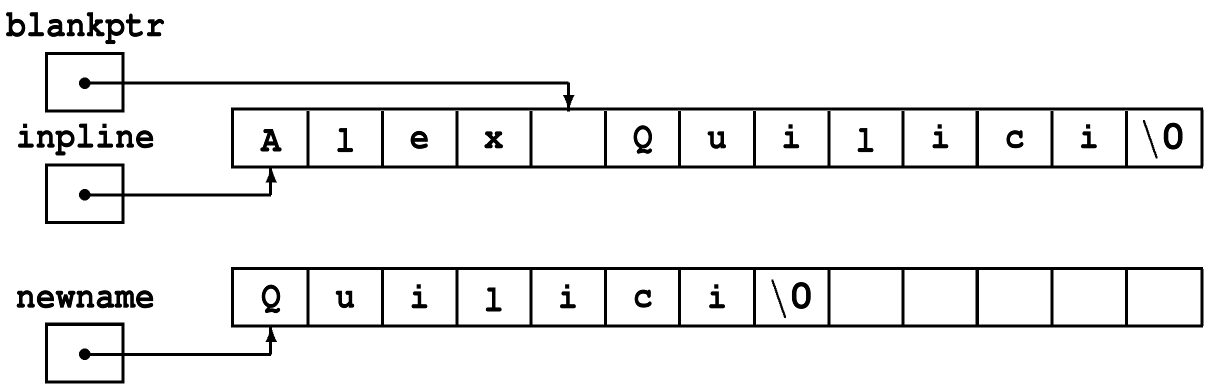
```
Quilici, Alex
Miller, Larry
```

convnam.c

(Figure 10.5 – Page 238)

```
/*
 * Name conversion program
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>

#define MAXLEN    80

int main()
{
  int   getline(char line[], int maxlen);
  char  inpline[MAXLEN + 1];           /* input line */
  int   len;                           /* length of input line */
  char  newname[MAXLEN + 1];           /* converted input name */
  char *blankptr;                      /* pointer to blank separator */

  while ((len = getline(inpline, MAXLEN)) != -1)
    if (len == 0 || (blankptr = strchr(inpline, ' ')) == NULL ||
        strlen(blankptr + 1) == 0)
      printf("ERROR: bad format input line\n");
    else
    {                                  /* split into two strings */
      strcpy(newname, blankptr + 1);   /* make last name first */
      strcat(newname, ", ");           /* append comma */
      *blankptr = '\0';
      strcat(newname, inpline);        /* put first name last */
      printf("%s\n", newname);         /* write new name */
    }
  return 0;
}
```

POINTERS USED TO CONVERT NAMES



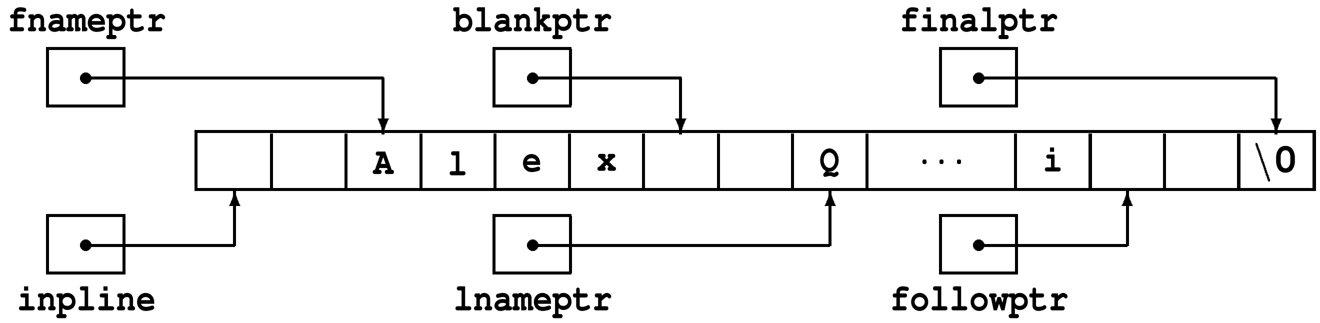
convnam2.c

(Figure 10.7 – Page 240)

```
/*
 * Name conversion program, more free-format version.
 */
#include <stdio.h>
#include <string.h>
#include <stddef.h>

#define MAXLEN  80                      /* longest input line */

const char *const Spaces = " \t";

int main()
{
  int   getline(char line[], int maxline);
  char  inpline[MAXLEN + 1];            /* input line */
  int   len;                            /* input line length */
  char  newname[MAXLEN + 1];            /* final converted name */
  char *fnameptr;                       /* ptr to first name */
  char *lnameptr;                       /* ptr to last name */
  char *blankptr;                       /* ptr to separating blank */
  char *followptr;                      /* ptr to char after last name */
  char *finalptr;                       /* ptr to end of trailing blanks */
```

```
  while ((len = getline(inpline, MAXLEN)) != -1)
    if (len == 0)
      printf("ERROR: empty input line\n");
    else
    {                               /* compute needed pointers */
      fnameptr = inpline + strspn(inpline, Spaces);
      blankptr = fnameptr + strcspn(fnameptr, Spaces);
      lnameptr = blankptr + strspn(blankptr, Spaces);
      followptr = lnameptr + strcspn(lnameptr, Spaces);
      finalptr = followptr + strspn(followptr, Spaces);

      if (fnameptr == blankptr || blankptr == lnameptr ||
        lnameptr == followptr || finalptr != inpline + len)
        printf("ERROR: not exactly two names on input line\n");
      else
      {                             /* do the actual conversion */
        *followptr = '\0';          /* delete trailing blanks */
        *blankptr = '\0';           /* separate into two strings */
        strcpy(newname, lnameptr);
        strcat(newname, ", ");
        strcat(newname, fnameptr);
        printf("%s\n", newname);    /* display new name */
      }
    }
  return 0;
}
```

POINTERS USED TO CONVERT NAMES



yesorno2.c

(Figure 10.9 – Page 241)

```
/*
 * Obtain a yesorno answer, allowing multicharacter responses.
 */
#include <stdio.h>
#include <string.h>

#define MAXLEN  80

static const int YES = 1;
static const int NO = 0;

int yesorno(const char *prompt)
{
  int  getline(char line[], int max);
  char line[MAXLEN + 1];                    /* input line */

  while (printf(prompt), getline(line, MAXLEN) != -1)
  {
    if (strcmp(line, "yes") == 0 || strcmp(line, "YES") == 0)
      return YES;
    if (strcmp(line, "no") == 0 || strcmp(line, "NO") == 0)
      return NO;
    printf("That's not a reasonable answer.\n");
  }
  return NO;
}
```

string.c

(Figure 10.10 – Page 243)

```
/*
 * Implementations of several standard string functions
 */
#include <stddef.h>

char *strcpy(char d[], const char s[])          /* copy "s" -> "d" */
{
  int i;

  for (i = 0; (d[i] = s[i]) != '\0'; i++)
    ;
  return d;
}

int strcmp(const char s[], const char t[])      /* compare "s" and "t" */
{
  int i;

  for (i = 0; s[i] == t[i] && s[i] != '\0'; i++)
    ;
  return s[i] - t[i];
}

size_t strlen(const char s[])                   /* length of "s"? */
{
  size_t i;

  for (i = 0; s[i] != '\0'; i++)
    ;
  return i;
}
```

calc.c

(Figure 10.11 – Page 244)

```
/*
 * Simple calculator program (uses getline) to read its input.
 */
#include <stdio.h>

#define MAXLEN 80

int main()
{
  int    getline(char line[], int max);
  double op1, op2;                      /* operands */
  char   operator;                      /* operator */
  double result;                        /* result */
  char   inpline[MAXLEN + 1];           /* to hold input line */
```

calc.c

(Figure 10.11 – Page 244)

```
  while (getline(inpline, MAXLEN) != -1)
  {
    if (sscanf(inpline, "%lf%c%lf", &op1, &operator, &op2) != 3)
    {
      printf("Input line in error.\n");
      continue;
    }
    switch(operator)
    {
      case '+':
        result = op1 + op2;
        break;
      case '-':
        result = op1 - op2;
        break;
      case '*':
        result = op1 * op2;
        break;
      case '/':
        if (op2 != 0)
          result = op1 / op2;
        else
        {
          result = 0.0;
          printf("Warning: division by zero.\n");
        }
        break;
      default:
        printf("Unknown operator: %c\n", operator);
        continue;
    }
    printf("%f\n", result);
  }
  return 0;
}
```

countpct.c

(Figure 10.12 – Page 245)

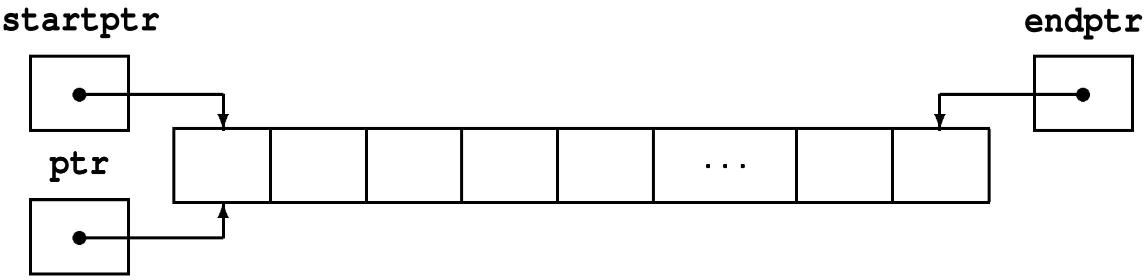
```
/*
 * Counts various types of punctuation characters.
 */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stddef.h>

const char *const Puncts     = ",.;:!?";
const char *const Brackets   = "(){}[]";
const char *const Quotes     = "'`\"";
const int         PrintWidth = 11;         /* width of counter */

int main()
{
  int           c;
  unsigned long quotes = 0,
                brackets = 0,
                delims = 0,
                others = 0;

  while ((c = getchar()) != EOF)
    if (strchr(Quotes, c) != NULL)
      quotes++;                            /* single & double quotes */
    else if (strchr(Brackets, c) != NULL)
      brackets++;                          /* punctuation brackets */
    else if (strchr(Puncts, c) != NULL)
      delims++;                            /* usual punctuation */
    else
      others++;                            /* other characters */
  printf("Quotes     %*lu\n", PrintWidth, quotes);
  printf("Brackets   %*lu\n", PrintWidth, brackets);
  printf("Delimiters %*lu\n", PrintWidth, delims);
  printf("Other      %*lu\n", PrintWidth, others);
  return 0;
}
```

getline2.c

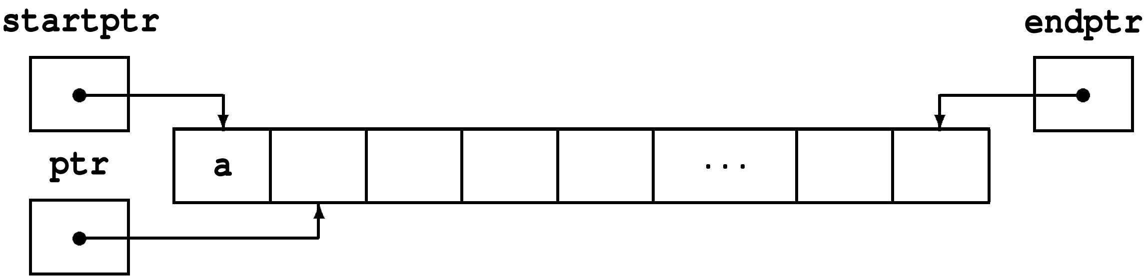
(Figure 10.13 – Page 246)

```
/*
 * Read input line into an array, pointer version.
 */
#include <stdio.h>

int getline(char *ptr, int max)
{
  int         c;                         /* current character */
  char *const startptr = ptr;            /* pointer to line start */
  char *const endptr   = ptr + max;      /* pointer to line end */

  while ((c = getchar ()) != '\n' && c != EOF)
    if (ptr < endptr)
      *ptr++ = c;
  *ptr = '\0';                           /* terminate with null */
  return (c == EOF) ? -1 : ptr - startptr;
}
```

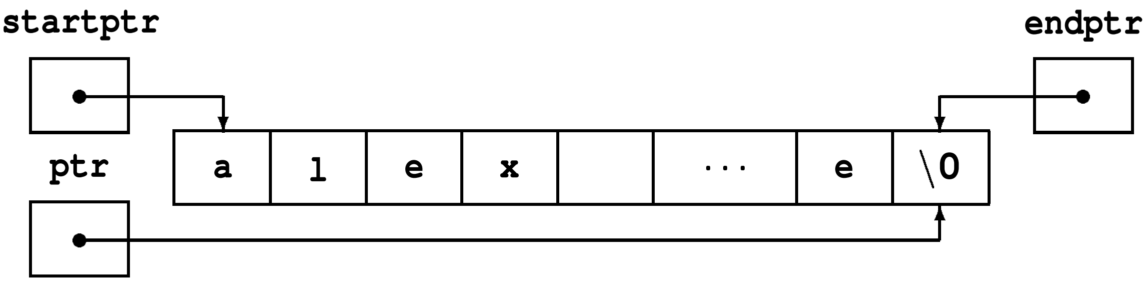
POINTERS USED TO READ LINE



(a) The pointers when **getline** starts up.



(b) The pointers after reading one character.



(c) The pointers after reading the input line “alex was here”.

string2.c

(Figure 10.15 – Page 248)

```
/*
 * Pointer versions of our string functions
 */
#include <stddef.h>

char *strcpy(char *dptr, const char *sptr)
{
  char *const startptr = dptr;      /* save starting pointer */

  for (; (*dptr = *sptr) != '\0'; dptr++, sptr++)
    ;
  return startptr;
}

int strcmp(const char *sptr, const char *tptr)
{
  for (; *sptr == *tptr && *sptr != '\0'; sptr++, tptr++)
    ;
  return *sptr - *tptr;             /* difference of two chars */
}

size_t strlen(const char *ptr)
{
  const char *const startptr = ptr;

  for (; *ptr != '\0'; ptr++)
    ;
  return ptr - startptr;
}
```

Sample I/O – uniq

```
C
C
is
a
wonderful
wonderful
wonderful
wonderful
language
language
```

_ _ _ _ _

```
C
is
a
wonderful
language
```

uniq.c

(Figure 10.16 – Page 249)

```
/*
 * Strip consecutive duplicate lines from the input.
 */
#include <stdio.h>
#include <string.h>

#define MAXLEN    80                       /* longest input line */
#define FALSE     0
#define TRUE      1

int main()
{
  int  getline(char *buf, int bufsize);
  char curr[MAXLEN + 1],                   /* current line */
       prev[MAXLEN + 1];                   /* previous line */
  int  first = TRUE;

  for (; getline(curr, MAXLEN) != -1; strcpy(prev, curr))
    if (first || strcmp(prev, curr) != 0)
    {
      printf("%s\n", curr);
      first = FALSE;
    }
  return 0;
}
```

REMINDERS

- Don't forget to supply the trailing null character when creating a string.

- We can't simply assign one character string to another, just as we couldn't assign arrays.

- Make sure to null-terminate any character arrays you pass the standard string functions.

CHAPTER SUMMARY

- A string is an array of characters, terminated with the null character, '`\0`'. When we create a string, we must provide the terminating null ourselves.

- We initialize strings by declaring an array and providing a list of characters, or by declaring a pointer and assigning it to a string constant.

- We can easily write a function to read an input line into a string. This allows us to easily recover from input errors.

- Since strings are arrays, we can run through them using either array subscripting or pointer indexing.

- C provides a large library of functions for manipulating strings. To use them, we need to include string.h. They all expect the string to be null-terminated.