Chapter 8 - Characters and Strings

Outline
8.1 Introduction
8.2 Fundamentals of Strings and Characters
8.3 Character Handling Library
8.4 String Conversion Functions
8.5 Standard Input/Output Library Functions
8.6 String Manipulation Functions of the String Handling Library
8.7 Comparison Functions of the String Handling Library
8.8 Search Functions of the String Handling Library
8.9 Memory Functions of the String Handling Library
8.10 Other Functions of the String Handling Library
8.1 Introduction

• Introduce some standard library functions
  – Easy string and character processing
  – Programs can process characters, strings, lines of text, and blocks of memory

• These techniques used to make
  – Word processors
  – Page layout software
  – Typesetting programs
8.2 Fundamentals of Strings and Characters

- **Characters**
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An `int` value represented as a character in single quotes
    - `'z'` represents the integer value of `z`
8.2 Fundamentals of Strings and Characters

- The following example illustrates the two methods for variable initialization:

```c
#include <stdio.h>
main () {
    int sum=33;
    float money=44.12;
    char letter;
    double pressure;
    letter='E'; /* assign character value */
    pressure=2.01e-10; /*assign double value */
    printf("value of sum is %d\n",sum);
    printf("value of money is %f\n",money);
    printf("value of letter is %c\n",letter);
    printf("value of pressure is %e\n",pressure);
}
```

- which produces the following output:

```
value of sum is 33
value of money is 44.119999
value of letter is E
value of pressure is 2.010000e-10
```
8.2 Fundamentals of Strings and Characters

- **Strings**
  - Strings are **1D arrays of characters**.
    - Can include letters, digits and special characters (*, /, $)
  - String literal (string constant) - written in double quotes
    - "Hello"
  - Unlike other 1D arrays the **number of elements set** for a string is only an **upper limit**. The actual strings used in the program can have fewer elements. For example:
    ```
    static char name[18] = "Ivanova";
    ```
    - The string called `name` actually has only **8** elements.
      'I' 'v' 'a' 'n' 'o' 'v' 'a' '\0'
  - Notice that string constants **marked with double quotes** automatically include the end-of-string character. The **curly braces are not required** for string initialization at declaration, but can be used if desired (but don’t forget the end-of-string character).
8.2 Fundamentals of Strings and Characters

• Strings
  – String a pointer to first character
    • Value of string is the address of first character
  – Strings must be terminated by the null character '\0' which is (naturally) called the end-of-string character.
  – Don’t forget to count the end-of-string character when you calculate the size of a string.
  – Initializing a string can be done in three ways:
    • at declaration,
    • by reading in a value for the string, and
    • by using the strcpy function.
8.2 Fundamentals of Strings and Characters

- Direct initialization using the = operator is invalid. The following code would produce an error:
  - char name[34];
  - name = "Erickson"; /* ILLEGAL */

- To read in a value for a string use the %s format identifier:
  - scanf("%s", name);

- Note that the address operator & is not needed for inputting a string variable. The end-of-string character will automatically be appended during the input process.
8.2 Fundamentals of Strings and Characters

- **String definitions**
  - Define as a character array or a variable of type `char *`
    ```
    char color[] = "blue";
    char *colorPtr = "blue";
    ```
  - Remember that strings represented as character arrays end with '\0'
    - `color` has 5 elements

- **Inputting strings**
  - Use `scanf`
    ```
    scanf("%s", word);
    ```
    - Copies input into `word[]`
    - Do not need & (because a string is a pointer)
  - Remember to leave room in the array for '\0'
8.5 Standard Input/Output Library Functions

- Functions in `<stdio.h>`
- Used to manipulate character and string data

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int getchar( void );</code></td>
<td>Inputs the next character from the standard input and returns it as an integer.</td>
</tr>
<tr>
<td><code>char *gets( char *s );</code></td>
<td>Inputs characters from the standard input into the array <code>s</code> until a newline or end-of-file character is encountered. A terminating null character is appended to the array.</td>
</tr>
<tr>
<td><code>int putchar( int c );</code></td>
<td>Prints the character stored in <code>c</code>.</td>
</tr>
<tr>
<td><code>int puts( const char *s );</code></td>
<td>Prints the string <code>s</code> followed by a newline character.</td>
</tr>
<tr>
<td><code>int sprintf( char *s, const char *format, ... );</code></td>
<td>Equivalent to <code>printf</code>, except the output is stored in the array <code>s</code> instead of printing it on the screen.</td>
</tr>
<tr>
<td><code>int sscanf( char *s, const char *format, ... );</code></td>
<td>Equivalent to <code>scanf</code>, except the input is read from the array <code>s</code> instead of reading it from the keyboard.</td>
</tr>
</tbody>
</table>
8.5 Standard Input/Output Library Functions

- There are special functions designed specifically for string I/O. They are
  
  \[ \text{gets(string\_name)}; \]
  \[ \text{puts(string\_name)}; \]

- The **gets function** reads in a string from the keyboard. When the user hits return the string is inputted. The return is not part of the string and the end-of-string character is automatically appended.

- The **function puts** displays a string on the monitor. It does not print the end-of-string character, but does output a return at the end of the string.
8.5 Standard Input/Output Library Functions

- Here is a sample program demonstrating the use of these functions:

```c
char phrase[100];
printf("Please enter a sentence\n");
gets(phrase);
puts(phrase);
```

- A sample session would look like this

```
Please enter a sentence
The best lack all conviction, while the worst are passionate.
The best lack all conviction, while the worst are passionate.
```
/ Fig. 8.14: fig08_14.c

Using getchar and puts *

#include <stdio.h>

int main()
{
    char c;       /* variable to hold character input by user */
    char sentence[ 80 ]; /* create char array */
    int i = 0;     /* initialize counter i */

    /* prompt user to enter line of text */
    puts( "Enter a line of text:" );

    /* use getchar to read each character */
    while ( ( c = getchar() ) != 'n' ) {
        sentence[i] = c;
        i++;
    } /* end while */

    sentence[ i ] = '0';

    /* use puts to display sentence */
    puts( "\nThe line entered was:" );
    puts( sentence );

    return 0; /* indicates successful termination */
} /* end main */
Enter a line of text:
This is a test.

The line entered was:
This is a test.
Enter an integer and a double:
298 87.375

The formatted output stored in array s is:
integer: 298
double: 87.38
/* Fig. 8.16: fig08_16.c 
   Using sscanf */

#include <stdio.h>

int main()
{
    char s[] = "31298 87.375"; /* initialize array s */
    int x; /* define x */
    double y; /* define y */

    sscanf( s, "%d%lf", &x, &y );

    printf("The values stored in character array s are:",
           "integer:" , x, "double:" , y );

    return 0; /* indicates successful termination */
}

/* end main */

The values stored in character array s are:
integer: 31298
double:  87.375
More String Functions

Included in the `string.h` are several more string-related functions that are free for you to use. Here is a brief table of some of the more popular ones:

<table>
<thead>
<tr>
<th>Function</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>strcat</code></td>
<td>Appends to a string</td>
</tr>
<tr>
<td><code>strchr</code></td>
<td>Finds first occurrence of a given character</td>
</tr>
<tr>
<td><code>strcmp</code></td>
<td>Compares two strings</td>
</tr>
<tr>
<td><code>strcmpi</code></td>
<td>Compares two, strings, non-case sensitive</td>
</tr>
<tr>
<td><code>strcpy</code></td>
<td>Copies one string to another</td>
</tr>
<tr>
<td><code>strlen</code></td>
<td>Finds length of a string</td>
</tr>
<tr>
<td><code>strncat</code></td>
<td>Appends ( n ) characters of string</td>
</tr>
<tr>
<td><code>strncmp</code></td>
<td>Compares ( n ) characters of two strings</td>
</tr>
<tr>
<td><code>strncpy</code></td>
<td>Copies ( n ) characters of one string to another</td>
</tr>
<tr>
<td><code>strset</code></td>
<td>Sets ( n ) characters of string to a given character</td>
</tr>
<tr>
<td><code>strrchr</code></td>
<td>Finds last occurrence of given character in string</td>
</tr>
<tr>
<td><code>strspn</code></td>
<td>Finds first substring from given character set in string</td>
</tr>
</tbody>
</table>
8.6 String Manipulation Functions of the String Handling Library

- String handling library has functions to
  - Manipulate string data
  - Search strings
  - Tokenize strings
  - Determine string length

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char *strcpy( char *s1, const char *s2 )</code></td>
<td>Copies string s2 into array s1. The value of s1 is returned.</td>
</tr>
<tr>
<td><code>char *strncpy( char *s1, const char *s2, size_t n )</code></td>
<td>Copies at most n characters of string s2 into array s1. The value of s1 is returned.</td>
</tr>
<tr>
<td><code>char *strcat( char *s1, const char *s2 )</code></td>
<td>Appends string s2 to array s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.</td>
</tr>
<tr>
<td><code>char *strncat( char *s1, const char *s2, size_t n )</code></td>
<td>Appends at most n characters of string s2 to array s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.</td>
</tr>
</tbody>
</table>
The *strcpy* function:

- To use these functions be sure to include the `string.h` header file at the beginning of your program.

- The syntax of *strcpy* is

  \[
  \text{strcpy}(\text{string1}, \text{string2});
  \]

- `string2` is copied into `string1` at the beginning of `string1`. The previous contents of `string1` are overwritten.

```c
#include <string.h>
main () {
    char job[50];
    strcpy(job,"Professor");
    printf("You are a \%s \n",job);
}
```

You are a Professor
Using strcpy and strncpy */

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

int main()
{
    char x[] = "Happy Birthday to You"; /* initialize char array x */
    char y[25]; /* create char array y */
    char z[15]; /* create char array z */

    /* copy contents of x into y */
    printf("The string in array x is: ", x,
           "The string in array y is: ", strcpy( y, x ) );

    /* copy first 14 characters of x into z. Does not copy null character */
    strncpy( z, x, 14 );

    z[14] = '\0'; /* append '\0' to z's contents */
    printf("The string in array z is: %s\n" , z);
    return 0; /* indicates successful termination */
} /* end main */
/* Fig. 8.19: fig08_19.c */
#include <stdio.h>
#include <string.h>

int main()
{
    char s1[20] = "Happy ";  /* initialize char array s1 */
    char s2[] = "New Year ";  /* initialize char array s2 */
    char s3[40] = "";        /* initialize char array s3 */

    printf( "s1 = %s\ns2 = %s\n", s1, s2 );
    /* concatenate s2 to s1 */
    printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
    /* concatenate first 6 characters of s1 to s3. Place '\0'
     after last character */
    printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
    /* concatenate s1 to s3 */
    printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
    return 0; /* indicates successful termination */
} /* end main */

s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
`strcmp` returns an integer that is less than zero, equal to zero, or greater than zero depending on whether `string1` is less than, equal to, or greater than `string2`.

```c
strcmp(string1,string2);
```

- String comparison is done character by character using the ASCII numerical code
Here are some examples of string functions in action:

```c
static char s1[]="big sky country";
static char s2[]="blue moon";
static char s3[]="then falls Caesar";
```

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>strlen(s1)</td>
<td>15 /* e-o-s not counted */</td>
</tr>
<tr>
<td>strlen(s2)</td>
<td>9</td>
</tr>
<tr>
<td>strcmp(s1,s2)</td>
<td>negative number</td>
</tr>
<tr>
<td>strcmp(s3,s2)</td>
<td>positive number</td>
</tr>
<tr>
<td>strcat(s2,&quot; tonight&quot;)</td>
<td>blue moon tonight</td>
</tr>
</tbody>
</table>
8.3 Character Handling Library

• Character handling library
  – Includes functions to perform useful tests and manipulations of character data
  – Each function receives a character (an `int`) or EOF as an argument

• The following slide contains a table of all the functions in `<ctype.h>`
## 8.3 Character Handling Library

<table>
<thead>
<tr>
<th>Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int isdigit(int c);</td>
<td>Returns true if c is a digit and false otherwise.</td>
</tr>
<tr>
<td>int isalpha(int c);</td>
<td>Returns true if c is a letter and false otherwise.</td>
</tr>
<tr>
<td>int isalnum(int c);</td>
<td>Returns true if c is a digit or a letter and false otherwise.</td>
</tr>
<tr>
<td>int isxdigit(int c);</td>
<td>Returns true if c is a hexadecimal digit character and false otherwise.</td>
</tr>
<tr>
<td>int islower(int c);</td>
<td>Returns true if c is a lowercase letter and false otherwise.</td>
</tr>
<tr>
<td>int isupper(int c);</td>
<td>Returns true if c is an uppercase letter; false otherwise.</td>
</tr>
<tr>
<td>int tolower(int c);</td>
<td>If c is an uppercase letter, tolower returns c as a lowercase letter. Otherwise, tolower returns the argument unchanged.</td>
</tr>
<tr>
<td>int toupper(int c);</td>
<td>If c is a lowercase letter, toupper returns c as an uppercase letter. Otherwise, toupper returns the argument unchanged.</td>
</tr>
<tr>
<td>int isspace(int c);</td>
<td>Returns true if c is a white-space character—newline (\n'), space (' '), form feed (\f'), carriage return (\r'), horizontal tab (\t'), or vertical tab (\v')—and false otherwise.</td>
</tr>
<tr>
<td>int iscntrl(int c);</td>
<td>Returns true if c is a control character and false otherwise.</td>
</tr>
<tr>
<td>int ispunct(int c);</td>
<td>Returns true if c is a printing character other than a space, a digit, or a letter and false otherwise.</td>
</tr>
<tr>
<td>int isprint(int c);</td>
<td>Returns true value if c is a printing character including space (' ') and false otherwise.</td>
</tr>
<tr>
<td>int isgraph(int c);</td>
<td>Returns true if c is a printing character other than space (' ') and false otherwise.</td>
</tr>
</tbody>
</table>
/ * Fig. 8.2: fig08_02.c  
Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
    printf( "According to isdigit: ",
           isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
           isdigit( '9' ) ? "9 is a " : "9 is not a ", "digit" );

    printf( "According to isalpha:",
           isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
           isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
           isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
           isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter" );

    printf( "According to isalnum:",
           isalnum( 'A' ) ? "A is a " : "A is not a ",
           "digit or a letter",
           isalnum( '8' ) ? "8 is a " : "8 is not a ",
           "digit or a letter",
           isalnum( ''#'' ) ? "# is a " : "# is not a ",
           "digit or a letter" );
printf( "%s\n%s\n%s\n%s\n%s\n\n", "According to isxdigit:", isxdigit( 'F' ) ? "F is a " : "F is not a ", "hexadecimal digit", isxdigit( 'J' ) ? "J is a " : "J is not a ", "hexadecimal digit", isxdigit( '7' ) ? "7 is a " : "7 is not a ", "hexadecimal digit", isxdigit( '$' ) ? "$ is a " : "$ is not a ", "hexadecimal digit", isxdigit( 'f' ) ? "f is a " : "f is not a ", "hexadecimal digit" );

return 0; /* indicates successful termination */

} /* end main */
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit

• Program Output
/* Fig. 8.3: fig08_03.c  
   Using functions islower, isupper, tolower, toupper */
#include <stdio.h> 
#include <ctype.h> 

int main() 
{
    printf("According to islower:", 
           islower('p') ? "p is a " : "p is not a ", 
                   "lowercase letter",
           islower('P') ? "P is a " : "P is not a ", 
                   "lowercase letter",
           islower('5') ? "5 is a " : "5 is not a ", 
                   "lowercase letter",
           islower('!') ? "! is a " : "! is not a ", 
                   "lowercase letter" );

    printf("According to isupper:", 
           isupper('D') ? "D is an " : "D is not an ", 
                   "uppercase letter",
           isupper('d') ? "d is an " : "d is not an ", 
                   "uppercase letter",
           isupper('8') ? "8 is an " : "8 is not an ", 
                   "uppercase letter",
           isupper('$') ? "$ is an " : "$ is not an ", 
                   "uppercase letter" );
}
According to islower:
- p is a lowercase letter
- P is not a lowercase letter
- 5 is not a lowercase letter
- ! is not a lowercase letter

According to isupper:
- D is an uppercase letter
- d is not an uppercase letter
- 8 is not an uppercase letter
- $ is not an uppercase letter

- u converted to uppercase is U
- 7 converted to uppercase is 7
- $ converted to uppercase is $
- L converted to lowercase is l
/* Fig. 8.4: fig08_04.c
   Using functions isspace, iscntrl, ispunct, isprint, isgraph */
#include <stdio.h>
#include <ctype.h>

int main()
{
    printf( "%s\n%s\n\n%s\n\n%s\n\n", 
        "According to isspace:",
        "Newline", isspace('\n') ? " is a " : " is not a ",
        "whitespace character", "Horizontal tab",
        isspace('\t') ? " is a " : " is not a ",
        "whitespace character",
        isspace('%') ? "% is a " : "% is not a ",
        "whitespace character" );

    printf( "%s\n%s\n\n%s\n\n", 
        "According to iscntrl:",
        "Newline", iscntrl('\n') ? " is a " : " is not a ",
        "control character", iscntrl('$') ? "$ is a " :
        "$ is not a ", "control character" );
}
printf( "%s\n", "According to ispunct:"
    , ispunct(';') ? "; is a " : "; is not a ",
    "punctuation character",
    ispunct('Y') ? "Y is a " : "Y is not a ",
    "punctuation character",
    ispunct('#') ? "# is a " : "# is not a ",
    "punctuation character" );

printf( "%s\n", "According to isprint:"
    , isprint('$') ? "$ is a " : "$ is not a ",
    "printing character",
    "Alert", isprint('\a') ? " is a " : " is not a ",
    "printing character" );

printf( "%s\n", "According to isgraph:"
    , isgraph('Q') ? "Q is a " : "Q is not a ",
    "printing character other than a space",
    "Space", isgraph(' ') ? " is a " : " is not a ",
    "printing character other than a space" );

    return 0; /* indicates successful termination */
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
8.4 String Conversion Functions

- Conversion functions
  - In `<stdlib.h>` (general utilities library)
- Convert strings of digits to integer and floating-point values

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double atof( const char *nPtr );</code></td>
<td>Converts the string nPtr to double.</td>
</tr>
<tr>
<td><code>int atoi( const char *nPtr );</code></td>
<td>Converts the string nPtr to int.</td>
</tr>
<tr>
<td><code>long atol( const char *nPtr );</code></td>
<td>Converts the string nPtr to long int.</td>
</tr>
<tr>
<td><code>double strtod( const char *nPtr, char **endPtr );</code></td>
<td>Converts the string nPtr to double.</td>
</tr>
<tr>
<td><code>long strtol( const char *nPtr, char **endPtr, int base );</code></td>
<td>Converts the string nPtr to long.</td>
</tr>
<tr>
<td><code>unsigned long strtoul( const char *nPtr, char **endPtr, int base );</code></td>
<td>Converts the string nPtr to unsigned long.</td>
</tr>
</tbody>
</table>
```c
/* Fig. 8.6: fig08_06.c 
Using atof */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    double d; /* variable to hold converted string */

    d = atof("99.0");

    printf("The string "99.0" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );

    return 0; /* indicates successful termination */
}
```

The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```c
/* Fig. 8.7: fig08_07.c
   Using atoi */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i; /* variable to hold converted string */

    i = atoi("2593");

    printf("The string "2593" converted to int is ", i,
            "The converted value minus 593 is ", i - 593);

    return 0; /* indicates successful termination */
}
```

The string "2593" converted to int is 2593
The converted value minus 593 is 2000
/* Fig. 8.8: fig08_08.c
Using atol */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long l; /* variable to hold converted string */

    l = atol("1000000");

    printf("The string \"1000000\" converted to long int is ", l,
           "The converted value divided by 2 is ", l / 2);

    return 0; /* indicates successful termination */
}

The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
/* Fig. 8.9: fig08_09.c 
   Using strtod */ 
#include <stdio.h> 
#include <stdlib.h> 

int main() 
{
    /* initialize string pointer */
    const char *string = "51.2% are admitted";

    double d;       /* variable to hold converted sequence */
    char *stringPtr; /* create char pointer */

    d = strtod( string, &stringPtr );

    printf( "The string \"%s\" is converted to the\n", string );
    printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );

    return 0; /* indicates successful termination */
} /* end main */

The string "51.2% are admitted" is converted to the double value 51.20 and the string "% are admitted"
/* Fig. 8.10: fig08_10.c */
#include <stdio.h>
#include <stdlib.h>

int main()
{
    const char *string = "-1234567abc"; /* initialize string pointer */
    char *remainderPtr; /* create char pointer */
    long x; /* variable to hold converted sequence */

    x = strtol( string, &remainderPtr, 0 );

    printf("The original string is ", string,
           "The converted value is ", x,
           "The remainder of the original string is ",
           remainderPtr,
           "The converted value plus 567 is ", x + 567 );

    return 0; /* indicates successful termination */
}

The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
/* Fig. 8.11: fig08_11.c 
   Using strtoul */ 
#include <stdio.h> 
#include <stdlib.h> 

int main() 
{ 
    const char *string = "1234567abc"; /* initialize string pointer */ 
    unsigned long x; /* variable to hold converted sequence */ 
    char *remainderPtr; /* create char pointer */ 

    x = strtoul( string, &remainderPtr, 0 );

    printf( "%s\n%s\n%s\n%s\n%s\n%s\n%s\n", 
        "The original string is ", string, 
        "The converted value is ", x, 
        "The remainder of the original string is ", 
        remainderPtr, 
        "The converted value minus 567 is ", x - 567 );

    return 0; /* indicates successful termination */ 
} /* end main */

The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000
8.7 Comparison Functions of the String Handling Library

• Comparing strings
  – Computer compares numeric ASCII codes of characters in string
  – Appendix D has a list of character codes

```c
int strcmp( const char *s1, const char *s2 );
```
  – Compares string `s1` to `s2`
  – Returns a negative number if `s1 < s2`, zero if `s1 == s2` or a positive number if `s1 > s2`

```c
int strncmp( const char *s1, const char *s2, size_t n );
```
  – Compares up to `n` characters of string `s1` to `s2`
  – Returns values as above
/* Fig. 8.21: fig08_21.c
   Using strcmp and strncmp */
#include <stdio.h>
#include <string.h>

int main()
{
    const char *s1 = "Happy New Year"; /* initialize char pointer */
    const char *s2 = "Happy New Year"; /* initialize char pointer */
    const char *s3 = "Happy Holidays"; /* initialize char pointer */

    printf("%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n\n", "s1 = ", s1, "s2 = ", s2, "s3 = ", s3,
                   "strcmp(s1, s2) = ", strcmp( s1, s2 ),
                   "strcmp(s1, s3) = ", strcmp( s1, s3 ),
                   "strcmp(s3, s1) = ", strcmp( s3, s1 )
               );

    printf("%s\n%s\n%s\n%s\n%s\n%s\n\n", "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ),
                  "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ),
                  "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 )
              );

    return 0; /* indicates successful termination */
}
/* end main */
\begin{verbatim}
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) = 0
strcmp(s1, s3) = 1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) = 0
strncmp(s1, s3, 7) = 1
strncmp(s3, s1, 7) = -1
\end{verbatim}
## 8.8 Search Functions of the String Handling Library

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>char *strchr(const char *s, int c)</strong>;</td>
<td>Locates the first occurrence of character c in string s. If c is found, a pointer to c in s is returned. Otherwise, a NULL pointer is returned.</td>
</tr>
<tr>
<td><strong>size_t strcspn(const char *s1, const char *s2)</strong>;</td>
<td>Determines and returns the length of the initial segment of string s1 consisting of characters not contained in string s2.</td>
</tr>
<tr>
<td><strong>size_t strspn(const char *s1, const char *s2)</strong>;</td>
<td>Determines and returns the length of the initial segment of string s1 consisting only of characters contained in string s2.</td>
</tr>
<tr>
<td><strong>char *strpbrk(const char *s1, const char *s2)</strong>;</td>
<td>Locates the first occurrence in string s1 of any character in string s2. If a character from string s2 is found, a pointer to the character in string s1 is returned. Otherwise, a NULL pointer is returned.</td>
</tr>
<tr>
<td><strong>char *strrchr(const char *s, int c)</strong>;</td>
<td>Locates the last occurrence of c in string s. If c is found, a pointer to c in string s is returned. Otherwise, a NULL pointer is returned.</td>
</tr>
<tr>
<td><strong>char *strstr(const char *s1, const char *s2)</strong>;</td>
<td>Locates the first occurrence in string s1 of string s2. If the string is found, a pointer to the string in s1 is returned. Otherwise, a NULL pointer is returned.</td>
</tr>
<tr>
<td><strong>char *strtok(char *s1, const char *s2)</strong>;</td>
<td>A sequence of calls to strtok breaks string s1 into “tokens”—logical pieces such as words in a line of text—separated by characters contained in string s2. The first call contains s1 as the first argument, and subsequent calls to continue tokenizing the same string contain NULL as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, NULL is returned.</td>
</tr>
</tbody>
</table>
/* Fig. 8.23: fig08_23.c */
#include <stdio.h>
#include <string.h>

int main()
{
    const char *string = "This is a test"; /* initialize char pointer */
    char character1 = 'a'; /* initialize character1 */
    char character2 = 'z'; /* initialize character2 */

    /* if character1 was found in string */
    if ( strchr( string, character1 ) != NULL ) {
        printf( "\%c\ was found in \%s\n", character1, string );
    } /* end if */
    else { /* if character1 was not found */
        printf( "\%c\ was not found in \%s\n", character1, string );
    } /* end else */
}
/* if character2 was found in string */
if ( strchr( string, character2 ) != NULL ) {
    printf( "'%c' was found in "%s".
", 
            character2, string );
} /* end if */
else { /* if character2 was not found */
    printf( "'%c' was not found in "%s".
", 
            character2, string );
} /* end else */

return 0; /* indicates successful termination */

} /* end main */
#include <stdio.h>
#include <string.h>

int main()
{
    /* initialize two char pointers */
    const char *string1 = "The value is 3.14159";
    const char *string2 = "1234567890";

    printf("%s%s\n%s%s\n%s\n%s%u", "string1 = ", string1, "string2 = ", string2, "The length of the initial segment of string1", "containing no characters from string2 = ",\n    "strcspn(string1, string2 ) ");

    return 0; /* indicates successful termination */
}

string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1 containing no characters from string2 = 13
/* Fig. 8.25: fig08_25.c 
Using strpbrk */

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

int main()
{
    const char *string1 = "This is a test"; /* initialize char pointer */
    const char *string2 = "beware"; /* initialize char pointer */

    printf("Of the characters in ", string2,
           *strpbrk( string1, string2 ),
           " is the first character to appear in ", string1);

    return 0; /* indicates successful termination */
}

Of the characters in "beware"
'a' is the first character to appear in
"This is a test"
/* Fig. 8.26: fig08_26.c 
Using strchr */
#include <stdio.h>
#include <string.h>

int main()
{
    /* initialize char pointer */
    const char *string1 = "A zoo has many animals "
    "including zebras";
    int c = 'z'; /* initialize c */

    printf("%s\n%s%c%s"%s"
", "The remainder of string1 beginning with the",
        "last occurrence of character ", c,
        " is: ", strchr(string1, c));

    return 0; /* indicates successful termination */
}

The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
/* Fig. 8.27: fig08_27.c
   Using strspn */
#include <stdio.h>
#include <string.h>

int main()
{
    /* initialize two char pointers */
    const char *string1 = "The value is 3.14159";
    const char *string2 = "aehi lsTuv";

    printf("%s\n%s\n%s\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The length of the initial segment of string1",
           "containing only characters from string2 = ",
           strspn( string1, string2 ) );

    return 0; /* indicates successful termination */
}

string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1 containing only characters from string2 = 13
/* Fig. 8.28: fig08_28.c
   Using strstr */

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

int main()
{
    const char *string1 = "abcdefabcdef"; /* initialize char pointer */
    const char *string2 = "def"; /* initialize char pointer */

    printf("%s\n%s\n%s\n%s\n", "string1 = ", string1, "string2 = ", string2, "The remainder of string1 beginning with the", "first occurrence of string2 is: ", strstr(string1, string2));

    return 0; /* indicates successful termination */
}

string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the first occurrence of string2 is: defabcdef
/* Fig. 8.29: fig08_29.c 
   Using strtok */ 

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

int main() 
{
    /* initialize array string */
    char string[] = "This is a sentence with 7 tokens";
    char *tokenPtr; /* create char pointer */

    printf("%s\n%s\n%s\n", "The string to be tokenized is:", string, "The tokens are:" );

    tokenPtr = strtok( string, " "); /* begin tokenizing sentence */

    /* continue tokenizing sentence until tokenPtr becomes NULL */
    while ( tokenPtr != NULL ) { 
        printf( "%s\n", tokenPtr );
        tokenPtr = strtok( NULL, " "); /* get next token */
    } /* end while */

    return 0; /* indicates successful termination */
} /* end main */
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This is a sentence with 7 tokens