C Characters and Strings

Imran Ali

2020-07-05

Contents

NOTE:

1. Contents presented in this handout have been taken from the book C

How to program by Deitel and Deitel (5th Edition)

2. Description of `strtol` function and related code in listing-5 has been taken from cplusplus.com

C Characters and Strings

Introduction

Topics covered here include C Standard Library functions that facilitate string and character processing. The functions enable pograms to process characters, strings, lines of text and blocks of memory.

Techniques discussed here can be used to develop editors, word processors, page layout software, computerized typesetting systems and other kinds of text processing software.

Fundamentals of Strings and Characters

Characters are the fundamental building blocks of source programs. Every program is composed of a sequence of characters that is interpreted by the computer as a series of instructions used to accomplish a task. A program may contain character constants. **A character constant is an int value represented as a character in single quotes. The value of a character constant is the integer value of the character in the machine's character set.** For example, 'z' represents the integer value of z, and '$\backslash n$' the integer value of newline(122 and 10 in ASCII respectively)

A **string** is a series of characters treated as a single unit. A string may include letters, digits and various special characters such as +, -, *, / and $. String literals, or string constants in C are written in double quotation marks e.g. "University of Balochistan".

A string in C is an array of characters ending in the null character ('\0'). A string is accessed via a pointer to the first character in the string. The value of a string is the address of its first character. Thus in C it is appropriate to say that a string is a pointer, in fact, a pointer to the string's first character. In this sense strings are like arrays, because an array is also a pointer to its first element.

A character array or a variable of type char * can be initialized with a string in a definition. The definitions

```
char color[]= "blue";
const char *colorPtr = "blue";
```

each initialize a variable to the string blue. The first definition creates a 5 element array color containing the characters 'b', 'l', 'u', 'e', and '\0'. The

second definition creates pointer variable colorPtr that points to the string "blue" somewhere in memory.

The preceding array definition could also have been written

```
char color[] = {'b', 'l', 'u', 'e', '\0'};
```

When defining a character array to contain a string, the array must be large enough to store the string and its terminating null character. The preceding definition automatically determines the size of the array based on the number of initializers in the initializer list.

Character-Handling Library

The character-handling library (<ctype.h>) includes several functions that perform useful tests and manipulations of character data. Each function receives an unsigned char (represented as an int) or EOF as an argument. Characters are often manipulated as integers, because a character in C is a one-byte integer. EOF normally has the value –1. Following table summarizes the functions of the character-handling library.

Prototype	Function description
int isblank(int c);	Returns a true value if c is a blank character that separates words in a line of text and 0 (false) otherwise.
int isdigit(int c);	Returns a true value if c is a digit and 0 (false) otherwise.
int isalpha(int c);	Returns a true value if c is a letter and 0 otherwise.
int isalnum(int c);	Returns a true value if c is a digit or a letter and 0 otherwise.
int islower(int c);	int isxdigit(int c); Returns a true value if c is a hexadecimal digit character and 0 otherwise.
int islower(int c);	Returns a true value if c is a lowercase letter and 0 otherwise
int isupper(int c);	Returns a true value if c is an uppercase letter and 0 otherwise.
int tolower(int c);	If c is an uppercase letter, tolower returns c as a lowercase letter. Otherwise, tolower returns the argument unchanged.
int toupper(int c);	If c is a lowercase letter, toupper returns c as an uppercase letter.Otherwise, toupper returns the argument unchanged.
int isspace(int c);	Returns a true value if c is a whitespace character—newline ('$\backslash n$'), space (' '), form feed ('$\backslash f$'), carriage return ('$\backslash r$'), horizontal tab ('$\backslash t$') or vertical tab ('$\backslash v$')—and 0 otherwise.

Prototype	Function description
int iscntrl(int c);	Returns a true value if c is a control character and 0 otherwise.
int ispunct(int c);	Returns a true value if c is a printing character other than a space, a digit, or a letter and returns 0 otherwise.
int isprint(int c);	Returns a true value if c is a printing character including a space and returns 0 otherwise.
int isgraph(int c);	Returns a true value if c is a printing character other than a space and returns 0 otherwise.

Functions isdigit, isalpha, isalnum and isxdigit

1. Function isdigit determines whether its argument is a digit (0-9).
2. Function isalpha determines whether its argument is an uppercase (A-Z) or lowercase letter (a-z)
3. Function isalnum determines whether its argument is an uppercase letter, a lowercase letter, or a digit.
4. Function isxdigit determines whether its argument is a hexadecimal digit (A-F, a-f, 0-9).

Functions islower, isupper, tolower and toupper

1. Function islower determines whether its argument is a lowercase letter (a – z).
2. Function isupper determines whether its argument is an uppercase letter (A – Z).
3. Function tolower converts an uppercase letter to a lowercase letter and returns the lowercase letter. If the argument is not an uppercase letter, tolower returns the argument unchanged.
4. Function toupper converts a lowercase letter to an uppercase letter and returns the uppercase letter. If the argument is not a lowercase letter, toupper returns the argument unchanged.

Functions isspace, iscntrl, ispunct, isprint and isgraph

1. Function isspace determines whether a character is one of the following whitespace characters: space (' '), form feed ('\f'), newline ('\n'), carriage return ('\r'), horizontal tab('\t') or vertical tab ('\v').
2. Function iscntrl determines whether a character is one of the following control characters: horizontal tab, vertical tab , form feed('\f'), alert ('\a'), backspace ('\b'), carriage return ('\r') or newline ('\n').
3. Function ispunct determines whether a character is a printing character other than a space, a digit or a letter, such as $, # , (,) , [,] , { , } , ; , : or %

4. Function isprint determines whether a character can be displayed on the screen (including the space character).
5. Function isgraph is the same as isprint , except that the space character is not included

String Conversion Functions

This section presents the string-conversion functions from the general utilities library (<stdlib.h>) . These functions convert strings of digits to integer and floating-point values. Table below summarizes the string-conversion functions.

Function prototype	Function description
`double strtod( const char \*nPtr, char \*\*endPtr );`	Converts the string nPtr to double
`long int strtol(const char* str, char** endptr, int base );`	Converts the string str to long
`unsigned long strtoul( const char \*nPtr, char \*\*endPtr, int base );`	Converts the string nPtr to unsigned long

The C standard also includes strtoll and strtoull for converting strings to long long int and unsigned long long respectively. Note the use of const to declare variable nPtr in the function headers (read from right to left as "nPtr is a pointer to a character constant"); const specifies that the argument value will not be modified.

Function strtod

Function strtod converts a sequence of characters representing a floating-point value to double . The function returns 0 if it's unable to convert any portion of its first argument to double . The function receives two arguments—a string (char *) and a pointer to a string (char **). The string argument contains the character sequence to be converted to double any whitespace characters at the beginning of the string are ignored. The function uses the char ** argument to modify a char * in the calling function (stringPtr) so that it points to the location of the first character after the converted portion of the string or to the entire string if no portion can be converted.

Function strtol

Parses the C-string str interpreting its content as an integral number of the specified base, which is returned as a long int value. If endptr is not a null pointer, the function also sets the value of endptr to point to the first character after the number.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes as many characters as possible that are valid following a syntax that depends on the base parameter, and interprets them as a numerical value. Finally, a pointer to the first character following the integer representation in str is stored in the object pointed by endptr.

If the value of base is zero, the syntax expected is similar to that of integer constants, which is formed by a succession of:

- An optional sign character (+ or -)
- An optional prefix indicating octal or hexadecimal base ("0" or "0x"/"0X" respectively)
- A sequence of decimal digits (if no base prefix was specified) or either octal or hexadecimal digits if a specific prefix is present

If the base value is between 2 and 36, the format expected for the integral number is a succession of any of the valid digits and/or letters needed to represent integers of the specified radix (starting from '0' and up to 'z'/'Z' for radix 36). The sequence may optionally be preceded by a sign (either + or -) and, if base is 16, an optional "0x" or "0X" prefix.

If the first sequence of non-whitespace characters in str is not a valid integral number as defined above, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

Function strtoul

Function strtoul converts to unsigned long int a sequence of characters representing an unsigned long int value. The function works identically to function strtol. The statement `x = strtoul( string, &remainderPtr, 0 );` of example code indicates that x is assigned the unsigned long int value converted from string . The second argument, &remainderPtr , is assigned the remainder of string after the conversion. The third argument, 0 , indicates that the value to be converted can be in octal, decimal or hexadecimal format

Standard Input/Output Library Functions

This section presents several functions from the standard input/output library (<stdio.h>) specifically for manipulating character and string data. Table below summarizes the character and string input/output functions of the standard input/output library.

Function prototype	Function description
int getchar(void);	Inputs the next character from the standard input and returns it as an integer.

Function prototype	Function description
char *fgets(char* s, int n, FILE *stream);	Inputs characters from the specified stream into the array s until a newline or end-of-file character is encountered, or until n - 1 bytes are read.
int putchar(int c);	Prints the character stored in c and returns it as an integer.
int puts(const char *s);	Prints the string s followed by a newline character. Returns a nonzero integer if successful, or EOF if an error occurs.
int sprintf(char *s, const char* format, …);	Equivalent to printf , except the output is stored in the array s instead of printed on the screen. Returns the number of characters written to s , or EOF if an error occurs.
int sscanf(char *s, const char* format, …);	Equivalent to scanf , except the input is read from the array s rather than from the keyboard. Returns the number of items successfully read by the function, or EOF if an error occurs.

Functions fgets and putchar

listing-7 uses functions fgets and putchar to read a line of text from the standard input (keyboard) and recursively output the characters of the line in reverse order. Function fgets reads characters from the standard input into its first argument an array of chars until a newline or the end-of-file indicator is encountered, or until the maximum number of characters is read. The maximum number of characters is one fewer than the value specified in fgets's second argument. The third argument specifies the stream from which to read characters—in this case, we use the standard input stream (stdin). A null character ('\0') is appended to the array when reading terminates.

Function putchar prints its character argument. The program calls recursive function reverse to print the line of text backward. If the first character of the array received by reverse is the null character '\0' , reverse returns. Otherwise, reverse is called again with the address of the subarray beginning at element sPtr[1] , and character sPtr[0] is output with putchar when the recursive call is completed. The order of the two statements in the else portion of the if statement causes reverse to walk to the terminating null character of the string before a character is printed. As the recursive calls are completed, the characters are output in reverse order.

Function getchar

listing-8 uses functions getchar and puts to read characters from the standard input into character array sentence and display the characters as a string. Function getchar reads a character from the standard input and returns the character as an integer. As you know, function puts takes a string as an argument and displays the string followed by a newline character. The program stops inputting characters when either 79 characters have been read or when getchar reads the newline character entered by the user to end the line of text. A null character is appended to array sentence so that the array may be treated as a string. Then, line 17 uses puts to display the string contained in sentence.

Function sprintf

listing-9 uses function sprintf to print formatted data into array s —an array of characters. The function uses the same conversion specifiers as printf. The program inputs an int value and a double value to be formatted and printed to array s . Array s is the first argument of sprintf . [Note: If your system supports snprintf_s, then use that in preference to sprintf . If your system doesn't support snprintf_s but does support snprintf , then use that in preference to sprintf.]

Function sscanf

listing-10 uses function sscanf to read formatted data from character array s . The function uses the same conversion specifiers as scanf . The program reads an int and a double from array s and stores the values in x and y , respectively. The values of x and y are printed. Array s is the first argument of sscanf.

String-Manipulation Functions of the String-Handling Library

The string-handling library (<string.h>) provides many useful functions for manipulating string data (copying strings and concatenating strings), comparing strings, searching strings for characters and other strings, tokenizing strings (separating strings into logical pieces) and determining the length of strings. This section presents the string-manipulation functions of the string-handling library. The functions are summarized in table below:

Function prototype	Function description
char *strcpy(char* s1, const char *s2)	Copies string s2 into array s1 . The value of s1 is returned.
char *strncpy(char* s1, const char *s2, size_t n)	Copies at most n characters of string s2 into array s1 . The value of s1 is returned.

Function prototype	Function description
char *strcat(char* s1, const char *s2)	Appends string s2 to array s1 . The first character of s2 overwrites the terminating null character of s1 . The value of s1 is returned.
char *strncat(char* s1, const char *s2, size_t n)	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.

Every function—except for strncpy —appends the null character to its result.

Functions strncpy and strncat specify a parameter of type size_t, which is a type defined by the C standard as the integral type of the value returned by operator `sizeof`.

Function strcpy copies its second argument (a string) into its first argument—a character array that you must ensure is large enough to store the string and its terminating null character, which is also copied.

Function strncpy is equivalent to strcpy , except that strncpy specifies the number of characters to be copied from the string into the array.

Function strncpy does not necessarily copy the terminating null character of its second argument. This occurs only if the number of characters to be copied is at least one more than the length of the string.

For example, if “test” is the second argument, a terminating null character is written only if the third argument to strncpy is at least 5 (four characters in “test” plus a terminating null character). If the third argument is larger than 5 , null characters are appended to the array until the total number of characters specified by the third argument are written.

Functions strcpy and strncpy

listing-11 uses strcpy to copy the entire string in array x into array y and uses strncpy to copy the first 14 characters of array x into array z . A null character (‘\0’) is appended to array z , because the call to strncpy in the program does not write a terminating null character (the third argument is less than the string length of the second argument).

Comparison Functions of the String-Handling Library

This section presents the string-handling library’s string-comparison functions, strcmp and strncmp . Table below contains their prototypes and a brief description of each function.

Function prototype	Function description
`int strcmp( const char *s1, const char *s2 );`	Compares the string s1 with the string s2 . The function returns 0 , less than 0 or greater than 0 if s1 is equal to, less than or greater than s2 , respectively.
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to n characters of the string s1 with the string s2 . The function returns 0 , less than 0 or greater than 0 if s1 is equal to, less than or greater than s2 , respectively.

listing-19 compares three strings using strcmp and strncmp. Function strcmp compares its first string argument with its second string argument, character by character. The function returns 0 if the strings are equal, a negative value if the first string is less than the second string and a positive value if the first string is greater than the second string. Function strncmp is equivalent to strcmp , except that strncmp compares up to a specified number of characters. Function strncmp does not compare characters following a null character in a string. The program prints the integer value returned by each function call.

To understand just what it means for one string to be "greater than" or "less than" another, consider the process of alphabetizing a series of last names. The reader would, no doubt, place "Jones" before "Smith," because the first letter of "Jones" comes before the first letter of "Smith" in the alphabet. But the alphabet is more than just a list of 26 letters—it's an ordered list of characters. Each letter occurs in a specific position within the list. "Z" is more than merely a letter of the alphabet; "Z" is specifically the 26th letter of the alphabet. How do the string comparison functions know that one particular letter comes before another? All characters are represented inside the computer as numeric codes in character sets such as ASCII and Unicode; when the computer compares two strings, it actually compares the numeric codes of the characters in the strings.

Search Functions of the String-Handling Library

This section presents the functions of the string-handling library used to search strings for characters and other strings. The functions are summarized in table below. The functions strcspn and strspn return size_t .

Function prototypes and descriptions

1. `size_t strspn( const char *s1, const char *s2 );` Determines and returns the length of the initial segment of string s1 consisting only of characters contained in string s2 .
2. `char *strpbrk( const char *s1, const char *s2 );` 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.

3. `char *strrchr( const char *s, int c );` 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.

4. `char *strstr( const char *s1, const char *s2 );` 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.

5. `char *strtok( char *s1, const char *s2 );` 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.

Function strchr

Function strchr searches for the first occurrence of a character in a string. If the character is found, strchr returns a pointer to the character in the string; otherwise, strchr returns NULL . Listing-12 searches for the first occurrences of 'a' and 'z' in "This is a test" .

Function strcspn

Function strcspn listing-13 determines the length of the initial part of the string in its first argument that does not contain any characters from the string in its second argument. The function returns the length of the segment.

Function strpbrk

Function strpbrk searches its first string argument for the first occurrence of any character in its second string argument. If a character from the second argument is found, strpbrk returns a pointer to the character in the first argument; otherwise, strpbrk returns NULL . listing-14 shows a program that locates the first occurrence in string1 of any character from string2 .

Function strrchr

Function strrchr searches for the last occurrence of the specified character in a string. If the character is found, strrchr returns a pointer to the character in the string; otherwise, strrchr returns NULL . listing-15 shows a program that searches for the last occurrence of the character 'z' in the string "A zoo has many animals including zebras" .

Function strspn

Function strspn listing-16 determines the length of the initial part of the string in its first argument that contains only characters from the string in its second argument. The function returns the length of the segment.

Function strstr

Function strstr searches for the first occurrence of its second string argument in its first string argument. If the second string is found in the first string, a pointer to the location of the string in the first argument is returned. listing-17 uses strstr to find the string "def" in the string "abcdefabcdef" .

Function strtok

Function strtok is used to break a string into a series of tokens. A token is a sequence of characters separated by delimiters (usually spaces or punctuation marks, but a delimiter can be any character).
For example, in a line of text, each word can be considered a token, and the spaces and punctuation separating the words can be considered delimiters.

Multiple calls to strtok are required to tokenize a string i.e., break it into tokens (assuming that the string contains more than one token).

In listing-18 the first call to strtok contains two arguments:
1. a string to be tokenized, and 2. a string containing characters that separate the tokens.

In line 11, the statement
`tokenPtr = strtok( string, " " ); // begin tokenizing sentence`
assigns tokenPtr a pointer to the first token in string .
The second argument, " " , indicates that tokens are separated by spaces.

Function strtok searches for the first character in string that's not a delimiting character (space). This begins the first token. The function then finds the next delimiting character in the string and replaces it with a null ('\0') character to terminate the current token.

Function strtok saves a pointer to the next character following the token in string and returns a pointer to the current token.

Subsequent strtok calls in line 15 continue tokenizing string . These calls contain NULL as their first argument.

The NULL argument indicates that the call to strtok should continue tokenizing from the location in string saved by the last call to strtok .

If no tokens remain when strtok is called, strtok returns NULL . You can change the delimiter string in each new call to strtok .

listing-18 uses strtok to tokenize the string “This is a sentence with 7 tokens” . Each token is printed separately.

Function strtok modifies the input string by placing ‘\0’ at the end of each token; therefore, a copy of the string should be made if the string will be used again in the program after the calls to strtok.

Programming Examples

1. Character functions isdigit, isalpha, isalnum, isxdigit

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

int main(void)
{
    printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
isdigit('8') ? "8 is a " : "8 is not a ", "digit",
isdigit('#') ? "# is a " : "# is not a ", "digit" );

 printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
 "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( "%s\n%s%s\n%s%s\n%s%s\n\n",
"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%s\n%s%s\n%s%s\n%s%s\n%s%s\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;
}
```

2. Character functions islower, isupper, tolower, toupper

```
#include <stdio.h>
#include <ctype.h>
int main( void )
{
    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
            "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( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
            "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" );
    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
            "u converted to uppercase is ", toupper('u') ,
            "7 converted to uppercase is ", toupper('7') ,
            "$ converted to uppercase is ", toupper('$') ,
            "L converted to lowercase is ", tolower('L') );
    return 0;
}
```

3. Character functions isspace, iscntrl, ispunct, isprint, and isgraph

```
#include <stdio.h>
#include <ctype.h>
int main( void )
{
    printf( "%s\n%s%s%s\n%s%s%s\n%s%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%s%s\n%s%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%s%s\n%s%s\n%s%s\n\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%s%s\n%s%s%s\n\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%s%s\n%s%s%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;
}
```

4. Function strtod

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    // initialize string pointer
    const char *string = "51.2% are admitted"; // initialize string
    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;
}
```

5. Function strtol

```c
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
  char szNumbers[] = "2001 60c0c0 -110111010011010010000 0x6fffff";
  char * pEnd;
  long int li1, li2, li3, li4;
  li1 = strtol (szNumbers,&pEnd,10);
  li2 = strtol (pEnd,&pEnd,16);
  li3 = strtol (pEnd,&pEnd,2);
  li4 = strtol (pEnd,NULL,0);
  printf ("The decimal equivalents are: %ld, %ld, %ld and %ld.\n", li1, li2, li3, li4);
  return 0;

}
```

6. Function strtoul

```c
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
    const char *string = "1234567abc"; // initialize string pointer
    unsigned long int x; // variable to hold converted sequence
    char *remainderPtr; // create char pointer
    x = strtoul( string, &remainderPtr, 0 );
    printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\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;
}
```

7. Functions fgets and putchar

```c
#include <stdio.h>
#define SIZE 80
void reverse( const char * const sPtr ); // prototype
int main( void )
{
    char sentence[ SIZE ]; // create char array
    puts( "Enter a line of text:" );
    // use fgets to read line of text
    fgets( sentence, SIZE, stdin );
```

```
    puts( "\nThe line printed backward is:" );
    reverse( sentence );
} // end main

// recursively outputs characters in string in reverse order
void reverse( const char * const sPtr )
{
    // if end of the string
    if ( '\0' == sPtr[ 0 ] ) { // base case
        return;
    } // end if
    else { // if not end of the string
        reverse( &sPtr[ 1 ] ); // recursion step
        putchar( sPtr[ 0 ] ); // use putchar to display character
    } // end else
}
```

8. Function getchar

```
#include <stdio.h>
#define SIZE 80
int main( void )
{
    int c; // variable to hold character input by user
    char sentence[ SIZE ]; // 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 ( i < SIZE - 1 && ( c = getchar() ) != '\n' ) {
        sentence[ i++ ] = c;
    } // end while
    sentence[ i ] = '\0'; // terminate string
    // use puts to display sentence
    puts( "\nThe line entered was:" );
    puts( sentence );
} // end main
```

9. Function sprintf

```
#include <stdio.h>
#define SIZE 80
int main( void )
{
    char s[ SIZE ]; // create char array
    int x; // x value to be input
    double y; // y value to be input
    puts( "Enter an integer and a double:" );
```

```
    scanf( "%d%lf", &x, &y );
    sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );
    printf( "%s\n%s\n",
            "The formatted output stored in array s is:", s );
}
```

10. Function sscanf

```
#include <stdio.h>
int main( void )
{
    char s[] = "31298 87.375"; // initialize array s
    int x; // x value to be input
    double y; // y value to be input
    sscanf( s, "%d%lf", &x, &y );
    printf( "%s\n%s%6d\n%s%8.3f\n",
            "The values stored in character array s are:",
            "integer:", x, "double:", y );
}
```

11. Functions strcpy and strncpy

```
#include <stdio.h>
#include <string.h>
#define SIZE1 25
#define SIZE2 15
int main( void )
{
    char x[] = "Happy Birthday to You"; // initialize char array x
    char y[ SIZE1 ]; // create char array y
    char z[ SIZE2 ]; // create char array z
    // copy
    printf(
            "The
            "The
            contents of x into y
            "%s%s\n%s%s\n",
            string in array x is: ", x,
            string in array y is: ", strcpy( y, x ) );
    // copy first 14 characters of x into z. Does not copy null
    // character
    strncpy( z, x, SIZE2 - 1 );
    z[ SIZE2 - 1 ] = '\0'; // terminate string in z
    printf( "The string in array z is: %s\n", z );
    return 0;
}
```

12. Function strchr

```c
#include <stdio.h>
#include <string.h>
int main( void )
{
    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\".\n",
                character2, string );
    } // end if
    else { // if character2 was not found
        printf( "\'%c\' was not found in \"%s\".\n",
                character2, string );
    } // end else
    return 0;
} // end main
```

13. Function strcspn

```c
#include <stdio.h>
#include <string.h>
int main( void )
{
    // initialize two char pointers
    const char *string1 = "The value is 3.14159";
    const char *string2 = "1234567890";
    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
            "string1 = ", string1, "string2 = ", string2,
            "The length of the initial segment of string1",
            "containing no characters from string2 = ",
            strcspn( string1, string2 ) );
    return 0;
} // end main
```

14. Function strpbrk

```c
#include <stdio.h>
```

```
#include <string.h>
int main( void )
{
    const char *string1 = "This is a test"; // initialize char pointer
    const char *string2 = "beware"; // initialize char pointer
    printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
            "Of the characters in ", string2,
            *strpbrk( string1, string2 ) ,
            " appears earliest in ", string1 );
    return 0;
} // end main
```

15. Function strrchr

```
#include <stdio.h>
#include <string.h>
int main( void )
{
    // initialize char pointer
    const char *string1 = "A zoo has many animals including zebras";
    int c = 'z'; // character to search for
    printf( "%s\n%s'%c'%s\"%s\"\n",
            "The remainder of string1 beginning with the",
            "last occurrence of character ", c,
            " is: ", strrchr( string1, c ) );
    return 0;
} // end main
```

16. Function strspn

```
#include <stdio.h>
#include <string.h>
int main( void )
{
    // initialize two char pointers
    const char *string1 = "The value is 3.14159";
    const char *string2 = "aehi lsTuv";
    printf( "%s%s\n%s%s\n\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;
} // end main
```

17. Function strstr

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

```
int main( void )
{
    // initialize two char pointers
    const char *string1 = "The value is 3.14159";
    const char *string2 = "aehi lsTuv";
    printf( "%s%s\n%s%s\n\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;
} // end main
```

18. Function strtok

```
#include <stdio.h>
#include <string.h>
int main( void )
{
    // initialize array string
    char string[] = "This is a sentence with 7 tokens";
    char *tokenPtr; // create char pointer
    printf( "%s\n%s\n\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;
} // end main
```

19. Function strcmp and strncmp

```
#include <stdio.h>
#include <string.h>
int main( void )
{
    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%s\n%s%s\n%s%s\n\n%s%2d\n%s%2d\n%s%2d\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 ) );
```

```
13      printf("%s%2d\n%s%2d\n%s%2d\n",
14              "strncmp(s1, s3, 6) = ", strncmp( s1, s3, 6 ) ,
15              "strncmp(s1, s3, 7) = ", strncmp( s1, s3, 7 ) ,
16              "strncmp(s3, s1, 7) = ", strncmp( s3, s1, 7 ) );
17      return 0;
18  }
```