Strings

Usman Habib.

Fundamentals of Characters and Strings

- Character constant
  - Integer value of a character
  - Single quotes
  - **`'z'`** is the integer value of **`z`**, which is **`122`**
- String
  - Series of characters treated as one unit
  - Can include letters, digits, special characters **`+`**, **`-`**, **`*`** ...
  - String literal (string constants)
    - Enclosed in double quotes, for example:

      `"I like C++"`
  - Array of characters, ends with null character **`'\0'`**
  - Strings are constant pointers (like arrays)
    - Value of string is the address of its first character

5.12.1 Fundamentals of Characters and Strings

- String assignment
  - Character array:

    ```
    char color[] = "blue";
    ```

    - Creates 5 element **`char`** array, **`color`**, (last element is **`'\0'`**)
  - variable of type **`char *`**

    ```
    char *colorPtr = "blue";
    ```

    - Creates a pointer to string **`“blue”`**, **`colorPtr`**, and stores it somewhere in memory

Examples Using Arrays

- Strings
  - Arrays of characters
  - All strings end with **`null`** (`'\0'`)
  - Examples:

```
char string1[] = "hello";
char string1[] = { 'h', 'e', 'l', 'l', 'o',
                   '\0' };
```

  - Subscripting is the same as for a normal array

    `String1[ 0 ]` is **`'h'`**

    `string1[ 2 ]` is **`'l'`**

- Input from keyboard

```
char string2[ 10 ];
cin >> string2;
```

  - Takes user input
  - Side effect: if too much text entered, data written beyond array

```cpp
// Fig. 4_12: fig04_12.cpp
// Treating character arrays as strings
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main()
{
   char string1[ 20 ], string2[] = "string literal";

   cout << "Enter a string: ";
   cin >> string1;
   cout << "string1 is: " << string1
        << "\nstring2 is: " << string2
        << "\nstring1 with spaces between characters is:\n";

   for ( int i = 0; string1[ i ] != '\0'; i++ )
      cout << string1[ i ] << ' ';

   cin >> string1;  // reads "there"
   cout << "\nstring1 is: " << string1 << endl;

   cout << endl;
   return 0;
}
```

Inputted strings are separated by whitespace characters. **"there"** stayed in the buffer.

Notice how string elements are referenced like arrays.

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
string1 is: there
```

Fundamentals of Characters and Strings

- Reading strings
  - Assign input to character array **`word[ 20 ]`**

```
cin >> word
```

    - Reads characters until whitespace or EOF
    - String could exceed array size

```
cin >> setw( 20 ) >> word;
```

    - Reads 19 characters (space reserved for **`'\0'`**)

- **`cin.getline`**
  - Reads a line of text
  - Using **`cin.getline`**

```
cin.getline( array, size, delimiter
  character);
```

Fundamentals of Characters and Strings

- **`cin.getline`**
  - Copies input into specified array until either
    - One less than the size is reached
    - The delimiter character is input
  - Example

```
char sentence[ 80 ];
cin.getline( sentence, 80, '\n' );
```

String Manipulation Functions of the String-handling Library

- String handling library **`<cstring>`** provides functions to
  - Manipulate strings
  - Compare strings
  - Search strings
  - Tokenize strings (separate them into logical pieces)
- ASCII character code
  - Strings are compared using their character codes
  - Easy to make comparisons (greater than, less than, equal to)
- Tokenizing
  - Breaking strings into tokens, separated by user-specified characters
  - Tokens are usually logical units, such as words (separated by spaces)
  - **`"This is my string"`** has 4 word tokens (separated by spaces)

String Manipulation Functions of the String-handling Library

`char *strcpy( char *s1, const char *s2 );`	Copies the string **`s2`** into the character array **`s1`**. The value of **`s1`** is returned.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copies at most **`n`** characters of the string **`s2`** into the character array **`s1`**. The value of **`s1`** is returned.
`char *strcat( char *s1, const char *s2 );`	Appends the string **`s2`** to the string **`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 string **`s1`**. The first character of **`s2`** overwrites the terminating null character of **`s1`**. The value of **`s1`** is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string **`s1`** with the string **`s2`**. The function returns a value of zero, less than zero or greater than zero if **`s1`** is equal to, less than or greater than **`s2`**, respectively.

String Manipulation Functions of the String-handling Library (III)

`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 zero, less than zero or greater than zero if **`s1`** is equal to, less than or greater than **`s2`**, respectively.
`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—delimited 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.
`size_t strlen( const char *s );`	Determines the length of string **`s`**. The number of characters preceding the terminating null character is returned.

References

Dietal and Dietal : How to Program C++

3rd Edition