C Style Strings

Lecture 10
COP 3014 Spring 2022

March 21, 2022

Recap

- Recall that a C-style string is a character array that ends with the null character
- Character literals in single quotes
  - 'a', '\n', '$'
- string literals in double quotes
  - "Hello World\n"
  - Remember that the null-character is *implicitly* a part of any string literal
- The name of an array acts as a pointer to the first element of an array (i.e. it stores the address of where the array starts)
-

The cctype library

This C library contains useful character testing functions, as well as the two conversion functions

Conversion functions: These return the ascii value of a character

- `int toupper(int c)` - returns the uppercase version of c if it's a lowercase letter, otherwise returns c as is
- `int tolower(int c)` - returns the lowercase version of c if it's an uppercase letter, otherwise returns c as is

Query Functions: These all return true (non-zero) or false (0), in answer to the question posed by the function's name. They all take in the ascii value of a character as a parameter.

- `int isdigit(int c)` - decides whether the parameter is a digit (0-9)
- `int isalpha(int c)` - decides whether the character is a letter (a-z, A-Z)
- `int isalnum(int c)` - digit or a letter?

More Query functions

- `int islower(int c)` - lowercase digit? (a-z)
- `int isupper(int c)` - uppercase digit? (A-Z)
- `int isxdigit(int c)` - hex digit character? (0-9, a-f)
- `int isspace(int c)` - white space character?
- `int iscntrl(int c)` - control character?
- `int ispunct(int c)` - printing character other than space, letter, digit?
- `int isprint(int c)` - printing character (including ' ')?
- `int isgraph(int c)` - printing character other than ' ' (space)?

String I/O:

In the special case of arrays of type `char`, which are used to implement c-style strings, we can use these special cases with the insertion and extraction operators:

```
char greeting[20] = ‘‘Hello, World";
cout <<greeting; // prints ‘‘Hello, World"
char lastname[20];
cin >> lastname; // reads a string into ‘lastname’
// adds the null character automatically
```

- Using a char array with the insertion operator <<will print the contents of the character array, up to the first null character encountered
- The extraction operator >>used with a char array will read in a string, and will stop at white space.
- These examples only apply to the special case of the character array.

Reading strings: get and getline

- The above `cin` example is only good for reading one word at a time. What if we want to read in a whole sentence into a string?
- There are two more member functions in class `istream` (in the `iostream` library), for reading and storing C-style strings into arrays of type char. Here are the prototypes:

```
char* get(char str[], int length, char delimiter
                                    = '\n');

char* getline(char str[], int length, char
                                   delimiter = '\n');
```

Reading strings: get and getline

- The functions `get` and `getline` (with the three parameters) will read and store a c-style string. The parameters:
  - First parameter (str) is the char array where the data will be stored. Note that this is an array passed into a function, so the function has access to modify the original array
  - Second parameter (length) should always be the size of the array – i.e. how much storage available.
  - Third parameter (delimiter) is an optional parameter, with the newline as the default. This is the character at which to stop reading
- Both of these functions will extract characters from the input stream, but they don't stop at any white space – they stop at the specified delimiter. They also automatically append the null character, which must (as always) fit into the size of the array.

Sample Calls

```
char buffer[80];
cin >>buffer; // reads one word into buffer
cin.get(buffer, 80, ‘,’); // reads up to the first
                  //comma, stores in buffer
cin.getline(buffer, 80); // reads an entire line
                 // (up to newline)
```

So what is the difference between get and getline?

- get will leave the delimiter character on the input stream, and it will be seen by the next input statement
- getline will extract and discard the delimiter character

Example

```
char greeting[15], name[10], other[20];
cin.getline(greeting,15); // gets input into greeting
cin.get(name,10,'.'); // gets input into name
cin.getline(other,20); // gets input into other
```

Suppose that the data on the input stream (i.e. typed onto the keyboard, for instance) is:

```
Hello, World
Joe Smith.  He says hello.
```

At this point, the contents of each string are:

```
greeting:  "'Hello, World"
name:  "Joe Smith"
other:  ".  He says hello."
```

The `cstring` library

- The standard string library in C is called `cstring`.
- To use it, we place the appropriate `#include` statement in a code file:
  `#include <cstring>`
- This string library contains many useful string manipulation functions.
- These are all for use with C-style strings. A few of the more commonly used ones are mentioned here.
- You can get more information on the online documentation for the library on cplusplus.com

strlen

- Takes one string argument, returns its length (not counting the null character)
- Prototype:

```
int strlen(const char str[]);
```

- Sample calls:

```
char phrase[30] = "Hello, World";
cout <<strlen("Greetings, Earthling!");
                              // prints 21
int length = strlen(phrase); // stores 12
```

strcpy

- Takes two string arguments, copies the contents of the second string into the first string.
- The first parameter is non-constant, the second is constant
- Prototype:

```
char* strcpy(char str1[], const char str2[]);
                    // copies str2 into str 1
```

- Sample calls:

```
char buffer[80], firstname[30], lastname[30] =
                                    "Smith";
strcpy(firstname, "Billy Joe Bob");
              // copies name into firstname array
strcpy(buffer, lastname);
              // copies "Smith" into buffer array
cout <<firstname; // prints "Billy Joe Bob"
cout <<buffer; // prints "Smith"
```

strcat

- Takes two string arguments (first non-constant, second is const), and concatenates the second one onto the first
- Prototype:

```
char* strcat(char str1[], const char str2[]);
// concatenates str2 onto the end of str1
```

- Sample calls:

```
char buffer[80] = "Bat";
char word[] = "man";

strcat(buffer, word); // buffer is now "Batman"
strcat(buffer, " is awesome");
            // buffer is now "Batman is awesome"
```

strcmp

- Takes two string arguments (both passed as const arrays), and returns an integer that indicates their lexicographic order
- Prototype:

```
int strcmp(const char str1[], const char str2[]);

// returns:
// a negative number, if str1 comes before str2
// a positive number, if str2 comes before str1
// 0 , if they are equal
//
// Note:  Lexicographic order is by ascii codes.
// It's NOT the same
// as alphabetic order.
```

Sample calls:

```
char word1[30] = "apple";
char word2[30] = "apply";

if (strcmp(word1, word2) != 0)
     cout <<"The words are different\n";

strcmp(word1, word2)
      // returns a negative, means word1 comes first
strcmp(word1, "apple")
      // returns a 0.  strings are the same
strcmp("apple", "Zebra")
      // returns a positive.  "Zebra" comes first!
// (all uppercase before lowercase in ascii)
```

Memory Safe Calls

- Note that the above calls rely on the null character as the terminator of C-style strings. Remember, there is no built-in bounds checking in C++
- **strncpy, strncat, strncmp** - these do the same as the three listed above, but they take one extra argument (an integer N), and they go up to the null character or up to N characters, whichever is first.
- These functions can be used to help do safer string operations.
- The extra parameter can be included to guarantee that array boundaries are not exceeded, as seen in the following examples

Examples

```
char buffer[80];
char word[11] = "applesauce";
char bigword[] = "antidisestablishmentarianism";

strncpy(buffer, word, 5); // buffer is "apple"
strncat(buffer, " piecemeal", 4);
          // buffer now stores "apple pie"
strncmp(buffer, "apple", 5);
          // returns 0, as first 5 characters
          // of the strings are equal
strncpy(word, bigword, 10);
           // word is now "antidisest"
          // word only had 11 slots!
```