PSPDiC(9): Strings

Cory L. Strope

Chapter 9

Strings

- In using arrays, we have seen limited use of character data.
  - Using characters, we mostly deal with a grouping of characters,
  - Groups of characters in C are called a *string*.
- C implements the string data structure using arrays of type `char`,
- Strings are important in computer science because many computer applications are concerned with the manipulation of textual data rather than numerical data.

String Basic

- We have already used string constants extensively in our earlier work.
- Calls to `scanf` or `printf` used a string constant as the first argument.
  - `printf(`$\longrightarrow$`"Average = %.2f"`$\longleftarrow$`, avg)`
  - `"Average = %.2f"` is a string of 14 characters
- We can also use a `#define INSUFF_DATA "Insufficient Data"`
  - `"Insufficient Data"` is a string and can be printed with a `%s`.

Declaring and Initializing String Variables

- Declaring a string variable is the same as declaring an array of type `char`.
  - `char string_var[30];`
  - the variable `string_var` will hold strings anywhere from 0 to 29 characters long.
- Strings differ from other data structures in the way C handles them.
- `char str[20] = "Initial value";`

Declaring Cont.

```
char str[20] = "Initial value";
```

The above will produce the following in memory:

[0]				[4]					[9]				[14]			
I	n	i	t	i	a	l		v	a	l	u	e	\0	?	...	

- The \0 is the null character, which marks the end of the string.
  - C needs this to be present in every string in order to properly print the string.
- Thus, in the above declaration, `str` can only hold **19** characters since we need one spot for the null character.

Arrays of Strings

- Because one string is an array of characters, an array of strings is a two-dimensional array of characters in which each row is one string.
- The following are statements to declare an array to store up to 30 names, each of which is less than 25 characters long:

```
#define NUM_PEOPLE 30
#define NAME_LEN 25
char names[NUM_PEOPLE][NAME_LEN];
```

- We can initialize an array of strings at declaration in the following manner:

```
char month[12][10] = {"January", "February",
"March","April", "May", "June", "July", "August",
"September","October", "November", "December"};
```

Input/Output with printf and scanf

- Both `printf` and `scanf` can handle string arguments as long as the placeholder `%s` is used in the format string:
  - `printf("Topic:  %s\n", string_var);`
- Functions that take string arguments depend on finding a null character to mark the end of the string.
  - If `printf` were passed a character array that contained no `'\0'`, the function would first interpret the contents of each array element as a character and display it.
  - Then `printf` would continue to display as characters the content of memory locations following the array argument until it encountered a null character or until it attempted to access a memory cell that was not assigned to the program, causing a run-time error.

Input/Output Cont.

- We can justify strings either on the left side or the right:
  - Default is left justification. `%-4s` sets a 4 space buffer for the words.
  - To right justify, we type `%3s`. This will right justify all strings that have a null character at or before their third element.
- `scanf` takes string input in a manner similar to numeric input.
  - `scanf` skips leading whitespace characters such as blanks, newlines, and tabs.
  - Starting with the first non-whitespace character, `scanf` copies the characters it encounters into successive memory cells of its character array argument.
  - When a whitespace character is reached, scanning stops, and `scanf` places the null character at the end of the string in its array argument.
- `scanf("%s", string_var);`
- Notice that there is no &, this is because arrays point to memory locations already and there is no use for &.

String Library Functions: Assignment and Substrings

- So far, we have used the assignment operator "=" to copy data into a variable.
  - We use the assignment symbol in a declaration of a string variable for initialization, but this is the *only* context that is valid.
  - This is because arrays point to the first memory location an not the value.
- C provides functions to work with strings, located in the `string.h` file.
  - Now we to do a `#include <string.h>` at the top of the program.
- Table 9.1 on page 441 summarizes which functions are provided.

String Assignment

- Function `strcpy(char *str1, char *str2)` copies `str2` into `str1`.
  - EX: To copy `"Test String"` into variable `one_str`:
    `strcpy(one_str, "Test String");`
  - Like a call to `scanf` with a `%s` placeholder, a call to `strcpy()` can overflow the space allocated for the destination variable.
  - If `one_str` was declared as the following:
    `char one_str[20];`
    and we performed the following function:
    `strcpy(one_str, "A very long test string");`
    The final characters `'i'`, `'n'`, `'g'`, and `'\0'` would be overwritten.

Selected String Library Functions

- `strcat(src,dest)`: Appends `dest` onto `src`.
  - if `src[20] = "Source "`, `dest[] = "dest"`, and `strcat(src,dest);`, then: `printf("%s\n", src)` $\Rightarrow$ `Source dest`.
- `strcmp(s1,s2)`: Compares `s1` and `s2` alphabetically.
  - if `s1` $<$ `s2` $\Rightarrow$ `return` negative value.
  - if `s1` $>$ `s2` $\Rightarrow$ `return` positive value.
  - if `s1` $==$ `s2` $\Rightarrow$ `return 0`.
- `strlen(str)`: Returns the number of characters in `str`.
  - `strlen("What");` $\Rightarrow$ 4.

String Assignment

- C provides another copying function called `strncpy()`.
- This copies *n* values of the second value to the destination.
  `strncpy(one_str, "Test String", 4);`
- This copies 4 characters of `"Test String"` into the variable `one_str`.
- If the first *n* characters does not contain the null character, then it is not copied into `one_str`
  - You *must* add `one_str[n] = '\0';`, otherwise you will overrun the array.
- If *n* is longer than the second string, then the last character or null string is copied over and over until we get up to *n*.
  `strncpy(one_str, "Test", 8);`
  - This means `one_str` contains `"Test"`, but this is not eight characters to the null character will be copied three times into `one_str`,
  - Thus `one_str` contains:
    `"Test\0\0\0\0"`.

Substrings

- We frequently need to reference a substring of a longer character string.
  - We might want to examine the "Cl" in the string "NaCl".
  - `strncpy` can be used to extract the first `n` characters of a string.
  - The working of `strncpy` will give us insight into how we could also use this function to copy a middle or an ending portion of a string.
    ```
    strncpy(one_str, &sl[2], 2);
    ```
  - This goes to the first spot in the array and gets the memory location, and then `strncpy` copies 2 elements starting at this new memory location[1].
  - To extract the end of this string we can do the follow:
    ```
    strcpy(one_str, &sl[2]);
    ```

[1] Note that we are copying only the "Cl" portion of the string. May need to manually add the null character.

Longer Strings: Concatenation and Whole-Line Input

- `strcat` and `strncat`:
  - Concatenation is taking two strings and then puts them into one string.
- `strcat` takes two strings as arguments.
  - The second string is appended to the end of the first string.
  - Thus if `one_var = "Cory "` and `two_var = "Strope"`, then `strcat(one_var, two_var);` would make `one_var = "Cory Strope"`
  - Make sure that `one_var` has enough space to accomodate both strings, or else this command will overflow the boundaries, and overwrite other data.
- `strncat` does the same as `strcat`, but it only copies `n` characters from the second string,
  - `strncat(one_var,two_var,3);` would make `one_var = "Cory Str"`.
  - Null character might not be added, so you may have to do this manually Remember to use `strlen` to make sure you know the size of the string and you don't have overflow.

Longer Strings

The two most important questions dealing with strings are the following:
(1) Is there enough room to perform the given operation?
(2) Does the created string end in '\0'?

Distinction Between Characters and Strings

- When using `strcat`, one may be tempted to supply a single character as one of the two strings.
- A type `char` value is not a valid argument for a function with a parameter of type `char *`.
- In order to concatenate a single character, you must use `"a"` and not `'a'`.

Scanning a Full Line

- The `scanf` only gets non-whitespace characters
- Sometimes it is necessary to get the whitespace characters. Thus in the `stdio` library there is a function `gets` and `fgets`.

```
char read_line[80];
gets(read_line);
```

- The first line declares a string of size 80, and the second will get 80 characters from the user.
  - If the user enters more than 80 characters we will get an overflow.
- `fgets` is similar, but it takes three arguments.

```
char read_line[80];
fgets(file_name,read_line,size);
```

String Comparison

- In earlier chapters we learned learned relational and equality operators such as: $=$, $!=$, $<$, $>$, $<=$, and $>=$.
- We can not use these for strings, since they are an array of characters.
  - If we try to refer to a string without the subscript we are actually referring to the memory location, so `string_1` $<$ `string_2`, would compare the memory locations.

String Comparison

- Because we can not use $=, !=, <, >, <=, >=$. C provides us a function called `strcmp` and `strncmp`.
- `strcmp(string_1,string_2);` compares `string_1` to `string_2`.
  - It gives a negative number if `string_1` is less than `string_2`,
  - It gives a zero if they are equal, and
  - It gives a positive number if `string_1` is larger than `string_2`.
- `strncmp(string_1,string_2,n);` compares the first `n` characters of `string_1` and `string_2`.
  - It returns the same results as `strcmp`.

Comparison and Swapping

We can perform a sorting algorithm to a list of strings:

```
for(i=0; i<num_string; i++){
    for(j=i; i<num_string; j++)
        if (strcmp(list[i], list[j]) < 0)
            Swap(list[i],list[j]);
}
```

What would `Swap` look like?

Comparison and Swapping

Swapping two strings:

```
strcpy(tmp, list[index_of_min]);
strcpy(list[index_of_min], list[fill]);
strcpy(list[fill], tmp);
```

Arrays of String Constants

We can use the idea of 2-dimensional arrays to store a list of names.
char names[98][20]; The first represents how many names we will have, the second tells us how long each name can be.
strcpy(names[0], "Cory Strope"); This would put the string "Cory Strope" in the first spot of names.

Character Operations

- We can put `#include <ctype.h>` at the top of our program to access more functions that operate on characters.
- `scanf("%c", &ch);` vs. `ch = getchar();`
  - Both get a character;
  - `getchar();` returns an integer; the `EOF` has no character associated with it.
- `fscanf(inp, "%c", &ch);` vs. `ch = getc(inp);`
  - Get characters to file. Similar to `scanf` and `getchar`.
- `printf("%c", ch);` vs. `putchar(ch);`
  - Outputs a character to the screen.
- `fprintf(outp, "%c", ch);` vs. `putc(ch, outp);`

Character Analysis and Conversion

- `isalpha(ch);` checks to see if the variable `ch` is a letter of the alphabet
- `isdigit(ch);` checks to see if the variable `ch` is holding a digit
  - returns 0 for no
  - returns a positive integer for yes
- `islower(ch);` checks to see if `ch` is holding a lowercase character (`toupper`)
- `isupper(ch);` other way 'round. (`tolower`)
- `ispunct(ch);` `ch` holding a punctuation?
- `isspace(ch);` `ch` holding a space?

String-to-Number and Number-to-String Conversions

- `sprintf` takes numbers, doubles, characters, and strings and concatenates them into one large string.
- `sprintf(string_1, "%d integer %c - %s", int_val, char_val, string_2);`
  -
  - If `int_val = 42`, `char_val = 'a'`, and `string_2 = "Cory Strope"`
  - then `string_1` would be `"42 integer a - Cory Strope"`.
- `sscanf` takes a string and parses it into integer, doubles, characters, and strings.
  - `sscanf(" 42 3.141592 Cory Strope", "%d %lf %s %s", &num, &pi, &string_1, &string_2);`
  - `num = 42`, `pi = 3.141592`, `string_1 = "Cory"`, and `string_2 = "Strope"`.

Common Programming Errors

- We usually use functions to compute some value and use the return to send that value back to the main function. However functions are not allowed to return strings, so we must use what we learned about input/output parameters.
- Know when to use & and when not to use them, don't use them for whole arrays, but use them for `int`, `char`, and `double`.
- Be careful not to overflow strings.
- Most importantly make sure that there is always a `'\0'` at the end of your strings.