1. STRINGS AND POINTERS

1.1 Introduction

A string is a sequence of characters (array), with the last character being the `NUL` character (`\0`).

> For example, the array of 6 characters: `I n d i a \0` is a string of length 5 (the `NUL` character is not counted as part of the length). Had the same array been without the terminating \0, it would not be called a string, but simply a character array.

Because of this extra \0 character (whose ASCII value is 0) that signals the end of strings, they can be treated in special ways that normal arrays cannot be treated.

1.2 String variables, string constants and declarations

`char word[40];` declares an array of forty characters and can be treated as a string if a \0 is one of the elements.

Consider these three array declarations:

```
char arr[] = "Hello";
char brr[7] = {'H', 'e', 'l', 'l', 'o'};
char crr[] = {'H', 'e', 'l', 'l', 'o'};
```

The entity `arr` is a string, because the variable `arr` is being initialized with the string constant "`Hello`" at the time of declaration. (This is similar to saying `int i = 4;` insofar initializing an integer variable with an integer constant.)

Similarly, `brr` is also a string, for if even one element of an array is initialized (at the time of declaring the array), the remainder of the uninitialized elements are automatically set to 0.

The third array, `crr` is not a string, because there is no \0 assignment in the declaration, and only 5 bytes are set aside in memory. Hence, crr is just a character array.

The following three entities have different meanings: `4`, `'4'`, `"4"`
4 is an integer constant, '4' is a character constant (with an ASCII value of 52), while "4" is a string constant.

1.3 Getting input into string variables and to printing out strings

The conversion specifier "%s" is used by both `scanf()` and `printf()` functions to get string input, and give string output, respectively. Consider this piece of code:

```
char word1[100];
printf("Enter the word: ");
scanf("%s", word1);
printf("The word you entered is: %s\n", word1);
```

Note that `scanf()` expects an argument that is an address (pointer). Since the name of the array is its base address, you don't need to have its name preceded with the address of operator (&).

The %s specifier in the `scanf()` function will take input until the first whitespace character, [i.e., a space, \n (newline) or \t (tab) character] is encountered. So, how to take a line of input?

```
scanf("%[^\n]", word1);
```

By specifying the character class as the format specifier, where you state that all characters till \n (the caret symbol denotes negation of the character class) should be accepted, you can now take a line of input.

The character class can be used in `scanf()` in other usual ways such as `%[A-Za-z0-9]` to accept a string of only alphanumeric characters, `%[aeiouAEIOU,]` to accept a string of vowels and a comma, and so on. In such cases, the `scanf()` stops accepting input at the first instance of the condition being violated. While no further input is taken into the string, the remaining input continues to stay in the input buffer! This may cause problems with subsequent calls to an input function like `scanf()` and `getchar()`. It is important to keep this in mind when writing code.

For instance, when scanning a line of text using the statement `scanf("%[^\n]", word1);` the \n character continues to stay in the input buffer, causing an empty string to be stored in the array when the next `scanf("%[^\n]", word1);` statement in encountered. One way to remedy this issue is by including an input statement to eat away the \n:

```
scanf("%[^\n]", word1);
getchar();   /* an empty getchar() to dissipate the \n */
```

Exercise 8-1: Write a program that accepts a line of text as a string from the user and prints out only the non-whitespace characters of it. You may assume a line contains at most 99 characters.

Next, modify the program so it works for N lines of text, where N is a number between 1 and 10 input by the user. Omit the empty `getchar()` statement after the `scanf()` and observe what happens. You should understand how to handle the input buffer stream properly, otherwise your program may show unexpected behaviour.

1.4 String Functions

The essence of treating a character array as a string is best put to use by string functions (available in the standard C library). Use of these functions requires the programmer to include the <string.h> header file in the program. The prototypes and use of some of the commonly used string library functions are given below:

```
size_t strlen(const char *s);    // size_t is long unsigned int on gcc
```

Returns the length (number of bytes, excluding \0) of the string **s**.

```
char *strcpy(char *dest, const char *src);
```

Copies the string **src** to **dest**, returning a pointer to the start of **dest**.

```
int strcmp(const char *s1, const char *s2);
```

Compares the strings **s1** with **s2**. Returns 0 if **s1** is identical to **s2**, a negative value if **s1** < **s2**, and a positive value if **s1** > **s2**.

```
char *strcat(char *dest, const char *src);
```

Appends the string **src** to the string **dest**, returning a pointer to **dest**.

```
char *strchr(const char *s, int c);
```

Returns a pointer to the first occurrence of the character **c** in the string **s**.

```
char *strstr(const char *haystack, const char *needle);
```

Finds the first occurrence of the substring **needle** in the string **haystack**, and returns a pointer to the found substring, or NULL if the substring is not found.

Exercise 8-2: Type `man string` on the shell prompt to see the manual page entry of the collection of string functions available. Look up the individual manual page entries of these four functions specifically: `strlen()`, `strcmp()`, `strcpy()`, `strcat()`. Write

few lines of code to understand how these functions work. For instance, you may try these as part of your program:

```
char s1[50] = "BITS", s2[] = "Pilani", s3[50];
printf("%lu %lu\n", strlen(s1), sizeof(s1));
    /* It is easy to get confused between strlen() and sizeof() */
strcat(s1," ");
strcat(s1, s2);
strcat(s1, ", ");
strcpy(s3,"Pilani Campus");
strcat(s1, s3);
puts(s1);       /* Look up the man page of puts() to learn more */
/* comparing two strings… */
if (!strcmp(s2,s3)) /* strcmp() has returned 0 */
    puts("Both strings are the same!\n");
else
if (strcmp(s2,s3) > 1)
    printf("%s is greater than %s\n", s2, s3);
else
    printf("%s is greater than %s\n", s3, s2);
```

Exercise 8-3: In the mid-semester test you took last week, a pointer-based implementation of `strlen()` function was provided:

```
int my_str(char p[])
{
    char *q = p;
    while (*q)
      q++;
    return q-p;
}
```

Now, you implement the same function without using pointers. You may name this function as **`my_strlen1()`**.

Exercise 8-4: Implement a function **`convertLower()`** (whose prototype is given below) to convert a given string into one that contains entirely lower case letters.

```
void convertLower(char *);
```

The function performs the conversion on the string received as its argument. First implement your function using ASCII codes only. Next, use the library function `tolower()` to do the job. Remember to include the header `<ctype.h>`. You can learn more about this function by using the online manual page.

[*Note:* There are a suite of useful library functions like `tolower()` and `toupper()` available in the ctype library. Get to know them by typing man isalpha at the shell prompt. You should be familiar with the first 13 functions – starting from `isalnum()` and ending with `isblank()`, whose descriptions are also provided in the manual page, so you can use them in your programs when needed.]

Exercise 8-5: Write a function that checks to see if the string it takes as an argument is a palindrome or not.

(a) Write the first version of the function **`isPalind_1()`** using array notation, and the second version **`isPalind_2()`** using pointers.

(b) Now, modify one of the above two functions so that **`isPalind_3()`** that you write now uses a function **`strrev()`** whose definition is given below:

```
void strrev(char *a) /* function to reverse the given string */
{
  char *b, temp;
  b = a + strlen(a) - 1;  /* b is made to the last character preceding
                             the terminating \0 */
  for (  ; a < b ; a++, b--)
       {           /* swapping the contents pointed to by a and b */
         temp = *a;
         *a = *b;
         *b = temp;
       }
  return;
}
```

Check this space on Friday 12 noon for

Additional Practice Exercise Problems!