Strings in C

Basically, strings are character arrays in C. However, that isn't the complete picture. It would be nice if strings didn't always have to be the same length, as character arrays are. In order to deal with this issue, strings in C, by default, are null terminated. This means that the last character storing a string is the null character, '\0'.

For example, the following is a valid way to initialize a string to store "hello";

```
char word[20];

word[0] = 'h';   word[1] = 'e';
word[2] = 'l';   word[3] = 'l';
word[4] = 'o';   word[5] = '\0';
```

In this example, the character array word is storing a string of length 5. Notice that we actually store six characters here. This means that a character array of size 20 can actually store a string with a maximum length of 19, NOT 20, since the last character in the array must be reserved for the null character.

Another way to read in a string is from the keyboard directly into a character array:

```
scanf("%s", word);
```

This reads in all the characters typed in until whitespace is read in and automatically adds the null character to the end of what is read in. One consequence of this idea is that the literals 'a' and "a" are different. The first is a single character, the second is stored in a character array of at least size two, where the last character is the null character.

Example of Processing a String

The three following examples turn a string into its uppercase version, return the length of a string and reverse the contents of a string.

```
void to_upper(char *word) {

  int index = 0;

  while (word[index] != '\0') {
    word[index] = toupper(word[index]);
    index++;
  }
}

int length(char *word) {

  int index=0;

  while (word[index] != '\0')
    index++;
  return index;
}

void reverse(char *word) {
  int index, len;
  char temp;
  len = length(word);
  for (index=0; index<len/2; index++) {
    temp = word[index];
    word[index] = word[len-1-index];
    word[len-1-index] = temp;
  }
}
```

Standard Libary (string.h) functions

There are four C string functions that are used quite commonly and covered in the text:

```
// This function concatenates the string s2 to the string s1 and
// stores the result in s1. The const in front of s2 indicates that
// the function will not change the contents of the string s2.
char* strcat(char *s1, const char *s2);

// This function compares the two strings s1 and s2. If s1 comes
// first alphabetically, an integer less than 0 is returned. If the
// two strings are equal 0 is returned. If s2 comes before s1
// instead, an integer greater than 0 is returned.
int strcmp(const char *s1, const char *s2);

// This contents of s2 are copied into s1. This works
// DIFFERENTLY than s1 = s2, which would just copy
// a pointer.
int strcpy(char *s1, const char *s2);

// Returns the number of characters in s before a '\0' is
// encountered.
int strlen(const char *s);
```

These functions come in handy with dealing with strings. It's important to follow the specification for each function when calling them. As mentioned above, the const guarantees that the contents of that string won't be changed by the function.

Finding the First String alphabetically

We know that the student with the first last name alphabetically always gets to go first! Here's a program to determine that student:

```
int main() {

  char curname[30], firstname[30];
  int numstuds, i;

  // Get number of names.
  printf("How many students are there?\n");
  scanf("%d", &numstuds);
  printf("Enter their last names.\n");

  for (i=0; i<numstuds; i++) {
    scanf("%s", curname);

    // Update the first name seen.
    if (i==0)
      strcpy(firstname, curname);

    // Update if we find a new first name.
    else if (strcmp(curname, firstname) < 0)
      strcpy(firstname, curname);
  }

  printf("The first person in line is ");
  printf("%s.\n", firstname);

  return 0;
}
```

More on String Functions

The key to using these String functions, (and any prewritten functions), is to understand exactly what the given functions do, so that you can call them with the appropriate parameters, in the appropriate manner.

In the program above, we wanted to "store" the name of the earliest name alphabetically we had seen. However, the statement

```
firstname = curname;
```

would have been inadequate, because it just changes a pointer. (That's essentially what a string is.) Instead the function strcpy, does the trick. In using it, we must make sure we pass in the proper parameters:

```
strcpy(firstname, curname);
```

This copies the value of the string curname into the string firstname, which is exactly what we want.

Furthermore, strcmp works in an even less obvious way than strcpy did. It returns an integer from comparing two strings. Basically, it's similar to checking for <, ==, and >, all at once. If the first is the correct relationship between the two strings, a negative integer is returned, if they are equal, 0 is returned, and otherwise, a positive integer is returned. Since we are checking to see if curname *comes before* firstname, the appropriate boolean condition is:

```
strcmp(curname, firstname) < 0
```

Palindrome Example

The following function returns true if and only if the input parameter is a palindrome. This determination is a case sensitive one:

```
int palindrome(char *word) {

  int len, index;
  index = 0;
  len = strlen(word);

  while (word[index] == word[len-1-index] &&
         index < len/2)
    index++;

  if (index == len/2)
    return 1;

  return 0;
}
```