http://shop.stringking.net/images/frontend/theme/stringking/pure_corde_info/pure_corde_stringking1.jpg

Character arrays are very useful…

Useful String Operations

- strcmp
- strcpy
- strlen
- strcat

Strings

- Strings are **arrays** of type `char` so they can be accessed using square brackets `[ ]`.

```
char str[10];
strcpy ( str, “abcde” );
if ( str[3] == ‘d’ ) {
   printf ( “str[3] is the character d\n”);
}
```

- **Reminder**: Use **double** quotes for strings and **single** quotes for single `character` comparisons.

String Operations

- Compare two strings (s1, s2) for equality
  - `strcmp ( s1, s2 )`

Returns:
> 0 if `s1` > `s2`
0 if `s1` = `s2`
< 0 if `s1` < `s2`

- Copy one string into another
  - `strcpy ( s1, s2 )`

Returns pointer to `s1`

- Find the length of a string
  - `strlen ( s1 )`

Returns length of string **not** including \0

Note: if you use strlen to determine the size of a `malloc()` to store the string you must add one to the length to store the `\0` (end of string/line) character.

The Issue with `strlen()`

```
char str[100];

char *ptr;
```

… read something into str …

```
ptr = malloc ( sizeof(char) * (strlen(str) + 1) );
```

length of the contents of `str` plus one for the end of line marker or string terminator (NULL)

String Operations

- Concatenate two strings (copy one onto the end of the other)
  - `strcat ( s1, s2 )` **Returns** the pointer to `s1`

    Note: `s1` must be long enough to hold both strings (`s1`+`s2`) and an end-of-line character.

- Find a character `c` in a string `s`
  - `index ( s1, c )`

    Returns a pointer to the located character or `NULL` if it does not appear in the string

Character Operations

- Comparing a character c and an element in a string s1

```
if ( s1[i] == 'c' ) …
```

Single quotes

- Changing a single character in an array

```
s1[i] = 'c';
```

Note: You do not need the **single** quotes if the character is a variable.

```
char letter;
char str[10];
letter = 'm';
str[3] = letter;
```

Reading a Single Character

- The `getc` function will read one character from a specified stream.
- The character read is the return value of `getc`.

```
#include <stdio.h>
int main ( )
{
   char c;
   int i;
   printf ( “Enter 3 characters\n” );
   for ( i=0; i<3; i++ ) {
      c = getc ( stdin );
      printf ( “%c\n”, c );
   }
}
```

Reading a Single Character

```
$ ./inputGetc
Enter 3 characters
abc
a
b
c
```

This is what you would expect the output to look like and it does if you type in the characters correctly.

```
./inputGetc
Enter 3 characters
1
1


2
2
```

But what if you try to press return after each character. Not what you might expect! Why??

Reading a Single Character

```
./inputGetc
Enter 3 characters
1
1


2
2
```

The program has done the following:

- *It reads the first character (1) and prints it out.*
- *Then it reads the* `<enter>` *key as the next character and prints it out.*
- *Then it reads the 2 as the 3rd character, prints it out and terminates - it has read in 3 characters:*

1 `<enter>` *2*

Using `fgets()`

- `fgets` can be used to read in an entire string (array of characters).
- It is an alternative to `fscanf` and should you should not mix `fgets` and `fscanf` in a program. They treat the end-of-line differently.
  - `fgets` reads everything typed on a line up until the `<enter>` key is pressed **or** until the maximum string length is reached.

fgets()

```
char str[10];

fgets ( str, 10, stdin );

printf ( "%s\n", str );
```

The arguments for `fgets()` are:

- *string to store values in (pointer)*
- *maximum length of string*
- *stream to read from (can be a file pointer)*

```
$ ./inputFgets
12 34 5678910
12 34 567
```

9 characters and the end-of-line character are used to fill the 10 character array `str`

The input string was too long for `fgets` to read so the last characters were not stored in the array. The extra characters are left waiting to be read and the next read will get them.

```
#include <stdio.h>

int main ( )
{

   char str[10];
   fgets ( str, 10, stdin );
   printf ( "%s\n", str );

   fgets ( str, 10, stdin );
   printf ( "%s\n", str );

}
```

```
$ ./input2Fgets
12 34 5678910
12 34 567
8910

$
```

Manipulating Strings

- `sprintf` and `sscanf` work exactly like `printf` and `scanf` but use a string instead of `stdin` and `stdout`.
- Both use a string as their first argument followed by the format statement and variables.

Manipulating Strings

- **`sprintf`**
  - prints results to a string
  - very useful for converting primitive types to a string (*i.e.* creating a string)
- **`sscanf`**
  - reads from a string
  - very useful for converting strings to primitive types (*e.g.* converting a string to a number)

Manipulating Strings

Use `sscanf` to convert a string to floats

- Read in a string containing 3 floating point numbers
- Convert them from strings to floats using `sscanf`

```
char inputStr[50];
float var1, var2, var3;
var1 = var2 = var3 = 0.0;

printf ( “Enter 3 floating point numbers\n” );

fgets ( inputStr, 50, stdin );

sscanf ( inputStr, “%f %f %f”, &var1, &var2, &var3 );

printf ( “Total: %f + %f + %f = %f\n”, var1, var2, var3,
var1+var2+var3 );
```

```
$ ./sscanfConvert
Enter 3 floating point numbers
2.3 2.3 3.4
Total: 2.300000 +  2.300000 +  3.400000 = 8.000000
```

```
$ ./sscanfConvert
Enter 3 floating point numbers
a.1 b.2 c.3
Total: 0.000000 +  0.000000 +  0.000000 = 0.000000
```

Manipulating Strings

Use sprintf to convert primitive types to a string.

```
int count;
float cost;
char str[100];

count = 10;
cost = 1.99;

sprintf ( str, “The total cost for %d items is
%f”, count, cost*count );

printf ( “%s\n”, str );
```

```
$ ./sprintfConvert
The total cost for 10 items is 19.900000
```