**Strings in data structures**

* Nowadays, computers are widely used for word processing applications such as creating, inserting, updating, and modifying textual data. Besides this, we need to search for a particular pattern within a text, delete it, or replace it with another pattern. So, there is a lot that we as users do to manipulate the textual data.
* Strings in C are stored as null character, '', terminated character arrays.
* This means that the length of a string is the number of characters it contains one more to store the null character.
* Common string operations include finding lengths, copying, searching, replacing and counting the occurrences of specific characters and worlds.

**Library Functions for String Manipulation**

1. **strcpy :**strcpy copies a string, including the null character terminator from the source string to the destination. This function returns a pointer to the destination string. Its prototype is :

char \*strcpy(char \*dst, const char \*src) ;

1. **strncpy :**strncpy is similar to strcpy, but it allows the number of characters to be copied to be specified.

char \*strncpy(char \*dst, const char \*src, size\_t len) ;

1. **strcat :** This function appends a source string to the end of a destination string. This function returns a pointer to the destination string, or NULL pointer on error. Its prototype is :

char \*strcat(char \*dst, const char \*src) ;

1. **strncat :** This function appends at most N characters from the source string to the end of the destination string. This function returns a pointer to the destination string, or a NULL pointer on error. Its prototype is :

char \*strncat(char \*dst, const char \*src, size\_t N) ;

1. **strcmp :** Two strings are compred with this function. If the first string is greater than the second, it returns a number greater than zero. If the second string is greater, it returns a number less than zero. If the strings are equal, it returns 0. Its prototype is :

int strcmp(const char \*first, const char \*second) ;

1. **strncmp :** This function compares the first N characters of each string. If the first string is greater than the second, it returns a number greater than zero. If the second string is greater, it returns a number less than zero. If the strings are equal, it returns 0. Its prototype is :

int strncmp(const char \*first, const char \*second, size\_T N) ;

1. **strlen :** This function returns the length of a string, not counting the null character at the end. That is, it returns the character count of the string, without the terminator. Its prototype is :

size\_t strlen(const char \*str) ;

1. **memset :** memset is useful to initialize a string to all nulls, or to any character.

void \*memset(const void \*dst, int c, site\_t N) ;

**Arrays of Strings**

An array of strings is just a two dimensional array of characters. Consider this :

char names[7][6] ={ "Abc", "Tom", "Chad", "Hello", "Xyz", "SSD", "TTT"} ;

**Passing Strings to Functions**

* It is often best to treat an array as an abstract data type with a set of allowed operations on the array which are performed by funtional modules.
* Let us return to our exam score example to read and store scores in array and then print them, except that we now wish to use functions to read and print the array.
* Read an array and print a list of scores using functional modules.

n = read\_intarray(exam\_scores, MAX) ;

print\_intarray(exam\_scores, n) ;

The algorithm is very similar to our previous task, except that the details of reading and  printing the array is hidden by functions. The function, read\_intarray( ), reads scores and stores them, returning the number of scores read. The function, print\_intarray( ), prints the contents of the array. The refined algorithm for main( ) can be written as : print title, etc.