Character strings , String Functions

- You already know that a character string is a char array terminated with a null character (\0)
  - ➔ Character strings are commonly used.
  - ➔ C provides many functions specifically designed to work with strings.
- Some of the more commonly performed operations on character strings include.
  - ➔ Getting the length of a string
    - ★ Strlen
- Copying one character string to another
  - ➔ strcpy() and strncpy()
- Combining two character strings together (concatenation)
  - ➔ strcat() and strncat()
- Determining if two character strings are equal
  - ➔ strcmp() and strncmp()

The C Library supplies these string-handling function prototypes in the string.h header file

Getting the length of a string

- The strlen() function finds the length of a string
  - ➔ Returned as a size_t

```
#include <stdio.h>
#include <string.h>

int main()
{
    char myString[] = "my string";

    printf("The length of this string is : %lu\n", strlen(myString));

    return 0;
}
```

- This function does change the string
  - ➔ The function header does not use const in declaring the formal parameter string.

Copying Strings

- Since you can not assign arrays in C , you can not assign strings either

```
char s[100]; //declare
S = “hello”; // initialize - DOESN’T WORK(lvalue required’ error)
```

- You can use the strcpy() function to copy a string to an existing string
  - ➔ The string equivalent of the assignment operator

```
Char src[50], dest[50];

strcpy(src, “This is source”);
strcpy(dest, “This is destination”);
```

- The strcpy() function does not check to see whether the source string actually fits in the target string.
  - ➔ Safer way to copy strings is to use **strncpy()**

- strncpy() takes a third argument
  - ➔ The maximum number of characters to copy

char src[40];
char dest[12];

memset(dest, ‘\0’, sizeof(dest));
strcpy(scr, “Hello how are you doing”);
strncpy(dest, src, 10);

String concatenation

- The strcat() function takes two strings for arguments
  - ➔ A copy of the second string is tackled onto the end of the first
  - ➔ This combined version becomes the new first string
  - ➔ The second string is not altered

- It returns the value of its first argument
  - ➔ The address of the first character of the string to which the second string is appended

```
int main()
{
    char myString[] = "My string";
    char input[80];

    printf("Enter a string to be concatenated: ");
    scanf("%s", input);

    printf("\nThe string %s concatenated with %s is ::::\n", 
myString, input);
    printf("\n %s", strcat(input , myString ));
    return (0);
}
```

- The strcat() function does not check to see whether the second string will fit in the first array /
    - ➔ If you fail to allocate enough space for the first array, you will number of characters overflow into adjacent memory locations
- Use strncat() instead
    - ➔ Takes a second argument indicating the maximum number of characters to add
- For example , strncat(bugs, addon, 13) will add the contents of the addon strings to bugs, stopping when it reaches 13 additional characters or null character , which ever comes first.

```
#include <stdio.h>
#include <string.h>

int main()
{
    char src [50], dest [50];

    strcpy(src, "This is source");
    strcpy(dest, "This is destination");

    strncat (dest, src, 15);
    printf("Final destination string: |%s|", dest);
    return (0);
}
```

Comparing Strings

- Suppose you want to compare someone's response to a stored string .
    - ➔ Cannot use == , will only check to see if the sting has the same address.

- There is a function that compares string contents , not string addresses.
    - ➔ It is the strcmp() (for string comparison) function
    - ➔ Does not compare arrays , so it can be used to compare strings stored in arrays of different sizes .
    - ➔ Does not compare characters
        - ★ You can use arguments such as “ apples” and “A”, but you cannot use character arguments, such as ‘A’
- This function does for strings what relational operators do for numbers.
    - ➔ It returns 0 if the two string arguments are the same and nonzero otherwise.
    - ➔ If returns value < 0 then it indicates str1 is less than str2
    - ➔ If returns value > 0 then it indicates str2 is less than str1

```
#include <stdio.h>
#include <string.h>

int main()
{
    printf("strcmp(\"A\",\"A\") is ");
    printf("%d\n", strcmp("A", "A"));

    printf("strcmp(\"A\",\"B\") is ");
    printf("%d\n", strcmp("A", "B"));

    printf("strcmp(\"B\",\"A\") is ");
    printf("%d\n", strcmp("B", "A"));

    printf("strcmp(\"C\",\"A\") is ");
    printf("%d\n", strcmp("C", "A"));

    printf("strcmp(\"Z\",\"a\") is ");
    printf("%d\n", strcmp("Z", "a"));

    printf("strcmp(\"apples\",\"apple\") is ");
    printf("%d\n", strcmp("apples", "apple"));
}
```

Comparing Strings (cont'd)

- The strcmp() function compares strings until it finds corresponding characters that differ .
  - ➔ Could take the search to the end of one of the strings.
- The strncmp() function compares the strings until it has compared a number of characters specified by the third argument .
  - ➔ If you wanted to search for that begin with “astro”, you could limit the search to the first five characters
  - ➔

    ```
    #include <stdio.h>
    #include <string.h>

    int main ()
    {
        if (strncmp("astronomy","astro",5) == 0)
        {
            printf("Found: Astronomy");
        }

        if (strncmp("astounding","astro",5) == 0)
    ```

```
    {
        printf("Found: astounding");
    }
}
```