Introduction to C Programming

Section 8

Introduction

- Until now
  - We have seen strings in printf
  - Our old definition: string is a set of char between “”

```
printf("This is a string\n");

printf("This is %s\n", "a string\n");
```

- Strings:
  - An array of chars
  - Terminated by the null char `'\0'`

Strings in C

- Since strings are array

```
char str1[] = {'p','r','o','g','r','a','m', '\0'};
char str2[8] = "program";
char str3[] = "program";
```

'p'	'r'	'o'	'g'	'r'	'a'	'm'	'\0'

String

- Initializing char array ...
  - char s[10] = "unix"; /* s[4] is '\0'; */
  - char s[ ] = "unix"; /* s has *five elements* */

Strings are Character Arrays

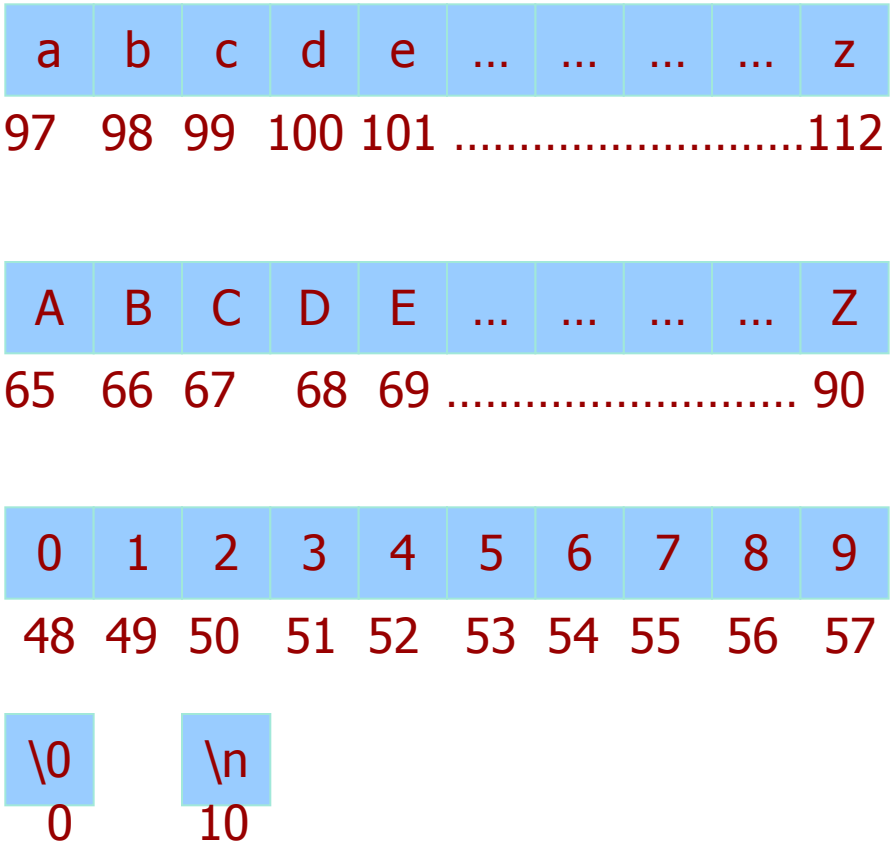
- Strings in C are simply arrays of characters
  - Example: `char s [10];`
- This is a ten (10) element array that can hold a character string consisting of ≤ 9 characters
- This is because C does not know where the end of an array is at run time
  - By convention, C uses a NULL character `'\0'` to terminate all strings in its library functions
- For example:

```
char str [10] = {'u', 'n', 'l', 'x', '\0'};
```

- It's the string terminator (not the size of the array) that determines the length of the string

Strings

- Each character has an integer representation



Accessing Individual Characters

- The first element of any array in C is at index 0. The second is at index 1, and so on ...

```
char s[10];
s[0] = 'h';
s[1] = 'i';
s[2] = '!';
s[3] = '\0';
```

?	?	?	?	?	?	0\	!	i	h
[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

s

- This notation can be used in all kinds of statements and expressions in C:
  - For example:

```
c = s[1];
if (s[0] == '-') ...
switch (s[1]) ...
```

String Library

- Access to string library by
  - `#include <string.h>`

- Many functions to work with strings
  - Find the length of string
  - Compare strings
  - Copy strings
  - Search in strings
- Concatenating strings

Length of String

- **`strlen(str)`**: Length of string

- From start to first occurrence of the null char

```
char str[] = "This is test";

char str1[10] = {'a', 'b', '\0', 'c', '\0'};

strlen(str)   >>>    12

strlen(str1) >>>    2
```

Compare Strings

- str1 and str2 are compared as follows
  - Compare char by char from left to right until str1 and str2 has same chars.
  - In the first different char
  - If(char of str1 < char of str2) > str1 < str2
  - If (both string finish) > str1 = str2
- **`strcmp(str1, str2)`**: compare str1 and str2
  - If(str1 == str2) > return 0
  - If(str1 < str2) > return -1
  - If(str1 > str2) > return 1

Compare Strings

- **`strcmpi(str1, str2)`**
- Compares str1 and str2 similar to strcmp
- But ignores uppercase/lowercase difference

```
char str1[]="ABC", str2[]="abC";

strcmpi(str1, str2)  0
```

strcpy

- Copying a string comes in the form:

  char *strcpy (char * destination, char * source);

- A copy of source is made at destination
  - source should be NULL terminated
  - destination should have enough room (its length should be at least the size of source)

Copy Strings: Example

```
char str1[] = "Test String";

char str2[20];

strcpy(str2, str1);

printf("%s\n", str2);       // test string

printf("%s\n", str1);       // test string
```

strcat

- Concatenating two strings:

  char * strcat (char * str1, char * str2);

  - Appends a copy of str2 to the end of str1

- Ensure that str1 has sufficient space for the concatenated string!
  - Array index out of range will be the most popular bug in your C programming career

Concatenate Strings: Example

```
char str1[]   = "String";
char str2[20]= "Test ";


 strcat(str2, str1);
 printf("%s\n", str2);          // test string
```