UCSC Silicon Valley Extension

Advanced C Programming

String and Character Functions

Instructor: Radhika Grover

More string functions

- char *strchr (const char   *s , int *c*) returns a pointer to the first occurrence of character *c* in the string *s*.

- char * strtok(const char *s1, const char *delim) breaks s1 into tokens by finding groups of characters separated by any of the delimiter characters in delim.
  - Return values are pointers to substrings of s1 rather than copies.

More string functions

- char *strstr(const char *s, const char *t) : determines if string *t* occurs inside string *s*

Returns a pointer to the start of string *t* within the string *s*. If *t* does not occur in *s*, a NULL pointer is returned.

- int strncmp(const char *s1, const char *s2, size_t n) :compares the first n characters of s1 and s2 alphabetically.

Returns:

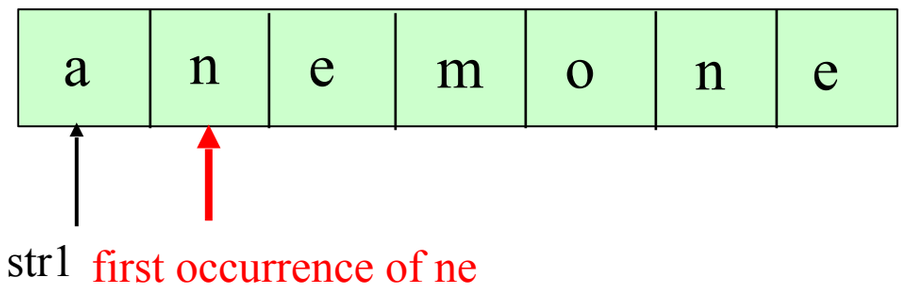
a negative value s1 precedes s2 in lexicographic order

a zero if strings are equal

a positive value if s2 precedes s1 in lexicographic order

Example strstr

char str1[] = “anemone”;
char str2[] = “ne”;
strstr(str1, str2) returns a pointer to the first occurrence of “ne” in “anemone”

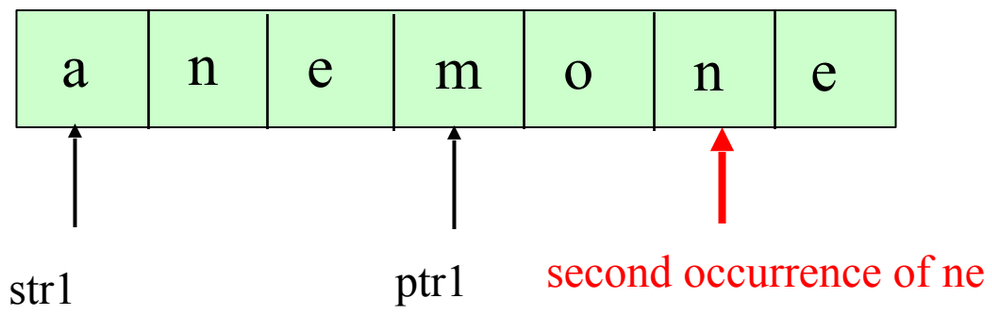


How can we find the next occurrence of “ne” in this string? Assume that overlap of occurrences is not allowed i.e. there is only one occurrence of axa in baxaxa.

Example strstr continued

- We can find the next occurrence of “ne” by searching in the portion of the string following the first occurrence of “ne”

  strstr(ptr1, str2) returns a pointer to the second occurrence of “ne” in “anemone”



Example strstr continued

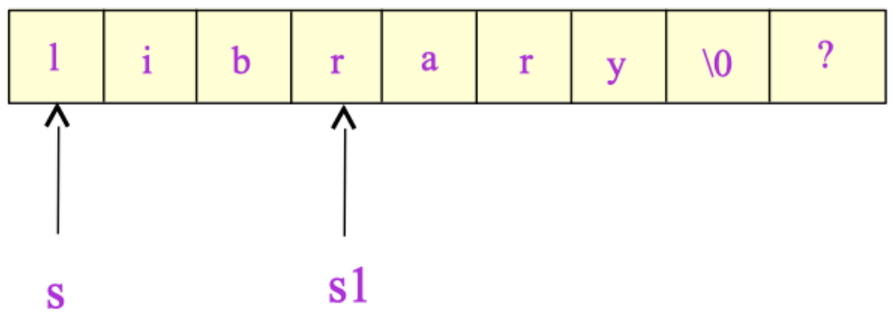
- This program counts and prints the number of times “an” appears within “planarians”.

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

int main(void)
{
    char str1[] = "planarians";
    char str2[] = "an";
    char *ptr1 = str1;          /* ptr1 points to the first element of str1 */
    int count  = 0;              /* Initialize count to 0 */

     /* Count the number of occurrences of str2 in the string that starts at pointer ptr1 . Every time a match is found
    increment count and move pointer ptr1 to the next portion of string. Repeat while ptr1 is not NULL */
    while( (ptr1 = strstr(ptr1, str2) ) != NULL ) {
            count++;
            ptr1 += strlen(str2) ;
     }
    printf("Count: %d \n", count);
    return 0;
}
```

strchr

- Example:

```
char   s[9] = "library";
char  *s1 = strchr (s, ‘r');
printf("Character 'r' found at position %d.\n", s1 - s);
```

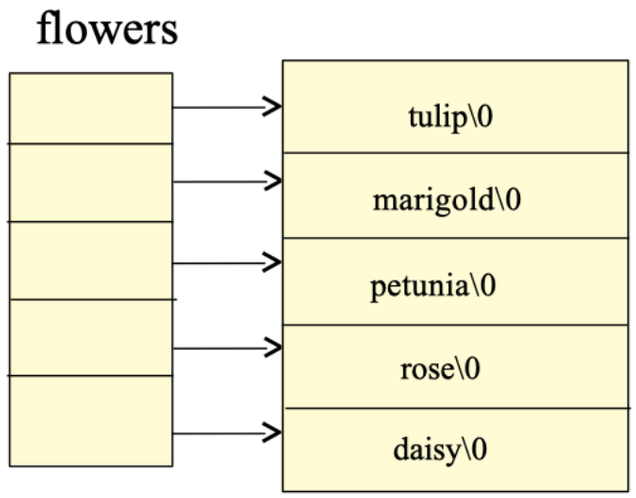


This prints out: Character ‘r’ found at position 3.

Arrays of pointers

- Create an array of 5 pointers called flowers and initialize it with strings as follows:

```
char  *flowers[5]  = {"tulip", "marigold", "petunia", "rose", "daisy"};
```



- Display the names of the strings in the array using this loop:

```
for(i = 0; i < 5; i++)
   printf("%s", flowers[i]);
```

- Note that this can also be written as a 2D array:

```
char  flowers[5][9] = {"tulip", "marigold", "petunia", "rose", "daisy"};
```

Character operations

- These functions are defined in library ctype.h.
- Classify characters as letters, punctuation digits etc.
- Each character is represented as an int.
- Return 0 if condition is false, otherwise a nonzero value

isalpha(ch)	Character is a letter of the alphabet
isdigit(ch)	Character is a decimal digit from 0 to 9
islower(ch)	Character is a lowercase letter of the alphabet
isupper(ch)	Character is an uppercase letter of the alphabet
ispunct(ch)	Character is a punctuation mark
isspace(ch)	Character is a space, newline, tab, form feed
isalnum(ch)	Character is alpha or digit
iscntrl(ch)	Character is new line, back space, tab, form feed
tolower(ch)	converts and returns the corresponding lowercase character
toupper(ch)	converts and returns the corresponding uppercase character

Character operations

- Examples:

```
char  c1 = 'a', c2 = '9', c3 = ' ', c4 = '\t';

printf ("%d\n", isalpha(c1));
printf ("%d\n", isdigit(c2));
printf ("%d\n", isalpha(c2));
printf ("%d\n", isspace(c3));
printf ("%d\n", isspace(c4));
printf ("%d\n", iscntrl(c4));
```

Formatting strings

- Function snprintf writes n –1 characters of the formatted arguments arg1, arg2, … to the buffer s of size n:

int snprintf(char *s, size_t n, const char *format, [, arg1, arg2, …]);

  - A null character is automatically added at the end of the output.
  - The size of the output is restricted to n –1 characters, and remaining characters are discarded.

- Function sprintf writes the formatted arguments arg1, arg2, … to string s:

int sprintf(char *s, const char * format, [, arg1, arg2,…]);

  - Returns an integer value that is the number of characters that were written.
  - Unsecure function that can lead to buffer overflow - use snprintf instead.

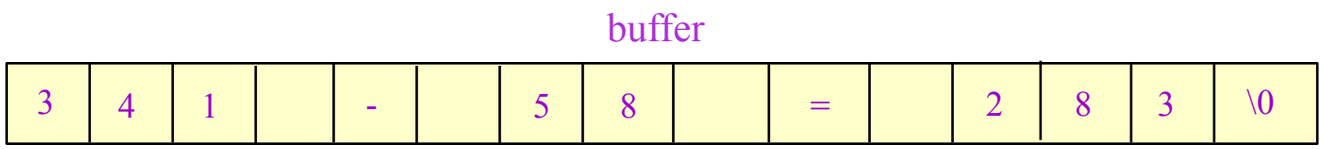
Example: snprintf and sprintf

Examples:

```
char buffer[15];
int i, a = 341, b = 58;
i = snprintf(buffer, sizeof(buffer), “%d  -  %d  =  %d", a, b, a-b);
```

The following usage is okay, but do not use sprintf when input string is provided by user as it could lead to a buffer overflow:

```
i = sprintf(buffer, "%d  -  %d  =  %d", a, b, a-b);
```



String to number conversions

- int atoi (const char *str) converts the string to an int representation.
- long atol (const char *str) converts the string to a long int representation.
- long long atoll (const char *str) converts the string to a long long int representation.