**Strings**

Strings are arrays of char with a sentinel marking the end. We play with arrays either using indices or using pointers. So, arrays, strings and pointers go together.

Let us recollect that pointers and integers are basically different types. Only arithmetic allowed on pointers are the following.

// add an int to a pointer : pointer

// subtract an int from a pointer : pointer

// diff of two pointers : int

We can write the function to find the length of a string using pointer arithmetic in this way. Check the file 1\_ex.c.

int mystrlen(char \*y)

{

char \*z = y;

while(\*z)

{

++z;

}

return z - y;

}

Make z point to the first character of the string pointed to by y.. Advance z until z points to the NULL character. Now, z – y is pointer arithmetic resulting in the number of components between the two pointer values. This is same as the length of the string.

There are a few functions which operate on strings. These are declared in header file string.h. We can also use man pages to find details of usage on a linux system. For ex, we can say

$ man strlen

STRLEN(3) Linux Programmer's Manual STRLEN(3)

NAME

strlen - calculate the length of a string

SYNOPSIS

#include <string.h>

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

DESCRIPTION

The strlen() function calculates the length of the string s,

excluding the terminating null byte ('\0').

RETURN VALUE

The strlen() function returns the number of bytes in the

string s.

**string concatenation:**

strcat function takes two arguments dst and src and appends src to dst.

char x[10] = "pesu"; // has 5 char

char y[5] = "univ"; // has 5 char

strcat(x, y); // x will not have 9 char; think why not 10?

printf("%s %s\n", x, y);// pesuuniv univ

If the destination does not have enough memory, the behaviour is undefined.

strcat(x, " India");

The program will crash if you are lucky!

**Search for a char in a string:**

The character we are searching for may or may not be found in the string.

We may decide to return the position if the element is found (0 to strlen of the str – 1) or -1 if the element is not found.

We will also try to find the leftmost occurrence.

Check the files : 2\_ex\_str.c 2\_mystr.h 2\_mystr.c.

The function signature :

// find the pos of ch in src;

// return the pos if found

// -1 if not found

int find\_left(const char \*src, char ch);

Let us look at the implementation.

int find\_left(const char \*src, char ch)

{

int pos = 0;

while(src[pos] != '\0' && src[pos] != ch)

++pos;

return src[pos] ? pos : -1;

}

The parameter src is supposed to point to a string which we do not plan to change. We can always treat a variable as a constant. It allows the compiler to optimize the code.

We start from the left end of the string – pos is 0 – and compare with the character ch until we find the character in the array or we come to the end of the string.

If the character is found, return pos else -1.

This is part of the client code.

char x[] = "kitkat";

// find the leftmost occurrence of say t

char ch;

ch = getchar();

int pos = find\_left(x, ch);

if(pos != -1)

{

printf("%c found at pos %d\n", ch, pos);

}

else

{

printf("%c not found in %s\n", ch, x);

}

What if we want to find all the occurrences of the given character. Once we find the character, we have to search from the next position. We should keep repeating this until the end of the string is encountered.

int pos = -1;

int i;

while( (i = find\_left(x + pos + 1, ch)) != -1)

{

pos = pos + i + 1;

printf("%d ", pos);

}

printf("\n");

The first time we call find\_left, the string starts at x – offset is 0

If the string is found at pos i, the next search from position pos + i.

Please go through the code to understand the logic.

We should also be able to appreciate multifile development. Should the developer of find\_left know how we are using the function in the client code?

Should the client know how the function is implemented in the implementation file?

**String matching:**

This is one of the biggest topics in algorithms. We search for strings in google. Now google is a verb! Searching for the presence of a string in another is string matching. There are a number of algorithms for string matching. We shall discuss the simplest algorithm in this course.

The problem statement:

We have a string : called text hereafter : with n characters.

We have a string : called pattern hereafter; with m characters.

Check if the pattern occurs in the text. If yes, return the position as an index else return -1.

Let us examine the files : 3\_client.c 3\_mystr.h 3\_mystr.c

The function signature is as follows.

// pos in the text if pattern found in the text otherwise -1

int mymatch(char text[], int n, char pattern[], int m);

Let us examine the algorithm.

int mymatch(char text[], int n, char pattern[], int m)

{

int i; int j;

int res = -1;

// outer loop : walk thro the text

for(i = 0; res == -1 && i <= n - m; ++i)

{

// inner loop : walk thto the pattern

for(j = 0; j < m && text[i + j] == pattern[j]; ++j)

{

}

if(j == m)

{

res = i;

}

}

return res;

}

The outer loop indicates the position in the text from which we start matching the pattern. We start with i = 0. We increment i on an unsuccessful match. When the number of characters is less than the length of the pattern, we shall stop – i < (n – m) indicating the element is not found.

The inner loop compares the jth character of pattern with the ith character of text starting with j = 0. If the whole pattern is matched, then j == m. then we set a variable res to indicate the position of the match. If there is a mismatch, we start the iteration in the outerloop with the next value of i.

We exit the outer for loop when the string is matched – res will be no more -1 or when the matching fails – i < (n – m) becomes false. In any case, the variable res will have the right result.

**Difference between pointer to a string literal and an array of char:**

Let us examine the file 4\_str.c.

char x[] = "pes";

char\* y = "pes";

x is an array of 4 characters with ‘p’, ‘e’, ‘s’, ‘\0’. This is like a book owned by me.

y is a pointer to a string constant or literal. This is a like a library book – borrowed and not owned by me.

We can change the elements of x.

x[0] = ‘P’; // we can tear our book.

y0] = ‘p’ // undefined behaviour. Do not know what happens if you tear the pages of a borrowed book. I do not encourage to try it.

We can increment y. y is a pointer variable. Therefore it can be incremented.

We can not increment x as it is an array name. x is a constant.

Thats all about arrays, strings and pointers.