(/wiki/Rosetta_Code)

I'm working on modernizing Rosetta Code's infrastructure. Starting with communications. Please accept this time-limited open invite to RC's Slack. (https://join.slack.com/t/rosettacode/shared_invite/zt-glwmugtu-xpMPcqHs0u6MsK5zCmJF~Q). --Michael Mol (/wiki/User:Short_Circuit) (talk (/wiki/User_talk:Short_Circuit)) 20:59, 30 May 2020 (UTC)

String matching

Task

Given two strings, demonstrate the following three types of string matching:

1. Determining if the first string starts with second string
2. Determining if the first string contains the second string at any location
3. Determining if the first string ends with the second string

Optional requirements:

1. Print the location of the match for part 2
2. Handle multiple occurrences of a string for part 2.

Other tasks related to string operations: [Expand]

Contents

(/wiki/Category:Solutions_by

String matching
You are encouraged to solve this task (/wiki/Rosetta_Code:Solve_a according to the task description, using any language you may know.

Basic Data Operation
This is a basic data operation. It represents a fundamental action on a basic data type.

You may see other such operations in the Basic Data Operations (/wiki/Category:Basic_Data_ category, or:

Integer Operations
Arithmetic (/wiki/Arithmetic/Integer) | Comparison (/wiki/Integer_comparison)

Boolean Operations
Bitwise (/wiki/Bitwise_operations | Logical (/wiki/Logical_operations)

String Operations
Concatenation (/wiki/String_concatenation) | Interpolation (/wiki/String_interpolation_(includ | Comparison (/wiki/String_comparison) | **Matching**

Memory Operations
Pointers & references (/wiki/Pointers_and_references) | Addresses (/wiki/Address_of_a_variable)

11l (/wiki/Category:11l)

Translation of: Python

```
print('abcd'.starts_with('ab'))
print('abcd'.ends_with('zn'))
print('bb' C 'abab')
print('ab' C 'abab')
print('abab'.find('bb') ? -1)
print('abab'.find('ab') ? -1)
```

Output:

```
1B
0B
0B
1B
-1
0
```

360 Assembly (/wiki/Category:360_Assembly)

```
*        String matching           04/04/2017
STRMATCH CSECT
         USING  STRMATCH,R15
         XPRNT  SS,L'SS
*
         CLC    SS(L'S1),S1
         BNE    NOT1
         XPRNT  =C'-- STARTS WITH',14
         XPRNT  S1,L'S1
NOT1     EQU    *
*
         CLC    SS+L'SS-L'S2(L'S2),S2
         BNE    NOT2
         XPRNT  =C'-- ENDS WITH',12
         XPRNT  S2,L'S2
NOT2     EQU    *
*
         LA     R0,L'SS-L'S3+1
         LA     R1,SS
LOOP     CLC    0(L'S3,R1),S3
         BNE    NOT3
         XPRNT  =C'-- CONTAINS',11
         XPRNT  S3,L'S3
NOT3     LA     R1,1(R1)
         BCT    R0,LOOP
*
         BR     R14
SS       DC     CL6'ABCDEF'
S1       DC     CL2'AB'
S2       DC     CL2'EF'
S3       DC     CL2'CD'
PG       DC     CL80' '
         YREGS
         END    STRMATCH
```

Output:

```
ABCDEF
-- STARTS WITH
AB
-- ENDS WITH
EF
-- CONTAINS
CD
```

AArch64 Assembly (/wiki/Category:AArch64_Assembly)

Works with: as (/mw/index.php?title=As&action=edit&redlink=1) version Raspberry Pi 3B version Buster 64 bits

```
/* ARM assembly AARCH64 Raspberry PI 3B */
/*  program strMatching64.s   */

/*********************************************/
/* Constantes file                           */
/*********************************************/
/* for this file see task include a file in language AArch64 assembly*/
.include "../includeConstantesARM64.inc"
/*********************************************/
/* Initialized data                          */
/*********************************************/
.data
szMessFound:             .asciz "String found. \n"
szMessNotFound:          .asciz "String not found. \n"
szString:                .asciz "abcdefghijklmnopqrstuvwxyz"
szString2:               .asciz "abc"
szStringStart:           .asciz "abcd"
szStringEnd:             .asciz "xyz"
szStringStart2:          .asciz "abcd"
szStringEnd2:            .asciz "xabc"
szCarriageReturn:        .asciz "\n"
/*********************************************/
/* UnInitialized data                        */
/*********************************************/
.bss
/*********************************************/
/*  code section                             */
/*********************************************/
.text
.global main
main:

    ldr x0,qAdrszString                  // address input string
    ldr x1,qAdrszStringStart             // address search string

    bl searchStringDeb                   // Determining if the first string starts with second string
    cmp x0,0
    ble 1f
    ldr x0,qAdrszMessFound               // display message
    bl affichageMess
    b 2f
1:
    ldr x0,qAdrszMessNotFound
    bl affichageMess
2:
    ldr x0,qAdrszString                 // address input string
    ldr x1,qAdrszStringEnd              // address search string
    bl searchStringFin                  // Determining if the first string ends with the second string
    cmp x0,0
    ble 3f
    ldr x0,qAdrszMessFound              // display message
    bl affichageMess
    b 4f
3:
    ldr x0,qAdrszMessNotFound
    bl affichageMess
4:
    ldr x0,qAdrszString2               // address input string
    ldr x1,qAdrszStringStart2          // address search string

    bl searchStringDeb                 //
    cmp x0,0
    ble 5f
    ldr x0,qAdrszMessFound             // display message
    bl affichageMess
    b 6f
```

```
5:
    ldr x0,qAdrszMessNotFound
    bl affichageMess
6:
    ldr x0,qAdrszString2               // address input string
    ldr x1,qAdrszStringEnd2            // address search string
    bl searchStringFin
    cmp x0,0
    ble 7f
    ldr x0,qAdrszMessFound            // display message
    bl affichageMess
    b 8f
7:
    ldr x0,qAdrszMessNotFound
    bl affichageMess
8:
    ldr x0,qAdrszString               // address input string
    ldr x1,qAdrszStringEnd            // address search string
    bl searchSubString                // Determining if the first string contains the second string at any location
    cmp x0,0
    ble 9f
    ldr x0,qAdrszMessFound            // display message
    bl affichageMess
    b 10f
9:
    ldr x0,qAdrszMessNotFound         // display substring result
    bl affichageMess
10:

100:                                  // standard end of the program
    mov x0,0                          // return code
    mov x8,EXIT                       // request to exit program
    svc 0                             // perform system call
qAdrszMessFound:          .quad szMessFound
qAdrszMessNotFound:       .quad szMessNotFound
qAdrszString:             .quad szString
qAdrszString2:            .quad szString2
qAdrszStringStart:        .quad szStringStart
qAdrszStringEnd:          .quad szStringEnd
qAdrszStringStart2:       .quad szStringStart2
qAdrszStringEnd2:         .quad szStringEnd2
qAdrszCarriageReturn:     .quad szCarriageReturn
/******************************************************************/
/*     search substring at begin of input string                  */
/******************************************************************/
/* x0 contains the address of the input string */
/* x1 contains the address of substring */
/* x0 returns 1 if find or 0 if not or -1 if error */
searchStringDeb:
    stp x1,lr,[sp,-16]!            // save  registers
    stp x2,x3,[sp,-16]!            // save  registers
    mov x3,0                       // counter byte  string
    ldrb w4,[x1,x3]                // load first byte of substring
    cbz x4,99f                     // empty string ?
1:
    ldrb w2,[x0,x3]                // load byte string input
    cbz x2,98f                     // zero final ?
    cmp x4,x2                      // bytes equals ?
    bne 98f                        // no not find
    add x3,x3,1                    // increment counter
    ldrb w4,[x1,x3]                // and load next byte of substring
    cbnz x4,1b                     // zero final ?
    mov x0,1                       // yes is ok
    b 100f
98:
    mov x0,0                       // not find
    b 100f
99:
    mov x0,-1                      // error
100:
    ldp x2,x3,[sp],16              // restaur  2 registers
    ldp x1,lr,[sp],16              // restaur  2 registers
    ret                            // return to address lr x30

/******************************************************************/
/*     search substring at end of input string                    */
/******************************************************************/
/* x0 contains the address of the input string */
/* x1 contains the address of substring */
/* x0 returns 1 if find or 0 if not or -1 if error */
searchStringFin:
```

```
    stp x1,lr,[sp,-16]!           // save  registers
    stp x2,x3,[sp,-16]!           // save  registers
    stp x4,x5,[sp,-16]!           // save  registers
    mov x3,0                      // counter byte  string
                                  // search the last character of substring
1:
    ldrb w4,[x1,x3]               // load byte of substring
    cmp x4,#0                     // zero final ?
    add x2,x3,1
    csel x3,x2,x3,ne              // no increment counter
    //addne x3,#1                 // no increment counter
    bne 1b                        // and loop
    cbz x3,99f                    // empty string ?

    sub x3,x3,1                   // index of last byte
    ldrb w4,[x1,x3]               // load last byte of substring
                                  // search the last character of string
    mov x2,0                      // index last character
2:
    ldrb w5,[x0,x2]               // load first byte of substring
    cmp x5,0                      // zero final ?
    add x5,x2,1                   // no -> increment counter
    csel x2,x5,x2,ne
    //addne x2,#1                 // no -> increment counter
    bne 2b                        // and loop
    cbz x2,98f                    // empty input string ?
    sub x2,x2,1                   // index last character
3:
    ldrb w5,[x0,x2]               // load byte string input
    cmp x4,x5                     // bytes equals ?
    bne 98f                       // no -> not found
    subs x3,x3,1                  // decrement counter
    blt 97f                       //  ok found
    subs x2,x2,1                  // decrement counter input string
    blt 98f                       // if zero -> not found
    ldrb w4,[x1,x3]               // load previous byte of substring
    b 3b                          // and loop
97:
    mov x0,1                      // yes is ok
    b 100f
98:
    mov x0,0                      // not found
    b 100f
99:
    mov x0,-1                     // error
100:
    ldp x4,x5,[sp],16             // restaur  2 registers
    ldp x2,x3,[sp],16             // restaur  2 registers
    ldp x1,lr,[sp],16             // restaur  2 registers
    ret                           // return to address lr x30

/*********************************************************************/
/*   search a substring in the string                                */
/*********************************************************************/
/* x0 contains the address of the input string */
/* x1 contains the address of substring */
/* x0 returns index of substring in string or -1 if not found */
searchSubString:
    stp x1,lr,[sp,-16]!           // save  registers
    stp x2,x3,[sp,-16]!           // save  registers
    stp x4,x5,[sp,-16]!           // save  registers
    mov x2,0                      // counter byte input string
    mov x3,0                      // counter byte string
    mov x6,-1                     // index found
    ldrb w4,[x1,x3]
1:
    ldrb w5,[x0,x2]               // load byte string
    cbz x5,99f                    // zero final ?
    cmp x5,x4                     // compare character
    beq 2f
    mov x6,-1                     // no equals - > raz index
    mov x3,0                      // and raz counter byte
    add x2,x2,1                   // and increment counter byte
    b 1b                          // and loop
2:                                // characters equals
    cmp x6,-1                     // first characters equals ?
    csel x6,x2,x6,eq              // yes -> index begin in x6
    //moveq x6,x2                 // yes -> index begin in x6
    add x3,x3,1                   // increment counter substring
    ldrb w4,[x1,x3]               // and load next byte
    cmp x4,0                      // zero final ?
```

```
    beq 3f                          // yes -> end search
    add x2,x2,1                     // else increment counter string
    b 1b                            // and loop
3:
    mov x0,x6
    b 100f

98:
    mov x0,0                       // not found
    b 100f
99:
    mov x0,-1                      // error
100:
    ldp x4,x5,[sp],16               // restaur  2 registers
    ldp x2,x3,[sp],16               // restaur  2 registers
    ldp x1,lr,[sp],16               // restaur  2 registers
    ret                             // return to address lr x30
/******************************************************/
/*        File Include fonctions                      */
/******************************************************/
/* for this file see task include a file in language AArch64 assembly */
.include "../includeARM64.inc"
```

Ada (/wiki/Category:Ada)

```
with Ada.Strings.Fixed;  use Ada.Strings.Fixed;
with Ada.Text_IO;        use Ada.Text_IO;

procedure Match_Strings is
   S1 : constant String := "abcd";
   S2 : constant String := "abab";
   S3 : constant String := "ab";
begin
   if S1'Length >= S3'Length and then S1 (S1'First..S1'First + S3'Length - 1) = S3 then
      Put_Line ("'" & S1 & "' starts with '" & S3 & "'");
   end if;
   if S2'Length >= S3'Length and then S2 (S2'Last - S3'Length + 1..S2'Last) = S3 then
      Put_Line ("'" & S2 & "' ends with '" & S3 & "'");
   end if;
   Put_Line ("'" & S3 & "' first appears in '" & S1 & "' at" & Integer'Image (Index (S1, S3)));
   Put_Line
   (  "'" & S3 & "' appears in '" & S2 & "' &
      Integer'Image (Ada.Strings.Fixed.Count (S2, S3)) & " times"
   );
end Match_Strings;
```

Output:

```
'abcd' starts with 'ab'
'abab' ends with 'ab'
'ab' first appears in 'abcd' at 1
'ab' appears in 'abab' 2 times
```

Aime (/wiki/Category:Aime)

```
text t;
data b;

b = "Bangkok";

t = "Bang";

o_form("starts with, embeds, ends with \"~\": ~, ~, ~\n", t, b.seek(t) == 0,
       b.seek(t) != -1,
       b.seek(t) != -1 && b.seek(t) + ~t == ~b);

t = "ok";

o_form("starts with, embeds, ends with \"~\": ~, ~, ~\n", t, b.seek(t) == 0,
       b.seek(t) != -1,
       b.seek(t) != -1 && b.seek(t) + ~t == ~b);

t = "Summer";

o_form("starts with, embeds, ends with \"~\": ~, ~, ~\n", t, b.seek(t) == 0,
       b.seek(t) != -1,
       b.seek(t) != -1 && b.seek(t) + ~t == ~b);
```

Output:

```
starts with, embeds, ends with "Bang": 1, 1, 0
starts with, embeds, ends with "ok": 0, 1, 1
starts with, embeds, ends with "Summer": 0, 0, 0
```

ALGOL 68 (/wiki/Category:ALGOL_68)

Translation of: python
Works with: ALGOL 68 (/wiki/ALGOL_68) version Revision 1 - no extensions to language used
Works with: ALGOL 68G (/wiki/ALGOL_68G) version Any - tested with release 1.18.0-9h.tiny (https://sourceforge.net/projects/algol68/files/algol68g/algol68g-1.18.0/algol68g-1.18.0-9h.tiny.el5.centos.fc11.i386.rpm/download)

```
# define some appropriate OPerators #
PRIO STARTSWITH = 5, ENDSWITH = 5;
OP STARTSWITH = (STRING str, prefix)BOOL: # assuming LWB = 1 #
  IF UPB str < UPB prefix THEN FALSE ELSE str[:UPB prefix]=prefix FI;
OP ENDSWITH = (STRING str, suffix)BOOL: # assuming LWB = 1 #
  IF UPB str < UPB suffix THEN FALSE ELSE str[UPB str-UPB suffix+1:]=suffix FI;

INT loc, loc2;

print((
  "abcd" STARTSWITH "ab", # returns TRUE #
  "abcd" ENDSWITH "zn", # returns FALSE #
  string in string("bb",loc,"abab"), # returns FALSE #
  string in string("ab",loc,"abab"), # returns TRUE #
  (string in string("bb",loc,"abab")|loc|-1), # returns -1 #
  (string in string("ab",loc,"abab")|loc|-1), # returns +1 #
  (string in string("ab",loc2,"abab"[loc+1:])|loc+loc2|-1) # returns +3 #
))
```

Output:

```
TFFT         -1         +1         +3
```

AppleScript (/wiki/Category:AppleScript)

```
set stringA to "I felt happy because I saw the others were happy and because I knew I should feel happy, but I wasn’t really happy."

set string1 to "I felt happy"
set string2 to "I should feel happy"
set string3 to "I wasn't really happy"

-- Determining if the first string starts with second string
stringA starts with string1  --> true

-- Determining if the first string contains the second string at any location
stringA contains string2     --> true

-- Determining if the first string ends with the second string
stringA ends with string3    --> false

-- Print the location of the match for part 2
offset of string2 in stringA --> 69
```

AppleScript doesn't have a builtin means of matching multiple occurrences of a substring, however one can redefine the existing **offset** command to add this functionality:

```
-- Handle multiple occurrences of a string for part 2
on offset of needle in haystack
	local needle, haystack

	if the needle is not in the haystack then return {}
	set my text item delimiters to the needle
	script
		property N : needle's length
		property t : {1 - N} & haystack's text items
	end script

	tell the result
		repeat with i from 2 to (its t's length) - 1
			set x to item i of its t
			set y to item (i - 1) of its t
			set item i of its t to (its N) + (x's length) + y
		end repeat

		items 2 thru -2 of its t
	end tell
end offset

offset of "happy" in stringA --> {8, 44, 83, 110}
```

or, defining an **offsets** function in terms of a more general **findIndices**:

```
-- offsets :: String -> String -> [Int]
on offsets(needle, haystack)
    script match
        property mx : length of haystack
        property d : (length of needle) - 1
        on |λ|(x, i, xs)
            set z to d + i
            mx ≥ z and needle = text i thru z of xs
        end |λ|
    end script

    findIndices(match, haystack)
end offsets


-- TEST ----------------------------------------------------
on run
    set txt to "I felt happy because I saw the others " & ¬
        "were happy and because I knew I should " & ¬
        "feel happy, but I wasn't really happy."

    offsets("happy", txt)

    --> {8, 44, 83, 110}
end run


-- GENERIC -------------------------------------------------

-- concatMap :: (a -> [b]) -> [a] -> [b]
on concatMap(f, xs)
    set lng to length of xs
    set acc to {}
    tell mReturn(f)
        repeat with i from 1 to lng
            set acc to acc & (|λ|(item i of xs, i, xs))
        end repeat
    end tell
    return acc
end concatMap


-- findIndices :: (a -> Bool) -> [a] -> [Int]
-- findIndices :: (String -> Bool) -> String -> [Int]
on findIndices(p, xs)
    script go
        property f : mReturn(p)
        on |λ|(x, i, xs)
            if f's |λ|(x, i, xs) then
                {i}
            else
                {}
            end if
        end |λ|
    end script
    concatMap(go, xs)
end findIndices


-- Lift 2nd class handler function into 1st class script wrapper
-- mReturn :: First-class m => (a -> b) -> m (a -> b)
on mReturn(f)
    if script is class of f then
        f
    else
        script
            property |λ| : f
        end script
    end if
end mReturn
```

Output:

```
{8, 44, 83, 110}
```

ARM Assembly (/wiki/Category:ARM_Assembly)

Works with: as (/mw/index.php?title=As&action=edit&redlink=1) version Raspberry Pi

```
/* ARM assembly Raspberry PI  */
/*  program strMatching.s   */

/* Constantes    */
.equ STDOUT, 1                          @ Linux output console
.equ EXIT,   1                          @ Linux syscall
.equ WRITE,  4                          @ Linux syscall

/* Initialized data */
.data
szMessFound:             .asciz "String found. \n"
szMessNotFound:          .asciz "String not found. \n"
szString:                .asciz "abcdefghijklmnopqrstuvwxyz"
szString2:               .asciz "abc"
szStringStart:           .asciz "abcd"
szStringEnd:             .asciz "xyz"
szStringStart2:          .asciz "abcd"
szStringEnd2:            .asciz "xabc"
szCarriageReturn:        .asciz "\n"

/* UnInitialized data */
.bss

/*  code section */
.text
.global main
main:

    ldr r0,iAdrszString                         @ address input string
    ldr r1,iAdrszStringStart                    @ address search string

    bl searchStringDeb                          @ Determining if the first string starts with second string
    cmp r0,#0
    ble 1f
    ldr r0,iAdrszMessFound                      @ display message
    bl affichageMess
    b 2f
1:
    ldr r0,iAdrszMessNotFound
    bl affichageMess
2:
    ldr r0,iAdrszString                         @ address input string
    ldr r1,iAdrszStringEnd                      @ address search string
    bl searchStringFin                          @ Determining if the first string ends with the second string
    cmp r0,#0
    ble 3f
    ldr r0,iAdrszMessFound                      @ display message
    bl affichageMess
    b 4f
3:
    ldr r0,iAdrszMessNotFound
    bl affichageMess
4:
    ldr r0,iAdrszString2                        @ address input string
    ldr r1,iAdrszStringStart2                   @ address search string

    bl searchStringDeb                          @
    cmp r0,#0
    ble 5f
    ldr r0,iAdrszMessFound                      @ display message
    bl affichageMess
    b 6f
5:
    ldr r0,iAdrszMessNotFound
    bl affichageMess
6:
    ldr r0,iAdrszString2                        @ address input string
    ldr r1,iAdrszStringEnd2                     @ address search string
    bl searchStringFin
    cmp r0,#0
    ble 7f
    ldr r0,iAdrszMessFound                      @ display message
    bl affichageMess
    b 8f
7:
    ldr r0,iAdrszMessNotFound
    bl affichageMess
8:
    ldr r0,iAdrszString                         @ address input string
    ldr r1,iAdrszStringEnd                      @ address search string
```

```
    bl searchSubString                      @ Determining if the first string contains the second string at any location
    cmp r0,#0
    ble 9f
    ldr r0,iAdrszMessFound                  @ display message
    bl affichageMess
    b 10f
9:
    ldr r0,iAdrszMessNotFound               @ display substring result
    bl affichageMess
10:

100:                                        @ standard end of the program
    mov r0, #0                              @ return code
    mov r7, #EXIT                           @ request to exit program
    svc 0                                   @ perform system call
iAdrszMessFound:          .int szMessFound
iAdrszMessNotFound:       .int szMessNotFound
iAdrszString:             .int szString
iAdrszString2:            .int szString2
iAdrszStringStart:        .int szStringStart
iAdrszStringEnd:          .int szStringEnd
iAdrszStringStart2:       .int szStringStart2
iAdrszStringEnd2:         .int szStringEnd2
iAdrszCarriageReturn:     .int szCarriageReturn
/******************************************************************/
/*     search substring at begin of input string                  */
/******************************************************************/
/* r0 contains the address of the input string */
/* r1 contains the address of substring */
/* r0 returns 1 if find or 0 if not or -1 if error */
searchStringDeb:
    push {r1-r4,lr}                         @ save  registers
    mov r3,#0                               @ counter byte  string
    ldrb r4,[r1,r3]                         @ load first byte of substring
    cmp r4,#0                               @ empty string ?
    moveq r0,#-1                            @ error
    beq 100f
1:
    ldrb r2,[r0,r3]                         @ load byte string input
    cmp r2,#0                               @ zero final ?
    moveq r0,#0                             @ not find
    beq 100f
    cmp r4,r2                               @ bytes equals ?
    movne r0,#0                             @ no not find
    bne 100f
    add r3,#1                               @ increment counter
    ldrb r4,[r1,r3]                         @ and load next byte of substring
    cmp r4,#0                               @ zero final ?
    bne 1b                                  @ no -> loop
    mov r0,#1                               @ yes is ok
100:
    pop {r1-r4,lr}                          @ restaur registers
    bx lr                                   @ return

/******************************************************************/
/*     search substring at end of input string                    */
/******************************************************************/
/* r0 contains the address of the input string */
/* r1 contains the address of substring */
/* r0 returns 1 if find or 0 if not or -1 if error */
searchStringFin:
    push {r1-r5,lr}                         @ save  registers
    mov r3,#0                               @ counter byte  string
                                            @ search the last character of substring
1:
    ldrb r4,[r1,r3]                         @ load byte of substring
    cmp r4,#0                               @ zero final ?
    addne r3,#1                             @ no increment counter
    bne 1b                                  @ and loop
    cmp r3,#0                               @ empty string ?
    moveq r0,#-1                            @ error
    beq 100f
    sub r3,#1                               @ index of last byte
    ldrb r4,[r1,r3]                         @ load last byte of substring
                                            @ search the last character of string
    mov r2,#0                               @ index last character
2:
    ldrb r5,[r0,r2]                         @ load first byte of substring
    cmp r5,#0                               @ zero final ?
    addne r2,#1                             @ no -> increment counter
    bne 2b                                  @ and loop
```

```
    cmp r2,#0                           @ empty input string ?
    moveq r0,#0                         @ yes -> not found
    beq 100f
    sub r2,#1                           @ index last character
3:
    ldrb r5,[r0,r2]                     @ load byte string input
    cmp r4,r5                           @ bytes equals ?
    movne r0,#0                         @ no -> not found
    bne 100f
    subs r3,#1                          @ decrement counter
    movlt r0,#1                         @ if zero -> ok found
    blt 100f
    subs r2,#1                          @ decrement counter input string
    movlt r0,#0                         @ if zero -> not found
    blt 100f
    ldrb r4,[r1,r3]                     @ load previous byte of substring
    b 3b                                @ and loop

100:
    pop {r1-r5,lr}                      @ restaur registers
    bx lr                               @ return

/******************************************************************/
/*   search a substring in the string                             */
/******************************************************************/
/* r0 contains the address of the input string */
/* r1 contains the address of substring */
/* r0 returns index of substring in string or -1 if not found */
searchSubString:
    push {r1-r6,lr}                     @ save registers
    mov r2,#0                           @ counter byte input string
    mov r3,#0                           @ counter byte string
    mov r6,#-1                          @ index found
    ldrb r4,[r1,r3]
1:
    ldrb r5,[r0,r2]                     @ load byte string
    cmp r5,#0                           @ zero final ?
    moveq r0,#-1                        @ yes returns error
    beq 100f
    cmp r5,r4                           @ compare character
    beq 2f
    mov r6,#-1                          @ no equals - > raz index
    mov r3,#0                           @ and raz counter byte
    add r2,#1                           @ and increment counter byte
    b 1b                                @ and loop
2:                                      @ characters equals
    cmp r6,#-1                          @ first characters equals ?
    moveq r6,r2                         @ yes -> index begin in r6
    add r3,#1                           @ increment counter substring
    ldrb r4,[r1,r3]                     @ and load next byte
    cmp r4,#0                           @ zero final ?
    beq 3f                              @ yes -> end search
    add r2,#1                           @ else increment counter string
    b 1b                                @ and loop
3:
    mov r0,r6
100:
    pop {r1-r6,lr}                      @ restaur registers
    bx lr

/******************************************************************/
/*     display text with size calculation                         */
/******************************************************************/
/* r0 contains the address of the message */
affichageMess:
    push {r0,r1,r2,r7,lr}               @ save  registers
    mov r2,#0                           @ counter length */
1:                                      @ loop length calculation
    ldrb r1,[r0,r2]                     @ read octet start position + index
    cmp r1,#0                           @ if 0 its over
    addne r2,r2,#1                      @ else add 1 in the length
    bne 1b                              @ and loop
                                        @ so here r2 contains the length of the message
    mov r1,r0                           @ address message in r1
    mov r0,#STDOUT                      @ code to write to the standard output Linux
    mov r7, #WRITE                      @ code call system "write"
    svc #0                              @ call system
    pop {r0,r1,r2,r7,lr}                @ restaur registers
    bx lr                               @ return
```

Arturo (/wiki/Category:Arturo)

```
print prefix? "abcd" "ab"
print prefix? "abcd" "cd"
print suffix? "abcd" "ab"
print suffix? "abcd" "cd"

print contains? "abcd" "ab"
print contains? "abcd" "xy"

print in? "ab" "abcd"
print in? "xy" "abcd"

print index "abcd" "bc"
print index "abcd" "xy"
```

Output:

```
true
false
false
true
true
false
true
false
1
null
```

AutoHotkey (/wiki/Category:AutoHotkey)

```
String1 = abcd
String2 = abab

If (SubStr (http://www.autohotkey.com/docs/Functions.htm#BuiltIn)(String1,1,StrLen (http://www.autohotkey.com/docs/Functions.htm#BuiltIn)(String2)) = String2)
 MsgBox (http://www.autohotkey.com/docs/commands/MsgBox.htm), "%String1%" starts with "%String2%".
IfInString (http://www.autohotkey.com/docs/commands/IfInString.htm), String1, %String2%
{
 Position := InStr (http://www.autohotkey.com/docs/Functions.htm#BuiltIn)(String1,String2)
 StringReplace (http://www.autohotkey.com/docs/commands/StringReplace.htm), String1, String1, %String2%, %String2%, UseErrorLevel
 MsgBox (http://www.autohotkey.com/docs/commands/MsgBox.htm), "%String1%" contains "%String2%" at position %Position%`, and appears %ErrorLevel% times.
}
StringRight (http://www.autohotkey.com/docs/commands/StringRight.htm), TempVar, String1, StrLen (http://www.autohotkey.com/docs/Functions.htm#BuiltIn)(String2)
If TempVar = %String2%
 MsgBox (http://www.autohotkey.com/docs/commands/MsgBox.htm), "%String1%" ends with "%String2%".
```

AutoIt (/wiki/Category:AutoIt)

```
$string1 = "arduinoardblobard"
$string2 = "ard"

; == Determining if the first string starts with second string
If (https://www.autoitscript.com/autoit3/docs/keywords.htm) StringLeft (https://www.autoitscript.com/autoit3/docs/functions/StringLeft
.htm)($string1, StringLen (https://www.autoitscript.com/autoit3/docs/functions/StringLen.htm)($string2)) = $string2 Then (https://www.
autoitscript.com/autoit3/docs/keywords.htm)
        ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string starts with 2nd string." & @CR
LF (https://www.autoitscript.com/autoit3/docs/macros.htm))
Else (https://www.autoitscript.com/autoit3/docs/keywords.htm)
        ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string does'nt starts with 2nd string
." & @CRLF (https://www.autoitscript.com/autoit3/docs/macros.htm))
EndIf (https://www.autoitscript.com/autoit3/docs/keywords.htm)

; == Determining if the first string contains the second string at any location
; == Print the location of the match for part 2
; == Handle multiple occurrences of a string for part 2
$start = 1
$count = 0
$pos = StringInStr (https://www.autoitscript.com/autoit3/docs/functions/StringInStr.htm)($string1, $string2)
While (https://www.autoitscript.com/autoit3/docs/keywords.htm) $pos
        $count += 1
        ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string contains 2nd string at positio
n: " & $pos & @CRLF (https://www.autoitscript.com/autoit3/docs/macros.htm))
        $pos = StringInStr (https://www.autoitscript.com/autoit3/docs/functions/StringInStr.htm)($string1, $string2, 0, 1, $start + $p
os + StringLen (https://www.autoitscript.com/autoit3/docs/functions/StringLen.htm)($string2))
WEnd (https://www.autoitscript.com/autoit3/docs/keywords.htm)
If (https://www.autoitscript.com/autoit3/docs/keywords.htm) $count = 0 Then (https://www.autoitscript.com/autoit3/docs/keywords.htm) C
onsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string does'nt contain 2nd string." & @CRLF (h
ttps://www.autoitscript.com/autoit3/docs/macros.htm))

; == Determining if the first string ends with the second string
If (https://www.autoitscript.com/autoit3/docs/keywords.htm) StringRight (https://www.autoitscript.com/autoit3/docs/functions/StringRig
ht.htm)($string1, StringLen (https://www.autoitscript.com/autoit3/docs/functions/StringLen.htm)($string2)) = $string2 Then (https://ww
w.autoitscript.com/autoit3/docs/keywords.htm)
        ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string ends with 2nd string." & @CRLF
 (https://www.autoitscript.com/autoit3/docs/macros.htm))
Else (https://www.autoitscript.com/autoit3/docs/keywords.htm)
        ConsoleWrite (https://www.autoitscript.com/autoit3/docs/functions/ConsoleWrite.htm)("1st string does'nt ends with 2nd string."
 & @CRLF (https://www.autoitscript.com/autoit3/docs/macros.htm))
EndIf (https://www.autoitscript.com/autoit3/docs/keywords.htm)
```

AWK (/wiki/Category:AWK)

```
#!/usr/bin/awk -f
{
    if ($1 ~ "^"$2) {
        print $1" begins with "$2;
    } else {
        print $1" does not begin with "$2;
    }

    if ($1 ~ $2) {
        print $1" contains "$2;
    } else {
        print $1" does not contain "$2;
    }

    if ($1 ~ $2"$") {
        print $1" ends with "$2;
    } else {
        print $1" does not end with "$2;
    }
}
```

BASIC (/wiki/Category:BASIC)

Works with: QBasic (/wiki/QBasic)

```
first$ = "qwertyuiop"

'Determining if the first string starts with second string
second$ = "qwerty"
IF LEFT$ (http://www.qbasicnews.com/qboho/qckleft%24.shtml)(first$, LEN (http://www.qbasicnews.com/qboho/qcklen.shtml)(second$)) = sec
ond$ THEN
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' starts with '"; second$; "'"
ELSE
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' does not start with '"; second$; "'"
END (http://www.qbasicnews.com/qboho/qckend.shtml) IF

'Determining if the first string contains the second string at any location
'Print the location of the match for part 2
second$ = "wert"
x = INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml)(first$, second$)
IF x THEN
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' contains '"; second$; "' at position "; x
ELSE
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' does not contain '"; second$; "'"
END (http://www.qbasicnews.com/qboho/qckend.shtml) IF

' Determining if the first string ends with the second string
second$ = "random garbage"
IF RIGHT$ (http://www.qbasicnews.com/qboho/qckright%24.shtml)(first$, LEN (http://www.qbasicnews.com/qboho/qcklen.shtml)(second$)) = s
econd$ THEN
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' ends with '"; second$; "'"
ELSE
    PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) "'"; first$; "' does not end with '"; second$; "'"
END (http://www.qbasicnews.com/qboho/qckend.shtml) IF
```

Output:

```
'qwertyuiop' starts with 'qwerty'
'qwertyuiop' contains 'wert' at position  2
'qwertyuiop' does not end with 'random garbage'
```

Applesoft BASIC (/wiki/Category:Applesoft_BASIC)

```
10 A$ = "THIS, THAT, AND THE OTHER THING"
20 S$ = "TH"
30 DEF FN S(P) = MID$(A$, P, LEN(S$)) = S$
40 PRINT A$ : PRINT

110 S$(1) = "STARTS"
120 S$(0) = "DOES NOT START"
130 PRINT S$(FN S(1))" WITH "S$ : PRINT

210 R$ = "" : FOR I = 1 TO LEN(A$) - LEN(S$) : IF FN S(I) THEN R$ = R$ + STR$(I) + " "
220 NEXT I
230 IF LEN(R$) = 0 THEN PRINT "DOES NOT CONTAIN "S$
240 IF LEN(R$) THEN PRINT "CONTAINS "S$" LOCATED AT POSITION "R$
250 PRINT

310 E$(1) = "ENDS"
320 E$(0) =  "DOES NOT END"
330 PRINT E$(FN S(LEN(A$) - LEN(S$) + 1))" WITH "S$
```

Batch File (/wiki/Category:Batch_File)

```
::NOTE #1: This implementation might crash, or might not work properly if
::you put some of the CMD special characters (ex. %,!, etc) inside the strings.
::
::NOTE #2: The comparisons here are case-SENSITIVE.
::NOTE #3: Spaces in strings are considered.

@echo (https://www.ss64.com/nt/echo.html) off
setlocal (https://www.ss64.com/nt/setlocal.html) enabledelayedexpansion

::The main things...
set (https://www.ss64.com/nt/set.html) "str1=qwertyuiop"
set (https://www.ss64.com/nt/set.html) "str2=qwerty"
call (https://www.ss64.com/nt/call.html) :str2_lngth
call (https://www.ss64.com/nt/call.html) :matchbegin

set (https://www.ss64.com/nt/set.html) "str1=qweiuoiocghiioyiocxiisfguiioiuygvd"
set (https://www.ss64.com/nt/set.html) "str2=io"
call (https://www.ss64.com/nt/call.html) :str2_lngth
call (https://www.ss64.com/nt/call.html) :matchcontain

set (https://www.ss64.com/nt/set.html) "str1=blablabla"
set (https://www.ss64.com/nt/set.html) "str2=bbla"
call (https://www.ss64.com/nt/call.html) :str2_lngth
call (https://www.ss64.com/nt/call.html) :matchend

echo (https://www.ss64.com/nt/echo.html).
pause (https://www.ss64.com/nt/pause.html)
exit (https://www.ss64.com/nt/exit.html) /b 0
::/The main things.

::The functions...
:matchbegin
echo (https://www.ss64.com/nt/echo.html).
if (https://www.ss64.com/nt/if.html) "!str1:~0,%length%!"=="!str2!" (
        echo (https://www.ss64.com/nt/echo.html) "%str1%" begins with "%str2%".
) else (https://www.ss64.com/nt/else.html) (
        echo (https://www.ss64.com/nt/echo.html) "%str1%" does not (https://www.ss64.com/nt/not.html) begin with "%str2%".
)
goto (https://www.ss64.com/nt/goto.html) :EOF

:matchcontain
echo (https://www.ss64.com/nt/echo.html).
set (https://www.ss64.com/nt/set.html) curr=0&set (https://www.ss64.com/nt/set.html) exist (https://www.ss64.com/nt/exist.html)=0
:scanchrloop
if (https://www.ss64.com/nt/if.html) "!str1:~%curr%,%length%!"=="" (
        if (https://www.ss64.com/nt/if.html) !exist (https://www.ss64.com/nt/exist.html)!==0 echo (https://www.ss64.com/nt/echo.html)
"%str1%" does not (https://www.ss64.com/nt/not.html) contain "%str2%".
        goto (https://www.ss64.com/nt/goto.html) :EOF
)
if (https://www.ss64.com/nt/if.html) "!str1:~%curr%,%length%!"=="!str2!" (
        echo (https://www.ss64.com/nt/echo.html) "%str1%" contains "%str2%". ^(in (https://www.ss64.com/nt/in.html) Position %curr%^)
        set (https://www.ss64.com/nt/set.html) exist (https://www.ss64.com/nt/exist.html)=1
)
set (https://www.ss64.com/nt/set.html) /a curr+=1&goto (https://www.ss64.com/nt/goto.html) scanchrloop

:matchend
echo (https://www.ss64.com/nt/echo.html).
if (https://www.ss64.com/nt/if.html) "!str1:~-%length%!"=="!str2!" (
        echo (https://www.ss64.com/nt/echo.html) "%str1%" ends with "%str2%".
) else (https://www.ss64.com/nt/else.html) (
        echo (https://www.ss64.com/nt/echo.html) "%str1%" does not (https://www.ss64.com/nt/not.html) end with "%str2%".
)
goto (https://www.ss64.com/nt/goto.html) :EOF

:str2_lngth
set (https://www.ss64.com/nt/set.html) length=0
:loop
if (https://www.ss64.com/nt/if.html) "!str2:~%length%,1!"=="" goto (https://www.ss64.com/nt/goto.html) :EOF
set (https://www.ss64.com/nt/set.html) /a length+=1
goto (https://www.ss64.com/nt/goto.html) loop
::/The functions.
```

Output:

```
"qwertyuiop" begins with "qwerty".

"qweiuoiocghiioyiocxiisfguiioiuygvd" contains "io". (in Position 6)
"qweiuoiocghiioyiocxiisfguiioiuygvd" contains "io". (in Position 12)
"qweiuoiocghiioyiocxiisfguiioiuygvd" contains "io". (in Position 15)
"qweiuoiocghiioyiocxiisfguiioiuygvd" contains "io". (in Position 26)

"blablabla" does not end with "bbla".

Press any key to continue . . .
```

BBC BASIC (/wiki/Category:BBC_BASIC)

```
      first$ = "The fox jumps over the dog"

      FOR test% = 1 TO 3
        READ second$
        starts% = FN_first_starts_with_second(first$, second$)
        IF starts% PRINT """ first$ """ starts with """ second$ """"
        ends% = FN_first_ends_with_second(first$, second$)
        IF ends% PRINT """ first$ """ ends with """ second$ """"
        where% = FN_first_contains_second_where(first$, second$)
        IF where% PRINT """ first$ """ contains """ second$ """ at position " ; where%
        howmany% = FN_first_contains_second_howmany(first$, second$)
        IF howmany% PRINT """ first$ """ contains """ second$ """ " ; howmany% " time(s)"
      NEXT
      DATA "The", "he", "dog"
      END

      DEF FN_first_starts_with_second(A$, B$)
      = B$ = LEFT$(A$, LEN(B$))

      DEF FN_first_ends_with_second(A$, B$)
      = B$ = RIGHT$(A$, LEN(B$))

      DEF FN_first_contains_second_where(A$, B$)
      = INSTR(A$, B$)

      DEF FN_first_contains_second_howmany(A$, B$)
      LOCAL I%, N% : I% = 0
      REPEAT
        I% = INSTR(A$, B$, I%+1)
        IF I% THEN N% += 1
      UNTIL I% = 0
      = N%
```

Output:

```
"The fox jumps over the dog" starts with "The"
"The fox jumps over the dog" contains "The" at position 1
"The fox jumps over the dog" contains "The" 1 time(s)
"The fox jumps over the dog" contains "he" at position 2
"The fox jumps over the dog" contains "he" 2 time(s)
"The fox jumps over the dog" ends with "dog"
"The fox jumps over the dog" contains "dog" at position 24
"The fox jumps over the dog" contains "dog" 1 time(s)
```

Bracmat (/wiki/Category:Bracmat)

Bracmat does pattern matching in expressions `subject:pattern` and in strings `@(subject:pattern)`. The (sub)pattern `?` is a wild card.

```
( (sentence="I want a number such that that number will be even.")
& out$(@(!sentence:I ?) & "sentence starts with 'I'" | "sentence does not start with 'I'")
& out$(@(!sentence:? such ?) & "sentence contains 'such'" | "sentence does not contain 'such'")
& out$(@(!sentence:? "even.") & "sentence ends with 'even.'" | "sentence does not end with 'even.'")
& 0:?N
& ( @(!sentence:? be (? & !N+1:?N & ~))
  | out$str$("sentence contains " !N " occurrences of 'be'")
  )
)
```

In the last line, Bracmat is forced by the always failing node `~` to backtrack until all occurrences of 'be' are found. Thereafter the pattern match expression fails. The interesting part is the side effect: while backtracking, the accumulator `N` keeps track of how many are found.

Output:

```
sentence starts with 'I'
sentence contains 'such'
sentence ends with 'even.'
sentence contains 3 occurrences of 'be'
```

C (/wiki/Category:C)

Case sensitive matching:

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

int startsWith(const char* container, const char* target)
{
  size_t clen = strlen (https://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(container), tlen = strlen (https://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(target);
  if (clen < tlen)
    return 0;
  return strncmp (https://www.opengroup.org/onlinepubs/009695399/functions/strncmp.html)(container, target, tlen) == 0;
}

int endsWith(const char* container, const char* target)
{
  size_t clen = strlen (https://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(container), tlen = strlen (https://www.opengroup.org/onlinepubs/009695399/functions/strlen.html)(target);
  if (clen < tlen)
    return 0;
  return strncmp (https://www.opengroup.org/onlinepubs/009695399/functions/strncmp.html)(container + clen - tlen, target, tlen) == 0;
}

int doesContain(const char* container, const char* target)
{
  return strstr (https://www.opengroup.org/onlinepubs/009695399/functions/strstr.html)(container, target) != 0;
}

int main(void)
{
  printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Starts with Test ( Hello,Hell ) : %d\n", startsWith("Hello","Hell"));
  printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Ends with Test ( Code,ode ) : %d\n", endsWith("Code","ode"));
  printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Contains Test ( Google,msn ) : %d\n", doesContain("Google","msn"));

  return 0;
}
```

Output:

```
Starts with Test ( Hello,Hell ) : 1
Ends with Test ( Code,ode ) : 1
Contains Test ( Google,msn ) : 0
```

Code without using string library to demonstrate how char strings are just pointers:

```c
#include <stdio.h>

/* returns 0 if no match, 1 if matched, -1 if matched and at end */
int s_cmp(const char *a, const char *b)
{
        char c1 = 0, c2 = 0;
        while (c1 == c2) {
                c1 = *(a++);
                if ('\0' == (c2 = *(b++)))
                        return c1 == '\0' ? -1 : 1;
        }
        return 0;
}

/* returns times matched */
int s_match(const char *a, const char *b)
{
        int i = 0, count = 0;
        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("matching `%s' with `%s':\n", a, b);

        while (a[i] != '\0') {
                switch (s_cmp(a + i, b)) {
                case -1:
                        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("matched: pos %d (at end)\n\n",
i);
                        return ++count;
                case 1:
                        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("matched: pos %d\n", i);
                        ++count;
                        break;
                }
                i++;
        }
        printf (https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("end match\n\n");
        return count;
}

int main()
{
        s_match("A Short String", "ort S");
        s_match("aBaBaBaBa", "aBa");
        s_match("something random", "Rand");

        return 0;
}
```

Output:

```
matching `A Short String' with `ort S':
matched: pos 4
end match

matching `aBaBaBaBa' with `aBa':
matched: pos 0
matched: pos 2
matched: pos 4
matched: pos 6 (at end)

matching `something random' with `Rand':
end match
```

C# (/wiki/Category:C_sharp)

Works with: Mono (/wiki/Mono) version 2.6

```
class Program
{
        public static void Main (string[] args)
        {
                var value = "abcd".StartsWith("ab");
                value = "abcd".EndsWith("zn"); //returns false
                value = "abab".Contains("bb"); //returns false
                value = "abab".Contains("ab"); //returns true
                int loc = "abab".IndexOf("bb"); //returns -1
                loc = "abab".IndexOf("ab"); //returns 0
                loc = "abab".IndexOf("ab",loc+1); //returns 2
        }
}
```

C++ (/wiki/Category:C%2B%2B)

```
#include <string>
using namespace std;

string s1="abcd";
string s2="abab";
string s3="ab";
//Beginning
s1.compare(0,s3.size(),s3)==0;
//End
s1.compare(s1.size()-s3.size(),s3.size(),s3)==0;
//Anywhere
s1.find(s2)//returns string::npos
int loc=s2.find(s3)//returns 0
loc=s2.find(s3,loc+1)//returns 2
```

Clojure (/wiki/Category:Clojure)

Translation of: Java

```
(def evals '((. "abcd" startsWith "ab")
             (. "abcd" endsWith "zn")
             (. "abab" contains "bb")
             (. "abab" contains "ab")
             (. "abab" indexOf "bb")
             (let [loc (. "abab" indexOf "ab")]
               (. "abab" indexOf "ab" (dec loc)))))

user> (for [i evals] [i (eval i)])
([(. "abcd" startsWith "ab") true] [(. "abcd" endsWith "zn") false] [(. "abab" contains "bb") false] [(. "abab" contains "ab") true] [
(. "abab" indexOf "bb") -1] [(let [loc (. "abab" indexOf "ab")] (. "abab" indexOf "ab" (dec loc))) 0])
```

CoffeeScript (/wiki/Category:CoffeeScript)

This example uses string slices, but a better implementation might use indexOf for slightly better performance.

```
matchAt = (s, frag, i) ->
  s[i...i+frag.length] == frag

startsWith = (s, frag) ->
  matchAt s, frag, 0

endsWith = (s, frag) ->
  matchAt s, frag, s.length - frag.length

matchLocations = (s, frag) ->
  (i for i in [0..s.length - frag.length] when matchAt s, frag, i)

console.log startsWith "tacoloco", "taco" # true
console.log startsWith "taco", "tacoloco" # false
console.log startsWith "tacoloco", "talk" # false
console.log endsWith "tacoloco", "loco" # true
console.log endsWith "loco", "tacoloco" # false
console.log endsWith "tacoloco", "yoco" # false
console.log matchLocations "bababab", "bab" # [0,2,4]
console.log matchLocations "xxx", "x" # [0,1,2]
```

Common Lisp (/wiki/Category:Common_Lisp)

```
(defun starts-with-p (str1 str2)
  "Determine whether `str1` starts with `str2`"
  (let ((p (search str2 str1)))
    (and p (= 0 p))))

(print (starts-with-p "foobar" "foo")) ; T
(print (starts-with-p "foobar" "bar")) ; NIL

(defun ends-with-p (str1 str2)
  "Determine whether `str1` ends with `str2`"
  (let ((p (mismatch str2 str1 :from-end T)))
    (or (not p) (= 0 p))))

(print (ends-with-p "foobar" "foo")) ; NIL
(print (ends-with-p "foobar" "bar")) ; T

(defun containsp (str1 str2)
  "Determine whether `str1` contains `str2`.
   Instead of just returning T, return a list of starting locations
   for every occurence of `str2` in `str1`"
   (unless (string-equal str2 "")
     (loop for p = (search str2 str1) then (search str2 str1 :start2 (1+ p))
           while p
           collect p)))

(print (containsp "foobar" "oba")) ; (2)
(print (containsp "abababa" "ba")) ; (1 3)
(print (containsp "foobar" "x"))   ; NIL
```

Component Pascal (/wiki/Category:Component_Pascal)

BlackBox Component Builder

```
MODULE StringMatch;
IMPORT StdLog,Strings;
CONST
        strSize = 1024;
        patSize = 256;

TYPE
        Matcher* = POINTER TO LIMITED RECORD
                str: ARRAY strSize OF CHAR;
                pat: ARRAY patSize OF CHAR;
                pos: INTEGER
        END;

PROCEDURE NewMatcher*(IN str: ARRAY OF CHAR): Matcher;
VAR
        m: Matcher;
BEGIN
        NEW(m);m.str := str$;m.pos:= 0;
        RETURN m
END NewMatcher;

PROCEDURE (m: Matcher) Match*(IN pat: ARRAY OF CHAR): INTEGER,NEW;
VAR
        pos: INTEGER;
BEGIN
        m.pat := pat$;
        pos := m.pos;
        Strings.Find(m.str,m.pat,pos,m.pos);
        RETURN m.pos
END Match;

PROCEDURE (m: Matcher) Next*(): INTEGER, NEW;
VAR
        pos: INTEGER;
BEGIN
        pos := m.pos + LEN(m.pat$);
        Strings.Find(m.str,m.pat,pos,m.pos);
        RETURN m.pos;
END Next;
```

```
(* Some Helper functions based on Strings module *)
PROCEDURE StartsWith(IN str: ARRAY OF CHAR;IN pat: ARRAY OF CHAR): BOOLEAN;
VAR
	pos: INTEGER;
BEGIN
	Strings.Find(str,pat,0,pos);
	RETURN pos = 0
END StartsWith;

PROCEDURE Contains(IN str: ARRAY OF CHAR;IN pat: ARRAY OF CHAR; OUT pos: INTEGER): BOOLEAN;
BEGIN
	Strings.Find(str,pat,0,pos);
	RETURN pos >= 0
END Contains;

PROCEDURE EndsWith(IN str: ARRAY OF CHAR;IN pat: ARRAY OF CHAR): BOOLEAN;
VAR
	pos: INTEGER;
BEGIN
	Strings.Find(str,pat,0,pos);
	RETURN pos + LEN(pat$) = LEN(str$)
END EndsWith;

PROCEDURE Do*;
CONST
	aStr = "abcdefghijklmnopqrstuvwxyz";
VAR
	pat: ARRAY 128 OF CHAR;
	res: BOOLEAN;
	at: INTEGER;
	m: Matcher;
BEGIN
	(* StartsWith *)
	pat := "abc";
	StdLog.String(aStr + " startsWith " + pat + " :>");StdLog.Bool(StartsWith(aStr,pat));StdLog.Ln;
	pat := "cba";
	StdLog.String(aStr + " startsWith " + pat + " :>");StdLog.Bool(StartsWith(aStr,pat));StdLog.Ln;
	pat := "def";
	StdLog.String(aStr + " startsWith " + pat + " :>");StdLog.Bool(StartsWith(aStr,pat));StdLog.Ln;
	StdLog.Ln;
	(* Contains *)
	pat := 'def';
	StdLog.String(aStr + " contains " + pat + " :>");StdLog.Bool(Contains(aStr,pat,at));
	StdLog.String(" at: ");StdLog.Int(at);StdLog.Ln;
	pat := 'efd';
	StdLog.String(aStr + " contains " + pat + " :>");StdLog.Bool(Contains(aStr,pat,at));
	StdLog.String(" at: ");StdLog.Int(at);StdLog.Ln;
	pat := 'abc';
	StdLog.String(aStr + " contains " + pat + " :>");StdLog.Bool(Contains(aStr,pat,at));
	StdLog.String(" at: ");StdLog.Int(at);StdLog.Ln;
	pat := 'xyz';
	StdLog.String(aStr + " contains " + pat + " :>");StdLog.Bool(Contains(aStr,pat,at));
	StdLog.String(" at: ");StdLog.Int(at);StdLog.Ln;
	StdLog.Ln;
	(* EndsWith *)
	pat := 'xyz';
	StdLog.String(aStr + " endsWith " + pat + " :>");StdLog.Bool(EndsWith(aStr,pat));StdLog.Ln;
	pat := 'zyx';
	StdLog.String(aStr + " endsWith " + pat + " :>");StdLog.Bool(EndsWith(aStr,pat));StdLog.Ln;
	pat := 'abc';
	StdLog.String(aStr + " endsWith " + pat + " :>");StdLog.Bool(EndsWith(aStr,pat));StdLog.Ln;
	pat:= 'def';
	StdLog.String(aStr + " endsWith " + pat + " :>");StdLog.Bool(EndsWith(aStr,pat));StdLog.Ln;
	StdLog.Ln;

	m := NewMatcher("abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz");
	StdLog.String("Matching 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz' against 'abc':> ");
	StdLog.Ln;
	StdLog.String("Match at: ");StdLog.Int(m.Match("abc"));StdLog.Ln;
	StdLog.String("Match at: ");StdLog.Int(m.Next());StdLog.Ln;
	StdLog.String("Match at: ");StdLog.Int(m.Next());StdLog.Ln
END Do;
END StringMatch.
```

Execute: ^Q StringMatching.Do

Output:

```
abcdefghijklmnopqrstuvwxyz startsWith abc :> $TRUE
abcdefghijklmnopqrstuvwxyz startsWith cba :> $FALSE
abcdefghijklmnopqrstuvwxyz startsWith def :> $FALSE

abcdefghijklmnopqrstuvwxyz contains def :> $TRUE at:  3
abcdefghijklmnopqrstuvwxyz contains efd :> $FALSE at:  -1
abcdefghijklmnopqrstuvwxyz contains abc :> $TRUE at:  0
abcdefghijklmnopqrstuvwxyz contains xyz :> $TRUE at:  23

abcdefghijklmnopqrstuvwxyz endsWith xyz :> $TRUE
abcdefghijklmnopqrstuvwxyz endsWith zyx :> $FALSE
abcdefghijklmnopqrstuvwxyz endsWith abc :> $FALSE
abcdefghijklmnopqrstuvwxyz endsWith def :> $FALSE

Matching 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz' against 'abc':>
Match at:  0
Match at:  26
Match at:  -1
```

D (/wiki/Category:D)

```
void main() {
    import std.stdio;
    import std.algorithm: startsWith, endsWith, find, countUntil;

    "abcd".startsWith("ab").writeln;      // true
    "abcd".endsWith("zn").writeln;        // false
    "abab".find("bb").writeln;            // empty array (no match)
    "abcd".find("bc").writeln;            // "bcd" (substring start
                                           //        at match)
    "abab".countUntil("bb").writeln;      // -1 (no match)
    "abab".countUntil("ba").writeln;      //  1 (index of 1st match)

    // std.algorithm.startsWith also works on arrays and ranges:
    [1, 2, 3].countUntil(3).writeln;      //  2
    [1, 2, 3].countUntil([2, 3]).writeln; //  1
}
```

Output:

```
true
false

bcd
-1
1
2
1
```

DCL (/wiki/Category:DCL)

```
$ first_string = p1
$ length_of_first_string = f$length( first_string )
$ second_string = p2
$ length_of_second_string = f$length( second_string )
$ offset = f$locate( second_string, first_string )
$ if offset .eq. 0
$ then
$  write sys$output "first string starts with second string"
$ else
$  write sys$output "first string does not start with second string"
$ endif
$ if offset .ne. length_of_first_string
$ then
$  write sys$output "first string contains the second string at location ", offset
$ else
$  write sys$output "first string does not contain the second string at any location"
$ endif
$ temp = f$extract( length_of_first_string - length_of_second_string, length_of_second_string, first_string )
$ if second_string .eqs. temp
$ then
$  write sys$output "first string ends with the second string"
$ else
$  write sys$output "first string does not end with the second string"
$ endif
```

Output:

```
$ @string_matching efabcdef ef
first string starts with second string
first string contains the second string at location 0
first string ends with the second string
$ @string_matching efabcdef ab
first string does not start with second string
first string contains the second string at location 2
first string does not end with the second string
$ @string_matching efabcdef def
first string does not start with second string
first string contains the second string at location 5
first string ends with the second string
$ @string_matching efabcdef defx
first string does not start with second string
first string does not contain the second string at any location
first string does not end with the second string
```

Delphi (/wiki/Category:Delphi)

```
program CharacterMatching;

{$APPTYPE CONSOLE}

uses StrUtils;

begin
  WriteLn(AnsiStartsText('ab', 'abcd')); // True
  WriteLn(AnsiEndsText('zn', 'abcd')); // False
  WriteLn(AnsiContainsText('abcd', 'bb')); // False
  Writeln(AnsiContainsText('abcd', 'ab')); // True
  WriteLn(Pos('ab', 'abcd')); // 1
end.
```

Dyalect (/wiki/Category:Dyalect)

```
var value = "abcd".startsWith("ab")
value = "abcd".endsWith("zn") //returns false
value = "abab".contains("bb") //returns false
value = "abab".contains("ab") //returns true
var loc = "abab".indexOf("bb") //returns -1
loc = "abab".indexOf("ab") //returns 0
```

E (/wiki/Category:E)

```
def (http://wiki.erights.org/wiki/def) f(string1, string2) {
    println (http://wiki.erights.org/wiki/println)(string1.startsWith(string2))

    var (http://wiki.erights.org/wiki/var) index := 0
    while (http://wiki.erights.org/wiki/while) ((index := string1.startOf(string2, index)) != -1) {
        println (http://wiki.erights.org/wiki/println)(`at $index`)
        index += 1
    }

    println (http://wiki.erights.org/wiki/println)(string1.endsWith(string2))
}
```

EchoLisp (/wiki/Category:EchoLisp)

```
(string-suffix? "nette" "Antoinette") → #t
(string-prefix? "Simon" "Simon & Garfunkel") → #t

(string-match "Antoinette" "net") → #t ;; contains
(string-index "net" "Antoinette") → 5  ;; substring location
```

Elena (/wiki/Category:Elena)

ELENA 4.x :

```
import extensions;

public program()
{
    var s := "abcd";

    console.printLine(s," starts with ab: ",s.startingWith:"ab");
    console.printLine(s," starts with cd: ",s.startingWith:"cd");

    console.printLine(s," ends with ab: ",s.endingWith:"ab");
    console.printLine(s," ends with cd: ",s.endingWith:"cd");

    console.printLine(s," contains ab: ",s.containing:"ab");
    console.printLine(s," contains bc: ",s.containing:"bc");
    console.printLine(s," contains cd: ",s.containing:"cd");
    console.printLine(s," contains az: ",s.containing:"az");

    console.printLine(s," index of az: ",s.indexOf(0, "az"));
    console.printLine(s," index of cd: ",s.indexOf(0, "cd"));
    console.printLine(s," index of bc: ",s.indexOf(0, "bc"));
    console.printLine(s," index of ab: ",s.indexOf(0, "ab"));

    console.readChar()
}
```

Elixir (/wiki/Category:Elixir)

The String module has functions that cover the base requirements.

```
s1 = "abcd"
s2 = "adab"
s3 = "ab"

String.starts_with?(s1, s3)
# => true
String.starts_with?(s2, s3)
# => false

String.contains?(s1, s3)
# => true
String.contains?(s2, s3)
# => true

String.ends_with?(s1, s3)
# => false
String.ends_with?(s2, s3)
# => true


# Optional requirements:
Regex.run(~r/#{s3}/, s1, return: :index)
# => [{0, 2}]
Regex.run(~r/#{s3}/, s2, return: :index)
# => [{2, 2}]

Regex.scan(~r/#{s3}/, "abcabc", return: :index)
# => [[{0, 2}], [{3, 2}]]
```

Emacs Lisp (/wiki/Category:Emacs_Lisp)

With built-in functions

```
(defun match (word str)
  (progn

    (if (string-prefix-p word str)
        (insert (format "%s found in beginning of: %s\n" word str) )
      (insert (format "%s not found in beginning of: %s\n" word str) ))

    (setq pos (string-match word str) )

    (if pos
        (insert (format "%s found at position %d in: %s\n" word pos str) )
      (insert (format "%s not found in: %s\n" word str) ))

    (if (string-suffix-p word str)
        (insert (format "%s found in end of: %s\n" word str) )
      (insert (format "%s not found in end of: %s\n" word str) ))))

(setq string "before center after")

(progn
  (match "center" string)
  (insert "\n")
  (match "before" string)
  (insert "\n")
  (match "after" string) )
```

```
center not found in beginning of: before center after
center found at position 7 in: before center after
center not found in end of: before center after

before found in beginning of: before center after
before found at position 0 in: before center after
before not found in end of: before center after

after not found in beginning of: before center after
after found at position 14 in: before center after
after found in end of: before center after
```

With regex

```
(defun match (word str)
  (progn

    (setq regex (format "^%s.*$" word) )

    (if (string-match regex str)
        (insert (format "%s found in beginning of: %s\n" word str) )
      (insert (format "%s not found in beginning of: %s\n" word str) ))

    (setq pos (string-match word str) )

    (if pos
        (insert (format "%s found at position %d in: %s\n" word pos str) )
      (insert (format "%s not found in: %s\n" word str) ))

    (setq regex (format "^.*%s$" word) )

    (if (string-match regex str)
        (insert (format "%s found in end of: %s\n" word str) )
      (insert (format "%s not found in end of: %s\n" word str) ))))

(setq string "before center after")

(progn
  (match "center" string)
  (insert "\n")
  (match "before" string)
  (insert "\n")
  (match "after" string) )
```

Output:

```
Same output than above
```

Erlang (/wiki/Category:Erlang)

```
-module(character_matching).
-export([starts_with/2,ends_with/2,contains/2]).

%% Both starts_with and ends_with are mappings to 'lists:prefix/2' and

%% 'lists:suffix/2', respectively.


starts_with(S1,S2) ->
    lists (http://erlang.org/doc/man/lists.html):prefix(S2,S1).

ends_with(S1,S2) ->
    lists (http://erlang.org/doc/man/lists.html):suffix(S2,S1).

contains(S1,S2) ->
    contains(S1,S2,1,[]).

%% While S1 is at least as long as S2 we check if S2 is its prefix,

%% storing the result if it is. When S1 is shorter than S2, we return

%% the result. NB: this will work on any arbitrary list in erlang

%% since it makes no distinction between strings and lists.

contains([_|T]=S1,S2,N,Acc) when length(S2) =< length(S1) ->
    case starts_with(S1,S2) of
        true ->
            contains(T,S2,N+1,[N|Acc]);
        false ->
            contains(T,S2,N+1,Acc)
    end;
contains(_S1,_S2,_N,Acc) ->
    Acc.
```

Euphoria (/wiki/Category:Euphoria)

```
sequence first, second
integer x

first = "qwertyuiop"

-- Determining if the first string starts with second string
second = "qwerty"
if match(second, first) = 1 then
    printf(1, "'%s' starts with '%s'\n", {first, second})
else
    printf(1, "'%s' does not start with '%s'\n", {first, second})
end if

-- Determining if the first string contains the second string at any location
-- Print the location of the match for part 2
second = "wert"
x = match(second, first)
if x then
    printf(1, "'%s' contains '%s' at position %d\n", {first, second, x})
else
    printf(1, "'%s' does not contain '%s'\n", {first, second})
end if

-- Determining if the first string ends with the second string
second = "uio"
if length(second)<=length(first) and match_from(second, first, length(first)-length(second)+1) then
    printf(1, "'%s' ends with '%s'\n", {first, second})
else
    printf(1, "'%s' does not end with '%s'\n", {first, second})
end if
```

Output:

```
'qwertyuiop' starts with 'qwerty'
'qwertyuiop' contains 'wert' at position 2
'qwertyuiop' does not end with 'uio'
```

F# (/wiki/Category:F_Sharp)

```
[<EntryPoint>]
let main args =

    let text = "一二三四五六七八九十"
    let starts = "一二"
    let ends = "九十"
    let contains = "五六"
    let notContains = "百"

    printfn "text = %A" text
    printfn "starts with %A: %A" starts (text.StartsWith(starts))
    printfn "starts with %A: %A" ends (text.StartsWith(ends))
    printfn "ends with %A: %A" ends (text.EndsWith(ends))
    printfn "ends with %A: %A" starts (text.EndsWith(starts))
    printfn "contains %A: %A" contains (text.Contains(contains))
    printfn "contains %A: %A" notContains (text.Contains(notContains))
    printfn "substring %A begins at position %d (zero-based)" contains (text.IndexOf(contains))
    let text2 = text + text
    printfn "text = %A" text2
    Seq (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/namespaces.html).unfold (fun (n : int) ->
            let idx = text2.IndexOf(contains, n)
            if idx < 0 then None else Some (idx, idx+1)) 0
    |> Seq (http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/manual/namespaces.html).iter (printfn "substring %A begin
s at position %d (zero-based)" contains)
    0
```

Output:

```
text = "一二三四五六七八九十"
starts with "一二": true
starts with "九十": false
ends with "九十": true
ends with "一二": false
contains "五六": true
contains "百": false
substring "五六" begins at position 4 (zero-based)
text = "一二三四五六七八九十一二三四五六七八九十"
substring "五六" begins at position 4 (zero-based)
substring "五六" begins at position 14 (zero-based)
```

Factor (/wiki/Category:Factor)

Does `cheesecake` start with `cheese`?

```
"cheesecake" "cheese" head?   ! t
```

Does `cheesecake` contain `sec` at any location?

```
"sec" "cheesecake" subseq?   ! t
```

Does `cheesecake` end with `cake`?

```
"cheesecake" "cake" tail?   ! t
```

Where in `cheesecake` is the leftmost `sec`?

```
"sec" "cheesecake" subseq-start   ! 4
```

Where in `Mississippi` are all occurrences of `iss`?

```
USE: regexp
"Mississippi" "iss" <regexp> all-matching-slices [ from>> ] map   ! { 1 4 }
```

Falcon (/wiki/Category:Falcon)

'VBA/Python programmer's approach. I'm just a junior Falconeer but this code seems falconic

```
/* created by Aykayayciti Earl Lamont Montgomery
April 9th, 2018 */

s1 = "Naig Ialocin Olracnaig"
s2 = "Naig"

var = s1.startsWith(s2) ? s1 + " starts with " + s2 : s1 + " does not start with " + s2
> var

s2 = "loc"
var = s2 in s1 ? @ "$s1 contains $s2" : @ "$s1 does not contain $s2"
> var

> s1.endsWith(s2) ? @ "s1 ends with $s2" : @ "$s1 does not end with $s2"
```

Output:

```
Naig Ialocin Olracnaig starts with Naig
Naig Ialocin Olracnaig contains loc
Naig Ialocin Olracnaig does not end with loc
[Finished in 1.2s]
```

Fantom (/wiki/Category:Fantom)

Fantom provides several self-explanatory string-matching methods:

- `startsWith`
- `endsWith`
- `contains`
- `index` (takes an optional index, for the start position)
- `indexIgnoreCase` (like above, ignoring case for ASCII characters)
- `indexr` (start search from end of string, with an optional index)
- `indexrIgnoreCase` (like above, ignoring case for ASCII characters)

```
class Main
{
  public static Void main ()
  {
    string := "Fantom Language"
    echo ("String is: " + string)
    echo ("does string start with 'Fantom'? " + string.startsWith("Fantom"))
    echo ("does string start with 'Language'? " + string.startsWith("Language"))
    echo ("does string contain 'age'? " + string.contains("age"))
    echo ("does string contain 'page'? " + string.contains("page"))
    echo ("does string end with 'Fantom'? " + string.endsWith("Fantom"))
    echo ("does string end with 'Language'? " + string.endsWith("Language"))

    echo ("Location of 'age' is: " + string.index("age"))
    posn := string.index ("an")
    echo ("First location of 'an' is: " + posn)
    posn = string.index ("an", posn+1)
    echo ("Second location of 'an' is: " + posn)
    posn = string.index ("an", posn+1)
    if (posn == null) echo ("No third location of 'an'")
  }
}
```

Output:

```
String is: Fantom Language
does string start with 'Fantom'? true
does string start with 'Language'? false
does string contain 'age'? true
does string contain 'page'? false
does string end with 'Fantom'? false
does string end with 'Language'? true
Location of 'age' is: 12
First location of 'an' is: 1
Second location of 'an' is: 8
No third location of 'an'
```

FBSL (/wiki/Category:FBSL)

```
#APPTYPE CONSOLE

DIM (http://www.qbasicnews.com/qboho/qckdim.shtml) s = "roko, mat jane do"

IF LEFT(s, 4) = "roko" THEN PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) STRENC(s), " starts with ", STRENC("roko")
IF INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml)(s, "mat") THEN PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) STRENC
(s), " contains ", STRENC("mat"), " at ", INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml)
IF RIGHT(s, 2) = "do" THEN PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) STRENC(s), " ends with ", STRENC("do")
PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) STRENC(s), " contains ", STRENC("o"), " at the following locations:", InstrEx(s
, "o")

PAUSE

SUB InstrEx(mane, match)
    INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml) = 0
    WHILE INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml)(mane, match, INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml) +
 1): PRINT (http://www.qbasicnews.com/qboho/qckprint.shtml) " ", INSTR (http://www.qbasicnews.com/qboho/qckinstr.shtml);: WEND
END (http://www.qbasicnews.com/qboho/qckend.shtml) SUB
```

Output:

```
"roko, mat jane do" starts with "roko"
"roko, mat jane do" contains "mat" at 7
"roko, mat jane do" ends with "do"
"roko, mat jane do" contains "o" at the following locations: 2 4 17

Press any key to continue...
```

Forth (/wiki/Category:Forth)

```
: starts-with ( a l a2 l2 -- ? )
  tuck 2>r min 2r> compare 0= ;
: ends-with ( a l a2 l2 -- ? )
  tuck 2>r negate over + 0 max /string 2r> compare 0= ;
\ use SEARCH ( a l a2 l2 -- a3 l3 ? ) for contains
```

Fortran (/wiki/Category:Fortran)

Fortran does not offer a string type, but since F77 it has been possible to use a CHARACTER variable, of some specified size, whose size may be accessed via the LEN function. When passed as a parameter, a secret additional parameter specifies its size and so string-like usage is possible. Character matching is case sensitive, and, trailing spaces are ignored so that "xx" and "xx " are deemed equal. The function INDEX(text,target) determines the first index in *text* where *target* matches, and returns zero if there is no such match. Unfortunately, the function does not allow the specification of a starting position for a search, as to find any second and further matches. One must specify something like `INDEX(text(5:),target)` to start with position five, and then deal with the resulting offsets needed to relate the result to positions within the parameter. On the other hand, since there is no "length" conjoined to the text such substring selections can be made without copying the text to a work area, unlike the `copy(text,start,stop)` equivalent of Pascal for example. Some Fortran compilers *do* offer a starting point, and also an option to search backwards from the end, but these facilities are not guaranteed. Similarly, INDEX is only made available for CHARACTER searching, even though it could easily be generalised to other types.

A second problem is presented by the possibility that a logical expression such as `L.LT.0 .OR. etc.` will always or might possibly or in certain constructions but not others be fully evaluated, which is to say that the *etc* will be evaluated even though L < 0 is *true* so that the result is determined. And in this case, evaluating the *etc* will cause trouble because the indexing won't work! To be safe, therefore, a rather lame two-stage test is required - though optimising compilers might well shift code around anyway.

In the case of STARTS, these annoyances can be left to the INDEX function rather than comparing the start of A against B. At the cost of it searching the whole of A if B is n at the start. Otherwise, it would be the mirror of ENDS.

```
      SUBROUTINE STARTS(A,B)    !Text A starts with text B?
       CHARACTER*(*) A,B
        IF (INDEX(A,B).EQ.1) THEN       !Searches A to find B.
          WRITE (6,*) ">",A,"< starts with >",B,"<"
         ELSE
          WRITE (6,*) ">",A,"< does not start with >",B,"<"
        END IF
      END SUBROUTINE STARTS

      SUBROUTINE HAS(A,B)       !Text B appears somewhere in text A?
       CHARACTER*(*) A,B
       INTEGER L
        L = INDEX(A,B)          !The first position in A where B matches.
        IF (L.LE.0) THEN
          WRITE (6,*) ">",A,"< does not contain >",B,"<"
         ELSE
          WRITE (6,*) ">",A,"< contains a >",B,"<, offset",L
        END IF
      END SUBROUTINE HAS

      SUBROUTINE ENDS(A,B)      !Text A ends with text B.
       CHARACTER*(*) A,B
       INTEGER L
        L = LEN(A) - LEN(B)     !Find the tail end of A that B might match.
        IF (L.LT.0) THEN        !Dare not use an OR, because of full evaluation risks.
          WRITE (6,*) ">",A,"< is too short to end with >",B,"<"        !Might as well have a special message.
        ELSE IF (A(L + 1:L + LEN(B)).NE.B) THEN !Otherwise, it is safe to look.
          WRITE (6,*) ">",A,"< does not end with >",B,"<"
        ELSE
          WRITE (6,*) ">",A,"< ends with >",B,"<"
        END IF
      END SUBROUTINE ENDS

      CALL STARTS("This","is")
      CALL STARTS("Theory","The")
      CALL HAS("Bananas","an")
      CALL ENDS("Banana","an")
      CALL ENDS("Banana","na")
      CALL ENDS("Brief","Much longer")
      END
```

Output: text strings are bounded by >*etc.*< in case of leading or trailing spaces.

```
>This< does not start with >is<
>Theory< starts with >The<
>Bananas< contains a >an<, offset           2
>Banana< does not end with >an<
>Banana< ends with >na<
>Brief< is too short to end with >Much longer<
```

Similar program using modern Fortran style

```
!-----------------------------------------------------------------------
!Main program string_matching
!-----------------------------------------------------------------------
program    string_matching
   implicit none
   character(len=*), parameter :: fmt= '(I0)'
   write(*,fmt) starts("this","is")
   write(*,fmt) starts("theory","the")
   write(*,fmt) has("bananas","an")
   write(*,fmt) ends("banana","an")
   write(*,fmt) ends("banana","na")
   write(*,fmt) ends("brief","much longer")

 contains
   !     Determining if the first string starts with second string
   function  starts(string1, string2) result(answer)
      implicit none
      character(len=*), intent(in) :: string1
      character(len=*), intent(in) :: string2
      integer :: answer
      answer = 0
      if(len(string2)>len(string1)) return
      if(string1(1:len(string2))==string2) answer = 1
   end function starts
   !     Determining if the first string contains the second string at any location
   function  has(string1, string2) result(answer)
      implicit none
      character(len=*), intent(in) :: string1
      character(len=*), intent(in) :: string2
      character(len=:),allocatable :: temp
      integer :: answer, add
      character(len=*), parameter :: fmt= '(A6,X,I0)'
      answer = 0
      add = 0
      if(len(string2)>len(string1)) return
      answer = index(string1, string2)
      if(answer==0) return
      !     Print the location of the match for part 2
      write(*,fmt) " at ", answer
      !     Handle multiple occurrences of a string for part 2.
      add = answer
      temp = string1(answer+1:)
      do while(answer>0)
         answer = index(temp, string2)
         add = add + answer
         if(answer>0) write(*,fmt) " at ", add
         !          deallocate(temp)
         temp = string1(add+1:) ! auto reallocation
      enddo
      answer = 1
   end function has
   !     Determining if the first string ends with the second string
   function  ends(string1, string2) result(answer)
      implicit none
      character(len=*), intent(in) :: string1
      character(len=*), intent(in) :: string2
      integer :: answer
      answer = 0
      if(len(string2)>len(string1)) return
      if(string1(len(string1)-len(string2)+1:)==string2) answer = 1
   end function ends
end program string_matching
```

Output: false = 0, true = 1 (+ multiple occurrences if applicable)

```
0
1
   at  2
   at  4
1
0
1
0
```

In recent standards of Fortran strings as intrinsic first-class type and many intrinsic procedures for strings manipulation have been added.

FreeBASIC (/wiki/Category:FreeBASIC)

```
' FB 1.05.0 Win64

Dim As String s1 = "abracadabra"
Dim As String s2 = "abra"
Print "First string  : "; s1
Print "Second string : "; s2
Print
Print "First string begins with second string : "; CBool(s2 = Left(s1, Len(s2)))
Dim As Integer i1 = Instr(s1, s2)
Dim As Integer i2
Print "First string contains second string    : ";
If i1 Then
  Print "at index"; i1;
  i2 = Instr(i1 + Len(s2), s1, s2)
  If i2 Then
    Print " and at index"; i2
  Else
    Print
  End If
Else
  Print "false";
End If
Print "First string ends with second string   : "; CBool(s2 = Right(s1, Len(s2)))
Print
Print "Press any key to quit"
Sleep
```

Output:

```
First string  : abracadabra
Second string : abra

First string begins with second string : true
First string contains second string    : at index 1 and at index 8
First string ends with second string   : true
```

Gambas (/wiki/Category:Gambas)

Click this link to run this code (https://gambas-playground.proko.eu/?gist=07bb32f4e8e8f7d81898cf41d4431a2e)

```
Public (http://gambasdoc.org/help/lang/public) Sub (http://gambasdoc.org/help/lang/sub) Main()
Dim (http://gambasdoc.org/help/lang/dim) sString1 As (http://gambasdoc.org/help/lang/as) String (http://gambasdoc.org/help/lang/type/string) = "Hello world"
Dim (http://gambasdoc.org/help/lang/dim) sString2 As (http://gambasdoc.org/help/lang/as) String (http://gambasdoc.org/help/lang/type/string) = "Hello"

Print (http://gambasdoc.org/help/lang/print) sString1 Begins Left (http://gambasdoc.org/help/lang/left)(sString2, 5)
          'Determine if the first string starts with second string
If (http://gambasdoc.org/help/lang/if) InStr (http://gambasdoc.org/help/lang/instr)(sString1, sString2) Then (http://gambasdoc.org/help/lang/then) Print (http://gambasdoc.org/help/lang/print) "True" Else (http://gambasdoc.org/help/lang/else) Print (http://gambasdoc.org/help/lang/print) "False"   'Determine if the first string contains the second string at any location
Print (http://gambasdoc.org/help/lang/print) sString1 Ends Left (http://gambasdoc.org/help/lang/left)(sString2, 5)
          'Determine if the first string ends with the second string

End (http://gambasdoc.org/help/lang/end)
```

Output:

```
True
True
False
```

GML (/wiki/Category:GML)

Translation of: BASIC

```
#define charMatch
{
    first = "qwertyuiop";

    // Determining if the first string starts with second string
    second = "qwerty";
    if (string_pos(second, first) > 0) {
        show_message("'" + first + "' starts with '" + second + "'");
    } else {
        show_message("'" + first + "' does not start with '" + second + "'");
    }

    second = "wert"
    // Determining if the first string contains the second string at any location
    // Print the location of the match for part 2
    if (string_pos(second, first) > 0) {
        show_message("'" + first + "' contains '" + second + "' at position " + string(x));
    } else {
        show_message("'" + first + "' does not contain '" + second + "'");
    }
    // Handle multiple occurrences of a string for part 2.
    x = string_count(second, first);
    show_message("'" + first + "' contains " + string(x) + " instances of '" + second + "'");

// Determining if the first string ends with the second string
    second = "random garbage"
    temp = string_copy(first,
                       (string_length(first) - string_length(second)) + 1,
                       string_length(second));
    if (temp == second) {
        show_message("'" + first + "' ends with '" + second + "'");
    } else {
        show_message("'" + first + "' does not end with '" + second + "'");
    }
}
```

Output:

(in message boxes, 1 per line):

```
'qwertyuiop' starts with 'qwerty'
'qwertyuiop' contains 'wert' at position 0
'qwertyuiop' contains 1 instances of 'wert'
'qwertyuiop' does not end with 'random garbage'
```

Go (/wiki/Category:Go)

```
package main

import (
    "fmt"
    "strings"
)

func match(first, second string) {
    fmt.Printf("1. %s starts with %s: %t\n",
        first, second, strings.HasPrefix(first, second))
    i := strings.Index(first, second)
    fmt.Printf("2. %s contains %s: %t,\n", first, second, i >= 0)
    if i >= 0 {
        fmt.Printf("2.1. at location %d,\n", i)
        for start := i+1;; {
            if i = strings.Index(first[start:], second); i < 0 {
                break
            }
            fmt.Printf("2.2. at location %d,\n", start+i)
            start += i+1
        }
        fmt.Println("2.2. and that's all")
    }
    fmt.Printf("3. %s ends with %s: %t\n",
        first, second, strings.HasSuffix(first, second))
}

func main() {
    match("abracadabra", "abr")
}
```

Output:

```
1. abracadabra starts with abr: true
2. abracadabra contains abr: true,
2.1. at location 0,
2.2. at location 7,
2.2. and that's all
3. abracadabra ends with abr: false
```

Groovy (/wiki/Category:Groovy)

Examples:

```
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) "abcd".startsWith("ab")
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) ! "abcd".startsWith("zn")
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) "abcd".endsWith("cd")
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) ! "abcd".endsWith("zn")
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) "abab".contains (https://www.google.de/search?q=site%3Agr
oovy.codehaus.org/%20contains)("ba")
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) ! "abab".contains (https://www.google.de/search?q=site%3A
groovy.codehaus.org/%20contains)("bb")


assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) "abab".indexOf("bb") == -1 // not found flag
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) "abab".indexOf("ab") == 0

def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) indicesOf = { string, substring ->
    if (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20if) (!string) { return (https://www.google.de/search?q=site%3Agro
ovy.codehaus.org/%20return) [] }
    def (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20def) indices = [-1]
    while (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20while) (true (https://www.google.de/search?q=site%3Agroovy.cod
ehaus.org/%20true)) {
        indices << string.indexOf(substring, indices.last()+1)
        if (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20if) (indices.last() == -1) break (https://www.google.de/searc
h?q=site%3Agroovy.codehaus.org/%20break)
    }
    indices[1..<(indices.size (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20size)()-1)]
}
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) indicesOf("abab", "ab") == [0, 2]
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) indicesOf("abab", "ba") == [1]
assert (https://www.google.de/search?q=site%3Agroovy.codehaus.org/%20assert) indicesOf("abab", "xy") == []
```

All assertions pass, so there is no output.

Haskell (/wiki/Category:Haskell)

```
> import Data.List
> "abc" `isPrefixOf` "abcdefg"
True
> "efg" `isSuffixOf` "abcdefg"
True
> "bcd" `isInfixOf` "abcdefg"
True
> "abc" `isInfixOf` "abcdefg" -- Prefixes and suffixes are also infixes
True
> let infixes a b = findIndices (isPrefixOf a) $ tails b
> infixes "ab" "abcdefabqqab"
[0,6,10]
```

Icon (/wiki/Category:Icon) and Unicon (/wiki/Category:Unicon)

```
procedure main()

   write("Matching s2 :=",image(s2 := "ab")," within s1:= ",image(s1 := "abcdabab"))

   write("Test #1 beginning ",if match(s2,s1) then "matches " else "failed")
   writes("Test #2 all matches at positions [")
      every writes(" ",find(s2,s1)|"]\n")
   write("Test #3 ending ", if s1[0-:*s2] == s2 then "matches" else "fails")

end
```

Output:

```
Matching s2 :="ab" within s1:= "abcdabab"
Test #1 beginning matches
Test #2 all matches at positions [ 1 5 7 ]
Test #3 ending matches
```

J (/wiki/Category:J)

```
startswith=: ] -: ({.~ #)
contains=: +./@:E.~
endswith=: ] -: (-@# {.~ -@#)
```

Example use:

```
   'abcd' startswith 'ab'
1
   'abcd' startswith 'cd'
0
   'abcd' endswith 'ab'
0
   'abcd' endswith 'cd'
1
   'abcd' contains 'bb'
0
   'abcd' contains 'ab'
1
   'abcd' contains 'bc'
1
   'abab' contains 'ab'
1
   'abab' I.@E.~ 'ab'       NB. find starting indicies
0 2
```

Note that these verbs contain no constraints restricting them to sequences of characters and so also apply to arrays of type other than character:

```
   0 1 2 3 startswith 0 1              NB. integer
1
   4.2 5.1 1.3 9 3 contains 1.3 4.2    NB. floating point
0
   4.2 5.1 1.3 4.2 9 3 contains 1.3 4.2
1
```

Java (/wiki/Category:Java)

```
"abcd".startsWith("ab") //returns true
"abcd".endsWith("zn") //returns false
"abab".contains("bb") //returns false
"abab".contains("ab") //returns true
int loc = "abab".indexOf("bb") //returns -1
loc = "abab".indexOf("ab") //returns 0
loc = "abab".indexOf("ab",loc+1) //returns 2
```

// --// public class JavaApplication6 {

```
   public static void main(String[] args) {
       String strOne = "complexity";
       String strTwo = "udacity";
```

```
       //
       stringMatch(strOne, strTwo);
```

```
   }
```

```
   public static void stringMatch(String one, String two) {
       boolean match = false;
       if (one.charAt(0) == two.charAt(0)) {
           System.out.println(match = true);   // returns true
       } else {
           System.out.println(match);       // returns false
       }
       for (int i = 0; i < two.length(); i++) {
           int temp = i;
           for (int x = 0; x < one.length(); x++) {
               if (two.charAt(temp) == one.charAt(x)) {
                   System.out.println(match = true);    //returns true
                   i = two.length();
               }
           }
       }
       int num1 = one.length() - 1;
       int num2 = two.length() - 1;
       if (one.charAt(num1) == two.charAt(num2)) {
           System.out.println(match = true);
       } else {
           System.out.println(match = false);
       }
   }
```

}

JavaScript (/wiki/Category:JavaScript)

```
var stringA = "tacoloco"
  , stringB = "co"
  , q1, q2, q2multi, m
  , q2matches = []

// stringA starts with stringB
q1 = stringA.substring(0, stringB.length) == stringB

// stringA contains stringB
q2  = stringA.indexOf(stringB)

// multiple matches
q2multi = new RegExp(stringB,'g')

while(m = q2multi.exec(stringA)){
        q2matches.push(m.index)
}

// stringA ends with stringB
q3 = stringA.substr(-stringB.length) == stringB

console.log("1: Does '"+stringA+"' start with '"+stringB+"'? " + ( q1 ? "Yes." : "No."))
console.log("2: Is '"+stringB+"' contained in '"+stringA+"'? " + (~q2 ? "Yes, at index "+q2+"." : "No."))
if (~q2 && q2matches.length > 1){
        console.log("   In fact, it happens "+q2matches.length+" times within '"+stringA+"', at index"+(q2matches.length > 1 ? "es" :
"")+" "+q2matches.join(', ')+".")
}
console.log("3: Does '"+stringA+"' end with '"+stringB+"'? "   + ( q3 ? "Yes." : "No."))
```

Output:

```
1: Does 'tacoloco' start with 'co'? No.
2: Is 'co' contained in 'tacoloco'? Yes, at index 2.
   In fact, it happens 2 times within 'tacoloco', at indexes 2, 6.
3: Does 'tacoloco' end with 'co'? Yes.
```

jq (/wiki/Category:Jq)

Using the builtins of jq 1.4 and later:

```
# startswith/1 is boolean:
"abc" | startswith("ab")
#=> true
```

```
# index/1 returns the index or null,
# so the jq test "if index(_) then ...." can be used
# without any type conversion.

"abcd" | index( "bc")
#=> 1
```

```
# endswith/1 is also boolean:
"abc" | endswith("bc")
#=> true
```

Using the regex functions available in jq 1.5:

```
"abc" | test( "^ab")

"abcd" | test("bc")

"abcd" | test("cd$")
```

Multiple Occurrences

To determine all the indices of one string in another:

```
# In jq 1.4 or later:
jq -n '"abcdabcd" | indices("bc")'
[
  1,
  5
]
```

In jq 1.5, the regex function match/1 can also be used:

```
$ jq -n '"abcdabcd" | match("bc"; "g") | .offset'
1
5
```

Julia (/wiki/Category:Julia)

```
startswith("abcd","ab")            #returns true
findfirst("ab", "abcd")            #returns 1:2, indices range where string was found
endswith("abcd","zn")              #returns false
match(r"ab","abcd") != Nothing     #returns true where 1st arg is regex string
for r in eachmatch(r"ab","abab")
        println(r.offset)
end                                #returns 1, then 3 matching the two starting indices where the substring was found
```

K (/wiki/Category:K)

```
startswith: {:[0<#p:_ss[x;y];~*p;0]}
endswith: {0=(-#y)+(#x)-*_ss[x;y]}
contains: {0<#_ss[x;y]}
```

Example:

```
  startswith["abcd";"ab"]
1
  startswith["abcd";"bc"]
0
  endswith["abcd";"cd"]
1
  endswith["abcd";"bc"]
0
  contains["abcdef";"cde"]
1
  contains["abcdef";"bdef"]
0
  _ss["abcdabceabc";"abc"]    / location of matches
0 4 8
```

Kotlin (/wiki/Category:Kotlin)

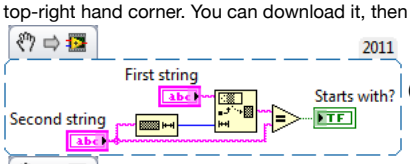
```
// version 1.0.6

fun main(args: Array<String>) {
    val (https://scala-lang.org) s1 = "abracadabra"
    val (https://scala-lang.org) s2 = "abra"
    println("$s1 begins with $s2 : ${s1.startsWith(s2)}")
    println("$s1 ends with $s2   : ${s1.endsWith(s2)}")
    val (https://scala-lang.org) b  = s2 in s1
    print("$s1 contains $s2    : $b")
    if (https://scala-lang.org) (b) println(" at locations ${s1.indexOf(s2) + 1} and ${s1.lastIndexOf(s2) + 1}")
    else (https://scala-lang.org) println()
}
```

Output:

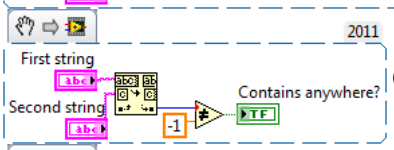
```
abracadabra begins with abra : true
abracadabra ends with abra   : true
abracadabra contains abra    : true at locations 1 and 8
```

LabVIEW (/wiki/Category:LabVIEW)

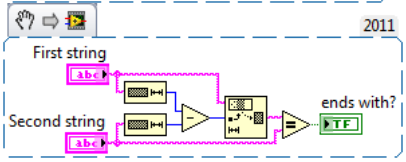
These images solve the task's requirements in order.
This image is a VI Snippet (http://zone.ni.com/devzone/cda/tut/p/id/9330), an executable image of LabVIEW (/wiki/LabVIEW) code. The LabVIEW version is shown on the top-right hand corner. You can download it, then drag-and-drop it onto the LabVIEW block diagram from a file browser, and it will appear as runnable, editable code.



(/wiki/File:LabVIEW_Character_matching_1.png)



(/wiki/File:LabVIEW_Character_matching_2.png)



(/wiki/File:LabVIEW_Character_matching_3.png)

Lang5 (/wiki/Category:Lang5)

```
: 2array  2 compress ; : bi*  '_ set dip _ execute ;  : bi@  dup bi* ;
: comb  "" split ;  : concat  "" join ;  : dip  swap '_ set execute _ ;
: first  0 extract swap drop ; : flip  comb reverse concat ;

: contains
    swap 'comb bi@ length do                    # create a matrix.
         1 - "dup 1 1 compress rotate" dip dup 0 == if break then
    loop drop length compress swap drop
    "length 1 -" bi@ rot 0 0 "'2array dip" '2array bi* swap 2array slice reverse
    : concat.(*)  concat ;
    'concat "'concat. apply" bi* eq 1 1 compress index collapse
    length if expand drop else drop 0 then ;    # r: position.
: end-with    'flip bi@ start-with ;
: start-with  'comb bi@ length rot swap iota subscript 'concat bi@ eq ;

"rosettacode" "rosetta" start-with .    # 1
"rosettacode" "taco" contains .         # 5
"rosettacode" "ocat" contains .         # 0
"rosettacode" "edoc" end-with .         # 0
"rosettacode" "code" contains .         # 7
```

Lasso (/wiki/Category:Lasso)

```
local(
 a = 'a quick brown peanut jumped over a quick brown fox',
 b = 'a quick brown'
)

//Determining if the first string starts with second string
#a->beginswith(#b) // true

//Determining if the first string contains the second string at any location
#a >> #b           // true
#a->contains(#b)   // true

//Determining if the first string ends with the second string
#a->endswith(#b)   // false
```

Liberty BASIC (/wiki/Category:Liberty_BASIC)

```
'1---Determining if the first string starts with second string
st1$="first string"
st2$="first"
if left$(st1$,len(st2$))=st2$ then
    print "First string starts with second string."
end if

'2---Determining if the first string contains the second string at any location
'2.1---Print the location of the match for part 2
st1$="Mississippi"
st2$="i"
if instr(st1$,st2$) then
    print "First string contains second string."
    print "Second string is at location ";instr(st1$,st2$)
end if

'2.2---Handle multiple occurrences of a string for part 2.
pos=instr(st1$,st2$)
while pos
    count=count+1
    pos=instr(st1$,st2$,pos+len(st2$))
wend
print "Number of times second string appears in first string is ";count

'3---Determining if the first string ends with the second string
st1$="first string"
st2$="string"
if right$(st1$,len(st2$))=st2$ then
    print "First string ends with second string."
end if
```

Lingo (/wiki/Category:Lingo)

```
a = "Hello world!"
b = "Hello"

-- Determining if the first string starts with second string
put a starts b
-- 1

-- Determining if the first string contains the second string at any location
put a contains b
-- 1

-- Determining if the first string ends with the second string
put a.char[a.length-b.length+1..a.length] = b
-- 0

b = "world!"
put a.char[a.length-b.length+1..a.length] = b
-- 1

-- Print the location of the match for part 2
put offset(b, a)
-- 7
```

Logo (/wiki/Category:Logo)

```
to starts.with? :sub :thing
  if empty? :sub [output "true]
  if empty? :thing [output "false]
  if not equal? first :sub first :thing [output "false]
  output starts.with? butfirst :sub butfirst :thing
end

to ends.with? :sub :thing
  if empty? :sub [output "true]
  if empty? :thing [output "false]
  if not equal? last :sub last :thing [output "false]
  output ends.with? butlast :sub butlast :thing
end

show starts.with? "dog "doghouse    ; true
show ends.with? "house "doghouse    ; true
show substring? "gho "doghouse       ; true  (built-in)
```

Lua (/wiki/Category:Lua)

```
s1 = "string"
s2 = "str"
s3 = "ing"
s4 = "xyz"

print( "s1 starts with s2: ", string.find( s1, s2 ) == 1 )
print( "s1 starts with s3: ", string.find( s1, s3 ) == 1, "\n" )

print( "s1 contains s3: ", string.find( s1, s3 ) ~= nil )
print( "s1 contains s3: ", string.find( s1, s4 ) ~= nil, "\n" )

print( "s1 ends with s2: ", select( 2, string.find( s1, s2 ) ) == string.len( s1 ) )
print( "s1 ends with s3: ", select( 2, string.find( s1, s3 ) ) == string.len( s1 ) )
```

M2000 Interpreter (/wiki/Category:M2000_Interpreter)

```


Module StringMatch {
      A$="Hello World"
      Print A$ ~ "Hello*"
      Print A$ ~ "*llo*"
      p=Instr(A$, "llo")
      Print p=3
      \\ Handle multiple occurance for "o"
      p=Instr(A$, "o")
      While p > 0 {
            Print "position:";p;{ for "o"}
            p=Instr(A$, "o", p+1)
      }
      Print A$ ~ "*orld"
}
{{out}}
<pre>
    True
    True
    True
position:5 for "o"
position:8 for "o"
    True
</pre>
StringMatch


```

Maple (/wiki/Category:Maple)

These facilities are all to be found in the StringTools package in Maple.

```
> with( StringTools ): # bind package exports at the top-level
> s := "dzrIemaWWIMidXYZwGiqkOOn":
> s[1..4]; # pick a prefix
                                  "dzrI"

> IsPrefix( s[ 1 .. 4 ], s ); # check it
                                   true

> s[ -4 .. -1 ]; # pick a suffix
                                  "kOOn"

> IsSuffix( s[ -4 .. -1 ], s ); # check it
                                   true

> p := Search( "XYZ", s ); # find a substring
                                 p := 14

> s[ p .. p + 2 ]; # check
                                  "XYZ"

> SearchAll( [ "WWI", "XYZ" ], s ); # search for multiple patterns
                             [8, 1], [14, 2]

> to 3 do s := cat( s, s ) end: length( s ); # build a longer string by repeated doubling
                                   192

> p := SearchAll( "XYZ", s ); # find all occurrences
                p := 14, 38, 62, 86, 110, 134, 158, 182

> {seq}( s[ i .. i + 2 ], i = p ); # check them
                                 {"XYZ"}
```

The StringTools package also contains facilities for regular expression matching, but for fixed string patterns, the Search and SearchAll tools are much faster.

Mathematica (/wiki/Category:Mathematica)

```
StartWith[x_, y_] := MemberQ[Flatten[StringPosition[x, y]], 1]
EndWith[x_, y_] :=  MemberQ[Flatten[StringPosition[x, y]], StringLength[x]]
StartWith["XYZaaabXYZaaaaXYZXYZ", "XYZ"]
EndWith["XYZaaabXYZaaaaXYZXYZ", "XYZ"]
StringPosition["XYZaaabXYZaaaaXYZXYZ", "XYZ"]
```

Output:

```
True
True
{{1,3},{8,10},{15,17},{18,20}}
```

MATLAB (/wiki/Category:MATLAB) / Octave (/wiki/Category:Octave)

```
   % 1. Determining if the first string starts with second string
        strcmp (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/strcmp.html)(str1,str2,length (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/length.html)(str2))
   % 2. Determining if the first string contains the second string at any location
        ~isempty(strfind(s1,s2))
   % 3. Determining if the first string ends with the second string
        ( (length (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/length.html)(str1)>=length (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/length.html)(str2)) && strcmp (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/strcmp.html)(str1(end+[1-length (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/length.html)(str2):0]),str2) )

   % 1. Print the location of the match for part 2
        disp (https://www.mathworks.com/access/helpdesk/help/techdoc/ref/disp.html)(strfind(s1,s2))
   % 2. Handle multiple occurrences of a string for part 2.
        ix = strfind(s1,s2);   % ix is a vector containing the starting positions of s2 within s1
```

min (/wiki/Category:Min)

One way might be:

Works with: min (/wiki/Min) version 0.19.6

```
(indexof 0 ==) :starts-with?
(indexof -1 !=) :contains?
((/ $/) swap 1 insert "" join regex ("") !=) :ends-with?

"minimalistic" "min" starts-with? puts!
"minimalistic" "list" contains? puts!
"minimalistic" "list" ends-with? puts!
```

Output:

```
true
true
false
```

MiniScript (/wiki/Category:MiniScript)

We first extend the built-in string class with three new methods, and then demonstrate their use on some sample strings.

```
string.startsWith = function(s)
    return self.len >= s.len and s[:s.len] == s
end function

string.endsWith = function(s)
    return self.len >= s.len and s[-s.len:] == s
end function

string.findAll = function(s)
    result = []
    after = null
    while true
        foundPos = self.indexOf(s, after)
        if foundPos == null then return result
        result.push foundPos
        after = foundPos + s.len - 1
    end while
end function

first = "The brown dog jumped jumped and jumped"
second = "jumped"

firstQ = """" + first + """"  // (first string, in quotes)
secondQ = """" + second + """"
doesOrNot = [" does not ", " does "]

print firstQ + doesOrNot[first.startsWith(second)] + "start with " + secondQ
print

found = first.findAll(second)
if not found then
    print firstQ + " does not contain " + secondQ + " anywhere"
else
    for pos in found
        print firstQ + " is found at position " + pos + " in " + secondQ
    end for
end if
print

print firstQ + doesOrNot[first.endsWith(second)] + "end with " + secondQ
```

Output:

```
"The brown dog jumped jumped and jumped" does start with "jumped"

"The brown dog jumped jumped and jumped" is found at position 14 in "jumped"
"The brown dog jumped jumped and jumped" is found at position 21 in "jumped"
"The brown dog jumped jumped and jumped" is found at position 32 in "jumped"

"The brown dog jumped jumped and jumped" does end with "jumped"
```

NetRexx (/wiki/Category:NetRexx)

```
/* NetRexx */
options replace format comments java crossref savelog symbols

lipsum = ''
x_ = 0
x_ = x_ + 1; lipsum[0] = x_; lipsum[x_] = 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'
x_ = x_ + 1; lipsum[0] = x_; lipsum[x_] = lipsum[1].reverse

srch = ''
srch[1] = 'Lorem ipsum dolor sit amet'
srch[2] = 'consectetur adipisicing elit'
srch[3] = 'dolore magna aliqua.'

loop j_ = 1 to lipsum[0]
  x1 = lipsum[j_].pos(srch[1])
  x2 = lipsum[j_].pos(srch[2])
  x3 = lipsum[j_].lastpos(srch[3])

  report(x1 = 1, lipsum[j_], srch[1], 'Test string starts with search string:', 'Test string does not start with search string:')
  report(x2 > 0, lipsum[j_], srch[2], 'Search string located in test string at position:' x2,  'Search string not found within test st
ring:')
  report(x3 \= srch[3].length, lipsum[j_], srch[3], 'Test string ends with search string:', 'Test string does not start with search st
ring:')
  end j_

many = ''
x_ = 0
x_ = x_ + 1; many[0] = x_; many[x_] = 'How many times does "many times" occur in this string?'
x_ = x_ + 1; many[0] = x_; many[x_] = 'How often does "many times" occur in this string?'
x_ = x_ + 1; many[0] = x_; many[x_] = 'How often does it occur in this string?'
srch[4] = 'many times'

loop j_ = 1 to many[0]
  o_ = 0
  k_ = 0
  loop label dups until o_ = 0
    o_ = many[j_].pos(srch[4], o_ + 1)
    if o_ \= 0 then k_ = k_ + 1
    end dups
  report(k_ > 0, many[j_], srch[4], 'Number of times search string occurs:' k_, 'Search string not found')
  end j_

method report(state = boolean, ts, ss, testSuccess, testFailure) public static
  if state then say testSuccess
  else          say testFailure
  say '    Test string:' ts
  say '  Search string:' ss
  say

  return
```

NewLISP (/wiki/Category:NewLISP)

```
(setq (http://www.newlisp.org/downloads/newlisp_manual.html#setq) str "abcdefbcghi")

;; test if str starts with "ab"
(starts-with (http://www.newlisp.org/downloads/newlisp_manual.html#starts-with) str "ab")

;; find "bc" inside str
(find (http://www.newlisp.org/downloads/newlisp_manual.html#find) "bc" str)

;; test if str ends with "ghi"
(ends-with (http://www.newlisp.org/downloads/newlisp_manual.html#ends-with) str "ghi")

;; find all positions of pattern inside str
(define (http://www.newlisp.org/downloads/newlisp_manual.html#define) (find-all-pos pattern str)
  (let (http://www.newlisp.org/downloads/newlisp_manual.html#let) ((idx -1) (pos '()))
    (while (http://www.newlisp.org/downloads/newlisp_manual.html#while) (setq (http://www.newlisp.org/downloads/newlisp_manual.html#se
tq) idx (find (http://www.newlisp.org/downloads/newlisp_manual.html#find) pattern str 0 (+ idx 1)))
      (push (http://www.newlisp.org/downloads/newlisp_manual.html#push) idx pos -1))))

(find-all-pos "bc" str)
```

Nim (/wiki/Category:Nim)

```
import strutils

let s = "The quick brown fox"
if s.startsWith("The quick"):
  echo "Starts with “The quick”."
if s.endsWith("brown Fox"):
  echo "Ends with “brown fox”."
if s.contains(" brown "):
  echo "Contains “ brown ”."
if "quick" in s:
  echo "Contains “quick”."      # Alternate form for "contains".

let pos = find(s, " brown ")    # −1 if not found.
if pos >= 0:
  echo "“ brown ” is located at position: " & $pos
```

Output:

```
Starts with “The quick”.
Contains “ brown ”.
Contains “quick”.
“ brown ” is located at position: 9
```

Objeck (/wiki/Category:Objeck)

```
bundle Default {
  class Matching {
    function : Main(args : System.String[]) ~ Nil {
      "abcd"->StartsWith("ab")->PrintLine(); # returns true
      "abcd"->EndsWith("zn")->PrintLine(); # returns false
      ("abab"->Find("bb") <> -1)->PrintLine(); # returns false
      ("abab"->Find("ab") <> -1)->PrintLine(); # returns true
      loc := "abab"->Find("bb"); # returns -1
      loc := "abab"->Find("ab"); # returns 0
      loc := "abab"->Find("ab", loc + 1); # returns 2
    }
  }
}
```

Objective-C (/wiki/Category:Objective-C)

```
[@"abcd" hasPrefix:@"ab"] //returns true
[@"abcd" hasSuffix:@"zn"] //returns false
int loc = [@"abab" rangeOfString:@"bb"].location //returns -1
loc = [@"abab" rangeOfString:@"ab"].location //returns 0
loc = [@"abab" rangeOfString:@"ab" options:0 range:NSMakeRange(loc+1, [@"abab" length]-(loc+1))].location //returns 2
```

OCaml (/wiki/Category:OCaml)

```
let match1 s1 s2 =
  let len1 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s1
  and len2 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s2 in
  if len1 < len2 then false else
    let sub = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).sub s1 0 len2 in
    (sub = s2)
```

testing in the top-level:

```
# match1 "Hello" "Hello World!" ;;
- : bool = false
# match1 "Hello World!" "Hello" ;;
- : bool = true
```

```
let match2 s1 s2 =
  let len1 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s1
  and len2 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s2 in
  if len1 < len2 then false else
    let rec aux i =
      if i < 0 then false else
        let sub = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).sub s1 i len2 in
        if (sub = s2) then true else aux (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) i)
    in
    aux (len1 - len2)
```

```
# match2 "It's raining, Hello World!" "umbrella" ;;
- : bool = false
# match2 "It's raining, Hello World!" "Hello" ;;
- : bool = true
```

```
let match3 s1 s2 =
  let len1 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s1
  and len2 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s2 in
  if len1 < len2 then false else
    let sub = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).sub s1 (len1 - len2) len2 in
    (sub = s2)
```

```
# match3 "Hello World" "Hello" ;;
- : bool = false
# match3 "Hello World" "World" ;;
- : bool = true
```

```
let match2_loc s1 s2 =
  let len1 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s1
  and len2 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s2 in
  if len1 < len2 then (false, -1) else
    let rec aux i =
      if i < 0 then (false, -1) else
        let sub = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).sub s1 i len2 in
        if (sub = s2) then (true, i) else aux (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) i)
    in
    aux (len1 - len2)
```

```
# match2_loc "The sun's shining, Hello World!" "raining" ;;
- : bool * int = (false, -1)
# match2_loc "The sun's shining, Hello World!" "shining" ;;
- : bool * int = (true, 10)
```

```
let match2_num s1 s2 =
  let len1 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s1
  and len2 = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).length s2 in
  if len1 < len2 then (false, 0) else
    let rec aux i n =
      if i < 0 then (n <> 0, n) else
        let sub = String (http://caml.inria.fr/pub/docs/manual-ocaml/libref/String.html).sub s1 i len2 in
        if (sub = s2)
        then aux (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) i) (succ (http://caml.inria.fr/pub/
docs/manual-ocaml/libref/Pervasives.html#VALsucc) n)
        else aux (pred (http://caml.inria.fr/pub/docs/manual-ocaml/libref/Pervasives.html#VALpred) i) (n)
    in
    aux (len1 - len2) 0
```

```
# match2_num "This cloud looks like a camel, \
    that other cloud looks like a llama" "stone" ;;
- : bool * int = (false, 0)
# match2_num "This cloud looks like a camel, \
    that other cloud looks like a llama" "cloud" ;;
- : bool * int = (true, 2)
```

Oforth (/wiki/Category:Oforth)

```
: stringMatching(s1, s2)
| i |
   s2 isAllAt(s1, 1) ifTrue: [ System.Out s1 << " begins with " << s2 << cr ]
   s2 isAllAt(s1, s1 size s2 size - 1 + ) ifTrue: [ System.Out s1 << " ends with " << s2 << cr ]

   s1 indexOfAll(s2) ->i
   i ifNotNull: [ System.Out s1 << " includes " << s2 << " at position : " << i << cr ]

   "\nAll positions : " println
   1 ->i
   while (s1 indexOfAllFrom(s2, i) dup ->i notNull) [
      System.Out s1 << " includes " << s2 << " at position : " << i << cr
      i s2 size + ->i
      ] ;
```

Output:

```
> "arduinoardblobard", "ard" stringMatching
arduinoardblobard begins with ard
arduinoardblobard ends with ard
arduinoardblobard includes ard at position : 1

All positions :
arduinoardblobard includes ard at position : 1
arduinoardblobard includes ard at position : 8
arduinoardblobard includes ard at position : 15
```

OxygenBasic (/wiki/Category:OxygenBasic)

```
string s="sdfkjhgsdfkdfgkbopefioqwurti487sdfkrglkjfs9wrtgjglsdfkdkjcnmmb.,msfjflkjsdfk"

string f="sdfk"

string cr=chr(13)+chr(10),tab=chr(9)

string pr="FIND STRING LOCATIONS" cr cr

sys a=0, b=1, count=0, ls=len(s), lf=len(f)

do
  a=instr b,s,f
  if a=0 then exit do
  count++
  if a=1 then  pr+="Begins with keyword" cr
  pr+=count tab a cr
  if a=ls-lf+1 then pr+="Ends with keyword at " a cr
  b=a+1
end do

pr+=cr "Total matches: " count cr

print pr

'FIND STRING LOCATIONS
'
'Begins with keyword
'1      1
'2      8
'3      32
'4      51
'5      73
'Ends with keyword at 73
'
'Total matches: 5
```

PARI/GP (/wiki/Category:PARI/GP)

This meets the first but not the second of the optional requirements. Note that GP treats any nonzero value as true so the location found by contains() can be ignore if not needed.

```
startsWith(string, prefix)={
  string=Vec(string);
  prefix=Vec(prefix);
  if(#prefix > #string, return(0));
  for(i=1,#prefix,if(prefix[i]!=string[i], return(0)));
  1
};
contains(string, inner)={
  my(good);
  string=Vec(string);
  inner=Vec(inner);
  for(i=0,#string-#inner,
    good=1;
    for(j=1,#inner,
      if(inner[j]!=string[i+j], good=0; break)
    );
    if(good, return(i+1))
  );
  0
};
endsWith(string, suffix)={
  string=Vec(string);
  suffix=Vec(suffix);
  if(#suffix > #string, return(0));
  for(i=1,#suffix,if(prefix[i]!=string[i+#string-#suffix], return(0)));
  1
};
```

Perl (/wiki/Category:Perl)

Using regexes:

```
$str1 =~ /^\Q$str2\E/  # true if $str1 starts with $str2
$str1 =~ /\Q$str2\E/   # true if $str1 contains $str2
$str1 =~ /\Q$str2\E$/  # true if $str1 ends with $str2
```

Using `index`:

```
index (https://perldoc.perl.org/functions/index.html)($str1, $str2) == 0                          # true if $str1 starts with $st
r2
index (https://perldoc.perl.org/functions/index.html)($str1, $str2) != -1                          # true if $str1 contains $str2
rindex (https://perldoc.perl.org/functions/rindex.html)($str1, $str2) == length (https://perldoc.perl.org/functions/length.html)($str1
) - length (https://perldoc.perl.org/functions/length.html)($str2)  # true if $str1 ends with $str2
```

Using `substr`:

```
substr (https://perldoc.perl.org/functions/substr.html)($str1, 0, length (https://perldoc.perl.org/functions/length.html)($str2)) eq $
str2  # true if $str1 starts with $str2
substr (https://perldoc.perl.org/functions/substr.html)($str1, - length (https://perldoc.perl.org/functions/length.html)($str2)) eq $s
tr2   # true if $str1 ends with $str2
```

Bonus task *(printing all positions where `$str2` appears in `$str1`)*:

```
print (https://perldoc.perl.org/functions/print.html) $-[0], "\n" while $str1 =~ /\Q$str2\E/g;  # using a regex
```

```
my $i = -1; print (https://perldoc.perl.org/functions/print.html) $i, "\n" while ($i = index (https://perldoc.perl.org/functions/index
.html) $str1, $str2, $i + 1) != -1;  # using index
```

Phix (/wiki/Category:Phix)

Library: Phix/basics (/wiki/Category:Phix/basics)

```
constant word = "the",                                      -- (also try this with "th"/"he")
         sentence = "the last thing the man said was the"
--       sentence = "thelastthingthemansaidwasthe"          -- (practically the same results)

-- A common, but potentially inefficient idiom for checking for a substring at the start is:
if match(word,sentence)=1 then
    ?"yes(1)"
end if
-- A more efficient method is to test the appropriate slice
if length(sentence)>=length(word)
and sentence[1..length(word)]=word then
    ?"yes(2)"
end if
-- Which is almost identical to checking for a word at the end
if length(sentence)>=length(word)
and sentence[-length(word)..-1]=word then
    ?"yes(3)"
end if
-- Or sometimes you will see this, a tiny bit more efficient:
if length(sentence)>=length(word)
and match(word,sentence,length(sentence)-length(word)+1) then
    ?"yes(4)"
end if
-- Finding all occurences is a snap:
integer r = match(word,sentence)
while r!=0 do
    ?r
    r = match(word,sentence,r+1)
end while
-- or equivalently:
?match_all(word,sentence)
```

Output:

```
"yes(1)"
"yes(2)"
"yes(3)"
"yes(4)"
1
16
33
{1,16,33}
```

PHP (/wiki/Category:PHP)

```
<?php
/*********************************************************************************
* This program gets needle and haystack from the caller (chm.html) (see below)
* and checks for occurrences of the needle in the haystack
* 02.05.2013 Walter Pachl
* Comments or Suggestions welcome
*********************************************************************************/
$haystack = $_POST['haystack']; if ($haystack=='') {$haystack='no haystack given';}
$needle   = $_POST['needle'];   if ($needle=='')   {$needle='no needle given';}

function rexxpos($h,$n) {
  $pos = strpos (http://www.php.net/strpos)($h,$n);
  if ($pos === false) { $pos=-1; }
  else                { $pos=$pos+1; }
  return ($pos);
 }

$pos=rexxpos($haystack,$needle);
$tx1 = "";
if ($pos==-1){ $n=0; }  // not found
else         { $n=1; }  // found once (so far)
// Special cases
if ($pos==1){ $tx1="needle found to be the start of the haystack"; }
if ($pos==strlen (http://www.php.net/strlen)($haystack)-strlen (http://www.php.net/strlen)($needle)+1)
            { $tx1="needle found at end of haystack"; }

if ($n>0) { // look for other occurrences
  $pl=$pos; // list of positions
  $p=$pos;  //
  $x="************************************";
  $h=$haystack;
  while ($p>0) {
    $h=substr (http://www.php.net/substr)($x,0,$p).substr (http://www.php.net/substr)($h,$p);
    $p=rexxpos($h,$needle);
    if ( $p>0 ) { $n=$n+1; $pl=$pl.", ".$p; }
  }
       if ($n==1) { $txt="needle found once in haystack, position: $pl."; }
  else if ($n==2) { $txt="needle found twice in haystack, position(s): $pl."; }
  else            { $txt="needle found $n times in haystack, position(s): $pl."; }
}
else              { $txt="needle not found in haystack."; }
?>
<html>
  <head>
    <title>Character Matching</title>
    <meta name="author" content="Walter Pachl">
    <meta name="date" content="02.05.2013">
    <style>
      p { font: 120% courier; }
    </style>
  </head>
  <body>
    <p><strong>Haystack: '<?php echo "$haystack" ?>'</strong></p>
    <p><strong>Needle:   '<?php echo "$needle" ?>'</strong></p>
    <p><strong><?php echo "$txt" ?></strong></p>
    <!-- special message: -->
    <p  style="color: red";><strong><?php echo "$tx1" ?></strong></p>
  </body>
</html>
```

```
<?php
<!DOCTYPE html>
<!--
/*****************************************************************************
* Here we prompt the user for a haystack and a needle
* We then invoke program chmx.php
* to check for occurrences of the needle in the haystack
* 02.05.2013 Walter Pachl
* Comments or Suggestions welcome
*****************************************************************************/
-->
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Character matching</title>
  </head>
  <body>
    <form id="test" name="test" method="post" action="chmx.php">
    <h1>Character matching</h1>
    <p>Given two strings, demonstrate the following 3 types of matchings:
    <ol style="margin-top:2; margin-bottom:2;">
    <li>Determining if the first string starts with second string
    <li>Determining if the first string contains the second string at any location
    <li>Determining if the first string ends with the second string
    </ol>
    <p>Optional requirements:
    <ol style="margin-top:2; margin-bottom:2;">
    <li>Print the location of the match(es) for part 2
    <li>Handle multiple occurrences of a string for part 2.
    </ol>
    <p style="margin-top:5; margin-bottom:3;">
       <font face="Courier"><strong>Haystack:</strong>
       <strong><input type="text" name="haystack" size="80"></strong></font></p>
    <p style="margin-top:5; margin-bottom:3;">
       <font face="Courier"><strong>Needle:  </strong>
       <strong><input type="text" name="needle" size="80"></strong></font></p>
    <p>Press <input name="Submit" type="submit" class="erfolg" value="CHECK"/>
       to invoke chmx.php.</p>
  </form>
  </body>
</html>
```

PicoLisp (/wiki/Category:PicoLisp)

```
: (pre? "ab" "abcd")
-> "abcd"
: (pre? "xy" "abcd")
-> NIL

: (sub? "bc" "abcd")
-> "abcd"
: (sub? "xy" "abcd")
-> NIL

: (tail (chop "cd") (chop "abcd"))
-> ("c" "d")
: (tail (chop "xy") (chop "abcd"))
-> NIL


(de positions (Pat Str)
   (setq Pat (chop Pat))
   (make
      (for ((I . L) (chop Str) L (cdr L))
         (and (head Pat L) (link I)) ) ) )

: (positions "bc" "abcdabcd")
-> (2 6)
```

PL/I (/wiki/Category:PL/I)

```
/*  Let s be one string, t be the other that might exist within s. */
                                                    /* 8-1-2011 */
k = index(s, t);
if k = 0 then put skip edit (t, ' is nowhere in sight') (a);
else if k = 1 then
   put skip edit (t, '  starts at the beginning of  ', s) (a);
else if k+length(t)-1 = length(s) then
   put skip edit (t, '  is at the end of  ', s) (a);
else put skip edit (t, '  is within  ', s) (a);

if k > 0 then put skip edit (t, '  starts at position  ', k) (a);
```

Optional extra:

```
/* Handle multiple occurrences. */
   n = 1;
   do forever;
      k = index(s, t, n);
      if k = 0 then
         do;
            if n = 1 then put skip list (t, ' is nowhere in sight');
            stop;
         end;
      else if k = 1 then
         put skip edit ('<', t, '> starts at the beginning of  ', s) (a);
      else if k+length(t)-1 = length(s) then
         put skip edit ('<', t, '> is at the end of  ', s) (a);
      else put skip edit ('<', t, '> is within  ', s) (a);
      n =   k + length(t);

      if k > 0 then
         put skip edit ('<', t, '> starts at position ', trim(k)) (a);
      else stop;
   end;
```

PowerShell (/wiki/Category:PowerShell)

```
"spicywiener".StartsWith("spicy")
"spicywiener".Contains("icy")
"spicywiener".EndsWith("wiener")
"spicywiener".IndexOf("icy")
[regex]::Matches("spicywiener", "i").count
```

Output:

```
True
True
True
2
2
```

Prolog (/wiki/Category:Prolog)

```
:- system:set_prolog_flag (http://pauillac.inria.fr/~deransar/prolog/bips.html)(double_quotes,codes) .

:- [library(lists)] .

%!  starts_with(FIRSTz,SECONDz)
%
% True if `SECONDz` is the beginning of `FIRSTz` .

starts_with(FIRSTz,SECONDz)
:-
lists:append(SECONDz,_,FIRSTz)
.

%!  contains(FIRSTz,SECONDz)
%
% True once if `SECONDz` is contained within `FIRSTz` at one or more positions .

contains(FIRSTz,SECONDz)
:-
contains(FIRSTz,SECONDz,_) ,
!
.

%!  contains(FIRSTz,SECONDz,NTH1)
%
% True if `SECONDz` is contained within `FIRSTz` at position `NTH1` .

contains(FIRSTz,SECONDz,NTH1)
:-
lists:append([PREFIXz,SECONDz,_SUFFIXz_],FIRSTz) ,
prolog:length(PREFIXz,NTH0) ,
NTH1 is (http://pauillac.inria.fr/~deransar/prolog/bips.html) NTH0 + 1
.

%!  ends_with(FIRSTz,SECONDz)
%
% True if `SECONDz` is the ending of `FIRSTz` .

ends_with(FIRSTz,SECONDz)
:-
lists:append(_,SECONDz,FIRSTz)
.
```

Output:

```
?- starts_with("abcdef","abc") .
true .

?- starts_with("abc","abc") .
true .

?- starts_with("abc","abcd") .
false .

?- starts_with("dabc","abc") .
false .

?- starts_with("","") .
true .

?-

?- contains("abcdef","abc") .
true.

?- contains("abcdef","abc",NTH).
NTH = 1 ;
false.

?- contains("abcdef","de",NTH).
NTH = 4 ;
false.

?- contains("abcdef","f",NTH).
NTH = 6 ;
false.

?- contains("abcde","f",NTH).
false.

?- contains("","",NTH).
NTH = 1 ; % wtf ?
false.

?- contains("a","a",NTH).
NTH = 1 ; % wtf ?
false.

?-

?- ends_with("abc","abc") .
true ;
false .

?- ends_with("abc","bc") .
true ;
false .

?- ends_with("abcd","bc") .
false .

?- ends_with("","") .
true ;
false .

?-
```

PureBasic (/wiki/Category:PureBasic)

```
Procedure StartsWith(String1$, String2$)
  Protected Result
  If FindString(String1$, String2$, 1) =1 ; E.g Found in possition 1
    Result =CountString(String1$, String2$)
  EndIf
  ProcedureReturn Result
EndProcedure

Procedure EndsWith(String1$, String2$)
  Protected Result, dl=Len(String1$)-Len(String2$)
  If dl>=0 And Right(String1$, Len(String2$))=String2$
    Result =CountString(String1$, String2$)
  EndIf
  ProcedureReturn Result
EndProcedure
```

And a verification

```
If OpenConsole()
  PrintN(Str(StartsWith("Rosettacode", "Rosetta")))          ; = 1
  PrintN(Str(StartsWith("Rosettacode", "code")))             ; = 0
  PrintN(Str(StartsWith("eleutherodactylus cruralis", "e"))) ; = 3
  PrintN(Str(EndsWith  ("Rosettacode", "Rosetta")))  ; = 0
  PrintN(Str(EndsWith  ("Rosettacode", "code")))     ; = 1
  PrintN(Str(EndsWith  ("Rosettacode", "e")))        ; = 2

  Print(#CRLF$ + #CRLF$ + "Press ENTER to exit"): Input()
  CloseConsole()
EndIf
```

An alternate and more complete solution:

```
Procedure startsWith(string1$, string2$)
  ;returns one if string1$ starts with string2$, otherwise returns zero
  If FindString(string1$, string2$, 1) = 1
    ProcedureReturn 1
  EndIf
  ProcedureReturn 0
EndProcedure

Procedure contains(string1$, string2$, location = 0)
  ;returns the location of the next occurrence of string2$ in string1$ starting from location,
  ;or zero if no remaining occurrences of string2$ are found in string1$
  ProcedureReturn FindString(string1$, string2$, location + 1)
EndProcedure

Procedure endsWith(string1$, string2$)
  ;returns one if string1$ ends with string2$, otherwise returns zero
  Protected ls = Len(string2$)
  If Len(string1$) - ls >= 0 And Right(string1$, ls) = string2$
    ProcedureReturn 1
  EndIf
  ProcedureReturn 0
EndProcedure

If OpenConsole()
  PrintN(Str(startsWith("RosettaCode", "Rosetta")))          ; = 1, true
  PrintN(Str(startsWith("RosettaCode", "Code")))             ; = 0, false

  PrintN("")
  PrintN(Str(contains("RosettaCode", "luck")))               ; = 0, no occurrences
  Define location
  Repeat
    location = contains("eleutherodactylus cruralis", "e", location)
    PrintN(Str(location))                                    ;display each occurrence: 1, 3, 7,  & 0 (no more occurrences)
  Until location = 0

  PrintN("")
  PrintN(Str(endsWith  ("RosettaCode", "Rosetta")))           ; = 0, false
  PrintN(Str(endsWith  ("RosettaCode", "Code")))              ; = 1, true

  Print(#CRLF$ + #CRLF$ + "Press ENTER to exit"): Input()
  CloseConsole()
EndIf
```

Output:

```
1
0

0
1
3
7
0

0
1
```

Python (/wiki/Category:Python)

```
"abcd".startswith("ab") #returns True
"abcd".endswith("zn") #returns False
"bb" in "abab" #returns False
"ab" in "abab" #returns True
loc = "abab".find("bb") #returns -1
loc = "abab".find("ab") #returns 0
loc = "abab".find("ab",loc+1) #returns 2
```

Racket (/wiki/Category:Racket)

```
#lang racket
(require srfi/13)
(string-prefix? "ab" "abcd")
(string-suffix? "cd" "abcd")
(string-contains "abab" "bb")
(string-contains "abab" "ba")
```

Output:

```
#t
#t
#f
1
```

Raku (/wiki/Category:Raku)

(formerly Perl 6)

Using string methods:

```
$haystack.starts-with($needle)  # True if $haystack starts with $needle
$haystack.contains($needle)     # True if $haystack contains $needle
$haystack.ends-with($needle)    # True if $haystack ends with $needle
```

Using regexes:

```
so $haystack ~~ /^ $needle  /  # True if $haystack starts with $needle
so $haystack ~~ /  $needle  /  # True if $haystack contains $needle
so $haystack ~~ /  $needle $/  # True if $haystack ends with $needle
```

Using `substr`:

```
substr($haystack, 0, $needle.chars) eq $needle  # True if $haystack starts with $needle
substr($haystack, *-$needle.chars) eq $needle   # True if $haystack ends with $needle
```

Bonus task:

```
$haystack.match($needle, :g)».from;  # List of all positions where $needle appears in $haystack
$haystack.indices($needle :overlap); # Also find any overlapping instances of $needle in $haystack
```

Retro (/wiki/Category:Retro)

```
: startsWith? ( $1 $2 - f )
  withLength &swap dip 0 swap ^strings'getSubset compare ;

"abcdefghijkl" "abcde" startsWith?
"abcdefghijkl" "bcd" startsWith?

"abcdefghijkl" "bcd" ^strings'search
"abcdefghijkl" "zmq" ^strings'search

: endsWith? ( $1 $2 - f )
  swap withLength + over getLength - compare ;

"abcdefghijkl" "ijkl" endsWith?
"abcdefghijkl" "abc" endsWith?
```

REXX (/wiki/Category:REXX)

Extra coding was added to take care of using plurals in the last output message.

```
/*REXX program  demonstrates  some  basic   character string   testing  (for matching). */
parse arg A B                                    /*obtain A and B from the command line.*/
say 'string  A  = '    A                         /*display string   A   to the terminal.*/
say 'string  B  = '    B                         /*   "       "     B    "  "       "   */
say copies('░', 70)                              /*display a line separator  (fence).   */
LB= length(B)                                    /*get the length of string  B  in bytes*/
if left(A, LB)==B  then say  'string  A  starts with string  B'
                   else say  "string  A  doesn't start with string  B"
say                                              /* [↓] another method using COMPARE BIF*/
p= pos(B, A)
if p==0  then say  "string  A  doesn't contain string  B"
         else say  'string  A  contains string  B  (starting in position'   p")"
say
if right(A, LB)==B  then say 'string  A  ends with string  B'
                    else say "string  A  doesn't end with string  B"
say
$=;   p= 0;                       do  until p==0;          p= pos(B, A, p+1)
                                  if p\==0  then $= $','   p
                                  end   /*until*/

$= space( strip($, 'L', ",") )                   /*elide extra blanks and leading comma.*/
#= words($)                                      /*obtain number of words in  $  string.*/

if #==0  then say "string  A  doesn't contain string  B"
         else say 'string  A  contains string  B '    #    " time"left('s', #>1),
              "(at position"left('s', #>1)  $")" /*stick a fork in it,  we're all done. */
```

output when the following is specified (the five Marx brothers): `Chico_Harpo_Groucho_Zeppo_Gummo p`

```
string  A  =  Chico_Harpo_Groucho_Zeppo_Gummo
string  B  =  p
░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
string  A  doesn't start with string B

string  A  contains string  B  (starting in position 10)

string  A  doesn't end with string B

string  A  contains string  B  3  times (at positions 10, 23, 24)
```

output when the following is specified: `Chico_Harpo_Groucho_Zeppo_Gummo Z`

```
string  A  =  Chico_Harpo_Groucho_Zeppo_Gummo
string  B  =  Z
░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
string  A  doesn't start with string  B

string  A  contains string  B  (starting in position 21)

string  A  doesn't end with string  B

string  A  contains string  B  1  time (at position 21)
```

output when the following is specified: `Chico_Harpo_Groucho_Zeppo_Gummo Chi`

```
string  A  =  Chico_Harpo_Groucho_Zeppo_Gummo
string  B  =  Chi
░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
string  A  starts with string  B

string  A  contains string B  (starting in position 1)

string  A  doesn't end with string  B

string  A  contains string  B  1  time (at position 1)
```

output when the following is specified: `Chico_Harpo_Groucho_Zeppo_Gummo mmo`

```
string  A  =  Chico_Harpo_Groucho_Zeppo_Gummo
string  B  =  mmo
░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
string  A  doesn't start with string  B

string  A  contains string  B  (starting in position 29)

string  A  ends with string  B

string  A  contains string  B  1  time (at position 29)
```

Ring (/wiki/Category:Ring)

```
aString = "Welcome to the Ring Programming Language"
bString = "Ring"
bStringIndex = substr(aString,bString)
if bStringIndex > 0 see "" + bStringIndex + " : " + bString ok
```

Ruby (/wiki/Category:Ruby)

```
p 'abcd'.start_with?('ab')  #returns true
p 'abcd'.end_with?('ab')    #returns false
p 'abab'.include?('bb')     #returns false
p 'abab'.include?('ab')     #returns true
p 'abab'['bb']              #returns nil
p 'abab'['ab']              #returns "ab"
p 'abab'.index('bb')        #returns nil
p 'abab'.index('ab')        #returns 0
p 'abab'.index('ab', 1)     #returns 2
p 'abab'.rindex('ab')       #returns 2
```

Run BASIC (/wiki/Category:Run_BASIC)

```
s1$ = "abc def ghi klmnop"
s2$  = "abc"  ' begins with
s3$  = "ef"   ' is in the string
s4$  = "nop"  ' ends with

sn2$ = "abcx"  ' not begins with
sn3$ = "efx"   ' not in the string
sn4$ = "nopx"  ' not ends with

if left$(s1$,len(s2$)) <> s2$ then a$ = "Not "
print "String:";s1$;" does ";a$;"begin with:";s2$

if instr(s1$,s3$) = 0 then a$ = "Not "
print "String:";s1$;" does ";a$;"contain:";s3$

if mid$(s1$,len(s1$) + 1 - len(s4$),len(s4$)) <> s4$ then a$ = "Not "
print "String:";s1$;" does ";a$;"end with:";s4$

' ---------- not ----------------------------
print
if left$(s1$,len(sn2$)) <> sn2$ then a$ = "Not "
print "String:";s1$;" does ";a$;"begin with:";sn2$

if instr(s1$,sn3$) = 0 then a$ = "Not "
print "String:";s1$;" does ";a$;"contain:";sn3$

if mid$(s1$,len(s1$) + 1 - len(sn4$),len(sn4$)) <> sn4$ then a$ = "Not "
print "String:";s1$;" does ";a$;"end with:";sn4$
```

Output:

```
String:abc def ghi klmnop does begin with:abc
String:abc def ghi klmnop does contain:ef
String:abc def ghi klmnop does end with:nop

String:abc def ghi klmnop does Not begin with:abcx
String:abc def ghi klmnop does Not contain:efx
String:abc def ghi klmnop does Not end with:nopx
```

Rust (/wiki/Category:Rust)

```
fn print_match(possible_match: Option<usize>) {
    match possible_match {
        Some(match_pos) => println!("Found match at pos {}", match_pos),
        None => println!("Did not find any matches")
    }
}

fn main() {
    let s1 = "abcd";
    let s2 = "abab";
    let s3 = "ab";

    // Determining if the first string starts with second string
    assert!(s1.starts_with(s3));
    // Determining if the first string contains the second string at any location
    assert!(s1.contains(s3));
    // Print the location of the match
    print_match(s1.find(s3)); // Found match at pos 0
    print_match(s1.find(s2)); // Did not find any matches
    // Determining if the first string ends with the second string
    assert!(s2.ends_with(s3));
}
```

```
fn main(){
    let hello = String::from("Hello world");
    println!(" Start with \"he\" {} \n Ends with \"rd\" {}\n Contains \"wi\" {}",
                                                        hello.starts_with("He"),
                                                        hello.ends_with("ld"),
                                                        hello.contains("wi"));
}
```

Output:

```
Start with "he" true
Ends with "ld" true
Contains "wi" false
```

Scala (/wiki/Category:Scala)

```
"abcd".startsWith("ab") //returns true
"abcd".endsWith("zn") //returns false
"abab".contains("bb") //returns false
"abab".contains("ab") //returns true

var (https://scala-lang.org) loc="abab".indexOf("bb") //returns -1
loc = "abab".indexOf("ab") //returns 0
loc = "abab".indexOf("ab", loc+1) //returns 2
```

Seed7 (/wiki/Category:Seed7)

```
$ include "seed7_05.s7i";

const proc: main is func
  local
    var integer: position is 0;
  begin
    writeln(startsWith("abcd", "ab")); # write TRUE
    writeln(endsWith("abcd", "zn"));   # write FALSE
    writeln(pos("abab", "bb") <> 0);   # write FALSE
    writeln(pos("abab", "ab") <> 0);   # write TRUE
    writeln(pos("abab", "bb"));        # write 0
    position := pos("abab", "ab");
    writeln(position);                 # position is 1
    position := pos("abab", "ab", succ(position));
    writeln(position);                 # position is 3
  end func;
```

Output:

```
TRUE
FALSE
FALSE
TRUE
0
1
3
```

Sidef (/wiki/Category:Sidef)

```
var first = "abc-abcdef-abcd";
var second = "abc";

say first.begins_with(second);      #=> true
say first.contains(second);         #=> true
say first.ends_with(second);        #=> false

# Get and print the location of the match
say first.index(second);            #=> 0

# Find multiple occurrences of a string
var pos = -1;
while (pos = first.index(second, pos+1) != -1) {
    say "Match at pos: #{pos}";
}
```

Smalltalk (/wiki/Category:Smalltalk)

```
a startsWith: b
a includesSubCollection: b. "inherited from superclass"
a includesString: b. "the same, but more readable"
a endsWith: b
a indexOfSubCollection: b "inherited"
a indexOfSubCollection: b startingAt: pos "inherited"
a indexOfString: b
a indexOfStringStartingAt: b
```

SNOBOL4 (/wiki/Category:SNOBOL4)

```
      s1 = 'abcdabefgab'
      s2 = 'ab'
      s3 = 'xy'
      OUTPUT = ?(s1 ? POS(0) s2)  "1. " s2 " begins " s1
      OUTPUT = ?(s1 ? POS(0) s3)  "1. " s3 " begins " s1  ;# fails

      n = 0
again s1 POS(n) ARB s2 @a                         :F(p3)
      OUTPUT = "2. " s2 " found at position "
+        a - SIZE(s2) " in " s1
      n = a                                       :(again)

p3    OUTPUT = ?(s1 ? s2 RPOS(0)) "3. " s2 " ends " s1
END
```

Output:

```
1. ab begins abcdabefgab
2. ab found at position 0 in abcdabefgab
2. ab found at position 4 in abcdabefgab
2. ab found at position 9 in abcdabefgab
3. ab ends abcdabefgab
```

Standard ML (/wiki/Category:Standard_ML)

```
String.isPrefix "ab" "abcd"; (* returns true *)
String.isSuffix "zn" "abcd"; (* returns false *)
String.isSubstring "bb" "abab"; (* returns false *)
String.isSubstring "ab" "abab"; (* returns true *)
#2 (Substring.base (#2 (Substring.position "bb" (Substring.full "abab")))); (* returns 4 *)
val loc = #2 (Substring.base (#2 (Substring.position "ab" (Substring.full "abab")))); (* returns 0 *)
val loc' = #2 (Substring.base (#2 (Substring.position "ab" (Substring.extract ("abab", loc+1, NONE))))); (* returns 2 *)
```

Swift (/wiki/Category:Swift)

```
var str = "Hello, playground"
str.hasPrefix("Hell")           //True
str.hasPrefix("hell")           //False

str.containsString("llo")       //True
str.containsString("xxoo")      //False

str.hasSuffix("playground")     //True
str.hasSuffix("world")          //False
```

Tailspin (/wiki/Category:Tailspin)

This assumes the string to be found does not contain any regex special characters, otherwise we should work with composers (parsers) see below.

```
templates find&{s:}
  when <'$s;.*'> do '$; starts with $s;' !
  when <'.*$s;'> do '$; ends with $s;' !
  when <'.*$s;.*'> do '$; contains $s;' !
  otherwise '$s; cannot be found in $;' !
end find

'abcd' -> find&{s:'ab'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'cd'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'bc'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'e'} -> !OUT::write
```

Output:

```
abcd starts with ab
abcd ends with cd
abcd contains bc
e cannot be found in abcd
```

Working with composers and literal matchers to be able to handle any string.

```
composer startsWith&{s:}
  @: 0;
  (<='$s;'>? -> @:1; <'.*'>) $@
end startsWith

composer endsWith&{s:}
  @: 0;
  (<ends|'.*'>) $@
  rule ends: (<'.'>* <='$s;'> -> @:1;)
end endsWith

composer contains&{s:}
  @: 0;
  (<~='$s;'>? <='$s;'>? -> @:1; <'.*'>) $@
end contains

templates find&{s:}
  when <?($ -> startsWith&{s:$s} <=1>)> do '$; starts with $s;' !
  when <?($ -> endsWith&{s:$s} <=1>)> do '$; ends with $s;' !
  when <?($ -> contains&{s:$s} <=1>)> do '$; contains $s;' !
  otherwise '$s; cannot be found in $;' !
end find

'abcd' -> find&{s:'ab'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'cd'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'bc'} -> !OUT::write
'
' -> !OUT::write
'abcd' -> find&{s:'e'} -> !OUT::write
'
' -> !OUT::write
'banana' -> find&{s:'na'} -> !OUT::write
```

Output:

```
abcd starts with ab
abcd ends with cd
abcd contains bc
e cannot be found in abcd
banana ends with na
```

In tailspin we don't manipulate strings by character indices but we can still work out the second part. String characters can be streamed and captured in an array, although we prefer to compare in strings, here with a composer (parser). This has also been crafted to work with strings containing special regex characters by using literal equality.

```
composer index&{s:}
  @index: 0;
  [<match>*]
  rule match: ([<~='$s;'>? ...] -> @: $@ + 1 + $::length;) <'.'>? -> $@
end index

'ba is found in positions $:'banana' -> index&{s:'ba'}; in banana' -> !OUT::write
'
' -> !OUT::write
'ana is found in positions $:'banana' -> index&{s:'ana'}; in banana' -> !OUT::write
'
' -> !OUT::write
'c is found in positions $:'banana' -> index&{s:'c'}; in banana' -> !OUT::write
```

Output:

```
ba is found in positions [1] in banana
ana is found in positions [2, 4] in banana
c is found in positions [] in banana
```

Tcl (/wiki/Category:Tcl)

In this code, we are looking in various ways for the string in the variable `needle` in the string in the variable `haystack`.

```
set isPrefix    [string equal -length [string length $needle] $haystack $needle]
set isContained [expr {[string first $needle $haystack] >= 0}]
set isSuffix    [string equal $needle [string range $haystack end-[expr {[string length $needle]-1}] end]]
```

Of course, in the cases where the needle is a glob-safe string (i.e., doesn't have any of the characters "*?[\" in), this can be written far more conveniently:

```
set isPrefix    [string match  $needle* $haystack]
set isContained [string match *$needle* $haystack]
set isSuffix    [string match *$needle  $haystack]
```

Another powerful technique is to use the regular expression engine in literal string mode:

```
set isContained [regexp ***=$needle $haystack]
```

This can be extended by getting the `regexp` to return the locations of the matches, enabling the other forms of match to be done:

```
set matchLocations [regexp -indices -all -inline ***=$needle $haystack]
# Each match location is a pair, being the index into the string where the needle started
# to match and the index where the needle finished matching

set isContained [expr {[llength $matchLocations] > 0}]
set isPrefix [expr {[lindex $matchLocations 0 0] == 0}]
set isSuffix [expr {[lindex $matchLocations end 1] == [string length $haystack]-1}]
set firstMatchStart [lindex $matchLocations 0 0]
puts "Found \"$needle\" in \"$haystack\" at $firstMatchStart"
foreach location $matchLocations {
    puts "needle matched at index [lindex $location 0]"
}
```

TUSCRIPT (/wiki/Category:TUSCRIPT)

```
$$ MODE TUSCRIPT
ASK "string1", string1=""
ASK "string2", string2=""

IF (string1.sw.string2)   THEN
PRINT string1," starts with     ",string2
ELSE
PRINT string1," not starts with ",string2
ENDIF
SET beg=STRING (string1,string2,0,0,0,end)
IF (beg!=0) THEN
PRINT string1," contains        ",string2
PRINT "  starting in position ",beg
PRINT "  ending   in position ",end
ELSE
PRINT string1," not contains    ",string2
ENDIF

IF (string1.ew.string2) THEN
PRINT string1," ends with       ",string2
ELSE
PRINT string1," not ends with   ",string2
ENDIF
```

Output:

```
string1 >Rosetta Code
string2 >Code
Rosetta Code not starts with Code
Rosetta Code contains        Code
  starting in position 9
  ending   in position 13
Rosetta Code ends with       Code
```

TXR (/wiki/Category:TXR)

TXR Lisp

```
(tree-case *args*
  ((big small)
   (cond
     ((< (length big) (length small))
      (put-line `@big is shorter than @small`))
     ((str= big small)
      (put-line `@big and @small are equal`))
     ((starts-with small big)
      (put-line `@small is a prefix of @big`))
     ((ends-with small big)
      (put-line `@small is a suffix of @big`))
     (t (iflet ((pos (search-str big small)))
          (put-line `@small occurs in @big at position @pos`)
          (put-line `@small does not occur in @big`)))))
  (otherwise
    (put-line `usage: @(ldiff *full-args* *args*) <bigstring> <smallstring>`)))
```

Output:

```
$ txr cmatch2.tl x
usage: txr cmatch2.tl <bigstring> <smallstring>
$ txr cmatch2.tl x y z
usage: txr cmatch2.tl <bigstring> <smallstring>
$ txr cmatch2.tl catalog cat
cat is a prefix of catalog
$ txr cmatch2.tl catalog log
log is a suffix of catalog
$ txr cmatch2.tl catalog at
at occurs in catalog at position 1
$ txr cmatch2.tl catalog catalogue
catalog is shorter than catalogue
$ txr cmatch2.tl catalog catalog
catalog and catalog are equal
$ txr cmatch2.tl catalog dog
dog does not occur in catalog
```

Pattern Language

```
@line
@(cases)
@  line
@  (output)
second line is the same as first line
@  (end)
@(or)
@  (skip)@line
@  (output)
first line is a suffix of the second line
@  (end)
@(or)
@  line@(skip)
@  (output)
first line is a suffix of the second line
@  (end)
@(or)
@  prefix@line@(skip)
@  (output)
first line is embedded in the second line at position @(length prefix)
@  (end)
@(or)
@  (output)
first line is not found in the second line
@  (end)
@(end)
```

Output:

```
$ txr cmatch.txr -
123
01234
first line is embedded in the second line at position 1
$ txr cmatch.txr -
123
0123
first line is a suffix of the second line
```

Vala (/wiki/Category:Vala)

```
void main() {
    var text = "一二三四五六七八九十";
    var starts = "一二";
    var ends = "九十";
    var contains = "五六";
    var not_contain = "百";

    stdout.printf(@"text: $text\n\n", );
    stdout.printf(@"starts with $starts: $(text.has_prefix(starts))\n");
    stdout.printf(@"ends with $ends: $(text.has_suffix(ends))\n");
    stdout.printf(@"starts with $starts: $(text.has_suffix(starts))\n");
    stdout.printf(@"contains $contains: $(contains in text)\n");
    stdout.printf(@"contains $not_contain: $(contains in text)\n");
}
```

Output:

```
text: 一二三四五六七八九十

starts with 一二: true
ends with 九十: true
starts with 一二: false
contains 五六: true
contains 百: false
```

VBA (/wiki/Category:VBA)

Translation of: Phix

```
Public Sub string_matching()
    word = "the"                                        '-- (also try this with "th"/"he")
    sentence = "the last thing the man said was the"
    '--       sentence = "thelastthingthemansaidwasthe" '-- (practically the same results)

    '-- A common, but potentially inefficient idiom for checking for a substring at the start is:
    If InStr(1, sentence, word) = 1 Then
        Debug.Print "yes(1)"
    End If
    '-- A more efficient method is to test the appropriate slice
    If Len(sentence) >= Len(word) _
        And Mid(sentence, 1, Len(word)) = word Then
        Debug.Print "yes(2)"
    End If
    '-- Which is almost identical to checking for a word at the end
    If Len(sentence) >= Len(word) _
        And Mid(sentence, Len(sentence) - Len(word) + 1, Len(word)) = word Then
        Debug.Print "yes(3)"
    End If
    '-- Or sometimes you will see this, a tiny bit more efficient:
    If Len(sentence) >= Len(word) _
    And InStr(Len(sentence) - Len(word) + 1, sentence, word) Then
        Debug.Print "yes(4)"
    End If
    '-- Finding all occurences is a snap:
    r = InStr(1, sentence, word)
    Do While r <> 0
        Debug.Print r
        r = InStr(r + 1, sentence, word)
    Loop
End Sub
```

Output:

```
yes(1)
yes(2)
yes(3)
yes(4)
 1
 16
 33
```

VBScript (/wiki/Category:VBScript)

```
Function StartsWith(s1,s2)
	StartsWith = False
	If Left(s1,Len(s2)) = s2 Then
		StartsWith = True
	End If
End Function

Function Contains(s1,s2)
	Contains = False
	If InStr(1,s1,s2) Then
		Contains = True & " at positions "
		j = 1
		Do Until InStr(j,s1,s2) = False
			Contains = Contains & InStr(j,s1,s2) & ", "
			If j = 1 Then
				If Len(s2) = 1 Then
					j = j + InStr(j,s1,s2)
				Else
					j = j + (InStr(j,s1,s2) + (Len(s2) - 1))
				End If
			Else
				If Len(s2) = 1 Then
					j = j + ((InStr(j,s1,s2) - j) + 1)
				Else
					j = j + ((InStr(j,s1,s2) - j) + (Len(s2) - 1))
				End If
			End If
		Loop
	End If
End Function

Function EndsWith(s1,s2)
	EndsWith = False
	If Right(s1,Len(s2)) = s2 Then
		EndsWith = True
	End If
End Function

WScript.StdOut.Write "Starts with test, 'foo' in 'foobar': " & StartsWith("foobar","foo")
WScript.StdOut.WriteLine
WScript.StdOut.Write "Contains test, 'o' in 'fooooobar': " & Contains("fooooobar","o")
WScript.StdOut.WriteLine
WScript.StdOut.Write "Ends with test, 'bar' in 'foobar': " & EndsWith("foobar","bar")
```

Output:

```
Starts with test, 'foo' in 'foobar': True
Contains test, 'o' in 'fooooobar': True at positions 2, 3, 4, 5, 6,
Ends with test, 'bar' in 'foobar': True
```

Visual Basic (/wiki/Category:Visual_Basic)

Works with: Visual Basic (/wiki/Visual_Basic) version VB6 Standard
works the same as in VBA, see String_matching#VBA (/wiki/String_matching#VBA)

Wren (/wiki/Category:Wren)

```
var s = "abracadabra"
var t = "abra"
var u = "ra"
var v = "cad"
System.print("'%(s)' starts with '%(t)' is %(s.startsWith(t))")
var indices = []
var start = 0
while (true) {
    var ix = s.indexOf(u, start)
    if (ix >= 0) {
        indices.add(ix)
        start = ix + u.count
        if (start >= s.count) break
    } else break
}
var contained = indices.count > 0
System.print("'%(s)' contains '%(u)' is %(contained) %(contained ? "at indices %(indices)" : "")")
System.print("'%(s)' ends with '%(v)' is %(s.endsWith(v))")
```

Output:

```
'abracadabra' starts with 'abra' is true
'abracadabra' contains 'ra' is true at indices [2, 9]
'abracadabra' ends with 'cad' is false
```

XPL0 (/wiki/Category:XPL0)

```
include c:\cxpl\codes;  \intrinsic 'code' declarations
string 0;               \use zero-terminated strings

func StrLen(A);         \Return number of characters in a string
char A;
int  I;
for I:= 0 to -1>>1-1 do
    if A(I) = 0 then return I;

func StrFind(A, B);     \Search for string B in string A
\Returns index of first occurrence of string B in A, or -1 if B is not found
char A, B;              \strings to be compared
int  LA, LB, I, J;
[LA:= StrLen(A);
 LB:= StrLen(B);
 for I:= 0 to LA-LB do
    [for J:= 0 to LB-1 do
        if B(J) # A(J+I) then J:= LB+1;
    if J = LB then return I;    \found
    ];
return -1;
];

char Str;  int I, J;
[Str:= "pack my box with";
\       0123456789012345
Text(0, if StrFind(Str, "pack") = 0 then "yes" else "no");  CrLf(0);    \1.
Text(0, if StrFind(Str,  "ack") = 0 then "yes" else "no");  CrLf(0);
I:= StrFind(Str, "x w");
Text(0, if I >= 0 then "yes" else "no");                                \2.
Text(0, ", at offset ");  IntOut(0, I);  CrLf(0);
I:= 0;  J:= 0;          \offsets of space characters
loop    [I:= StrFind(Str+J, " ");
        if I < 0 then quit;
        IntOut(0, I+J); ChOut(0, ^ );
        J:= J+I+1;
        ];
CrLf(0);
Text(0, if StrFind(Str, "X w") >= 0 then "yes" else "no");  CrLf(0);
Text(0, if StrFind(Str, "with") = StrLen(Str)-StrLen("with") then "yes" else "no");  CrLf(0); \3.
Text(0, if StrFind(Str, "x w" ) = StrLen(Str)-StrLen("x w" ) then "yes" else "no");  CrLf(0);
]
```

Output:

```
yes
no
yes, at offset 10
4 7 11
no
yes
no
```

XProfan (/wiki/Category:XProfan)

```
// XProfan can use StringParts, so the results here
// are the comma separated positions of the parts or 0
Proc Contains
   Parameters string content, part
   var string results = "0"
   var long posi = 1
   posi = InStr(part,content,posi)
   if posi <> 0
      results = str$(posi)
      repeat
         posi = InStr(part,content,posi+1)
         case posi <> 0 : results = results + "," + str$(posi)
      until posi == 0
   endif
   Return results
EndProc

Proc StartsWith
   Parameters string content, part
   Return if(Left$(content,Len(part)) == part, 1, 0)
EndProc

Proc EndsWith
   Parameters string content, part
   Return if(Right$(content,Len(part)) == part, 1, 0)
   'Return if(Left$(content,Len(content)-Len(part)+1) == part, 1, 0)
EndProc

var string theContent = "foobar"
var string thePart = "foo"
Print "Starts with: "
Print "  ("+thePart+" in "+theContent+") "+if(StartsWith(theContent,thePart),"Yes","No")
thePart = "back"
Print "  ("+thePart+" in "+theContent+") "+if(StartsWith(theContent,thePart),"Yes","No")

theContent = "fooooooobar"
Print "Contains: "
Print "  ("+thePart+" in "+theContent+") "+ Contains(theContent,thePart)
thePart = "o"
Print "  ("+thePart+" in "+theContent+") "+ Contains(theContent,thePart)

theContent = "foobar"
thePart = "back"
Print "Ends with: "
Print "  ("+thePart+" in "+theContent+") "+if(EndsWith(theContent,thePart),"Yes","No")
thePart = "bar"
Print "  ("+thePart+" in "+theContent+") "+if(EndsWith(theContent,thePart),"Yes","No")

waitkey
end
```

Output:

```
Starts with:
  (foo in foobar) Yes
  (back in foobar) No
Contains:
  (back in fooooooobar) 0
  (o in fooooooobar) 2,3,4,5,6,7
Ends with:
  (back in foobar) No
  (bar in foobar) Yes
```

Yabasic (/wiki/Category:Yabasic)

```
cadena1$ = "qwertyuiop"

//Determinar si la primera cadena comienza con la segunda cadena
cadena2$ = "qwerty"
if left$(cadena1$, len(cadena2$)) = cadena2$ then
    print "'", cadena1$, "' comienza con '", cadena2$, "'"
else
    print "'", cadena1$, "' no comienza con '", cadena2$, "'"
end if

//Determinar si la primera cadena contiene la segunda cadena en cualquier
//ubicación imprima la ubicación de la coincidencia para la parte 2
cadena2$ = "wert"
posic = instr(cadena1$, cadena2$)
if posic then
    print "'", cadena1$, "' contiene '", cadena2$, "' en la posicion ", posic
else
    print "'", cadena1$, "' no contiene '", cadena2$, "'"
end if

//Determinar si la primera cadena termina con la segunda cadena
cadena2$ = "random garbage"
if right$(cadena1$, len(cadena2$)) = cadena2$ then
    print "'", cadena1$, "' termina con '", cadena2$, "'"
else
    print "'", cadena1$, "' no termina con '", cadena2$, "'"
end if
end
```

Output:

```
'qwertyuiop' comienza con 'qwerty'
'qwertyuiop' contiene 'wert' en la posicion 2
'qwertyuiop' no termina con 'random garbage'
```

zkl (/wiki/Category:Zkl)

```
fcn f(text,p){ if(text.find(p)==0)println("Yep") else println("Nope") }
f("foobar","foo") //--> Yep
f("foobar","bar") //--> Nope
```

```
fcn f(text,p){ if(Void!=(n:=text.find(p)))println("Contained @",n) else println("Nope") }
f("foobar","ob")   //--> Contained @2
f("foobar","food") //--> Nope
```

```
fcn f(text,p){
   if( Void!=(n:=text.rfind(p)) and n+p.len()==text.len() )
      println("tail gunner") else println("Nope")
}
f("foobar","r"); f("foobar","ar"); //--> tail gunners
f("foobar","ob");  //--> Nope
f("foobarfoobar","bar"); //--> tail gunner
```

Categories (/wiki/Special:Categories): Programming Tasks (/wiki/Category:Programming_Tasks) | String manipulation (/wiki/Category:String_manipulation) | Basic Data Operations (/wiki/Category:Basic_Data_Operations) | Simple (/wiki/Category:Simple) | 11l (/wiki/Category:11l) | 360 Assembly (/wiki/Category:360_Assembly) | AArch64 Assembly (/wiki/Category:AArch64_Assembly) | Ada (/wiki/Category:Ada) | Aime (/wiki/Category:Aime) | ALGOL 68 (/wiki/Category:ALGOL_68) | AppleScript (/wiki/Category:AppleScript) | ARM Assembly (/wiki/Category:ARM_Assembly) | Arturo (/wiki/Category:Arturo) | AutoHotkey (/wiki/Category:AutoHotkey) | AutoIt (/wiki/Category:AutoIt) | AWK (/wiki/Category:AWK) | BASIC (/wiki/Category:BASIC) | Applesoft BASIC (/wiki/Category:Applesoft_BASIC) | Batch File (/wiki/Category:Batch_File) | BBC BASIC (/wiki/Category:BBC_BASIC) | Bracmat (/wiki/Category:Bracmat) | C (/wiki/Category:C) | C sharp (/wiki/Category:C_sharp) | C++ (/wiki/Category:C%2B%2B) | Clojure (/wiki/Category:Clojure) | CoffeeScript (/wiki/Category:CoffeeScript) | Common Lisp (/wiki/Category:Common_Lisp) | Component Pascal (/wiki/Category:Component_Pascal) | D (/wiki/Category:D) | DCL (/wiki/Category:DCL) | Delphi (/wiki/Category:Delphi) | Dyalect (/wiki/Category:Dyalect) | E (/wiki/Category:E) | EchoLisp (/wiki/Category:EchoLisp) | Elena (/wiki/Category:Elena) | Elixir (/wiki/Category:Elixir) | Emacs Lisp (/wiki/Category:Emacs_Lisp) | Erlang (/wiki/Category:Erlang) | Euphoria (/wiki/Category:Euphoria) | F Sharp (/wiki/Category:F_Sharp) | Factor (/wiki/Category:Factor) | Falcon (/wiki/Category:Falcon) | Fantom (/wiki/Category:Fantom) | FBSL (/wiki/Category:FBSL) | Forth (/wiki/Category:Forth) | Fortran (/wiki/Category:Fortran) | FreeBASIC (/wiki/Category:FreeBASIC) | Gambas (/wiki/Category:Gambas) | GML (/wiki/Category:GML) | Go (/wiki/Category:Go) | Groovy (/wiki/Category:Groovy) | Haskell (/wiki/Category:Haskell) | Icon (/wiki/Category:Icon) | Unicon (/wiki/Category:Unicon) | J (/wiki/Category:J) | Java (/wiki/Category:Java) | JavaScript (/wiki/Category:JavaScript) | Jq (/wiki/Category:Jq) | Julia (/wiki/Category:Julia) | K (/wiki/Category:K)

| Kotlin (/wiki/Category:Kotlin) | LabVIEW (/wiki/Category:LabVIEW) | Lang5 (/wiki/Category:Lang5) | Lasso (/wiki/Category:Lasso)
| Liberty BASIC (/wiki/Category:Liberty_BASIC) | Lingo (/wiki/Category:Lingo) | Logo (/wiki/Category:Logo) | Lua (/wiki/Category:Lua)
| M2000 Interpreter (/wiki/Category:M2000_Interpreter) | Maple (/wiki/Category:Maple) | Mathematica (/wiki/Category:Mathematica) | MATLAB (/wiki/Category:MATLAB)
| Octave (/wiki/Category:Octave) | Min (/wiki/Category:Min) | MiniScript (/wiki/Category:MiniScript) | NetRexx (/wiki/Category:NetRexx)
| NewLISP (/wiki/Category:NewLISP) | Nim (/wiki/Category:Nim) | Objeck (/wiki/Category:Objeck) | Objective-C (/wiki/Category:Objective-C)
| OCaml (/wiki/Category:OCaml) | Oforth (/wiki/Category:Oforth) | OxygenBasic (/wiki/Category:OxygenBasic) | PARI/GP (/wiki/Category:PARI/GP)
| Perl (/wiki/Category:Perl) | Phix (/wiki/Category:Phix) | Phix/basics (/wiki/Category:Phix/basics) | PHP (/wiki/Category:PHP) | PicoLisp (/wiki/Category:PicoLisp)
| PL/I (/wiki/Category:PL/I) | PowerShell (/wiki/Category:PowerShell) | Prolog (/wiki/Category:Prolog) | PureBasic (/wiki/Category:PureBasic)
| Python (/wiki/Category:Python) | Racket (/wiki/Category:Racket) | Raku (/wiki/Category:Raku) | Retro (/wiki/Category:Retro) | REXX (/wiki/Category:REXX)
| Ring (/wiki/Category:Ring) | Ruby (/wiki/Category:Ruby) | Run BASIC (/wiki/Category:Run_BASIC) | Rust (/wiki/Category:Rust) | Scala (/wiki/Category:Scala)
| Seed7 (/wiki/Category:Seed7) | Sidef (/wiki/Category:Sidef) | Smalltalk (/wiki/Category:Smalltalk) | SNOBOL4 (/wiki/Category:SNOBOL4)
| Standard ML (/wiki/Category:Standard_ML) | Swift (/wiki/Category:Swift) | Tailspin (/wiki/Category:Tailspin) | Tcl (/wiki/Category:Tcl)
| TUSCRIPT (/wiki/Category:TUSCRIPT) | TXR (/wiki/Category:TXR) | Vala (/wiki/Category:Vala) | VBA (/wiki/Category:VBA) | VBScript (/wiki/Category:VBScript)
| Visual Basic (/wiki/Category:Visual_Basic) | Wren (/wiki/Category:Wren) | XPL0 (/wiki/Category:XPL0) | XProfan (/wiki/Category:XProfan)
| Yabasic (/wiki/Category:Yabasic) | Zkl (/wiki/Category:Zkl) | Bc/Omit (/wiki/Category:Bc/Omit) | Dc/Omit (/wiki/Category:Dc/Omit)