

Programming with String

A string of Characters

A **string of characters** is a sequence of data of type **char** (the ASCII codes) stored in consecutive memory locations and **terminated** by the **null character** '**\0**' (the ASCII value is 0). The **null string** (of length zero (0)) is the **null character** only.

An Example ...

A string constant is written within a pair of double quotes. Consider the string “IIT Kharagpur”. It is stored as:

0														13
I	I	T		K	h	a	r	a	g	p	u	r	\0	...
73	73	84	32	...								0	...	

The length of this string is 13 [0 ... 13].

Character Array

An 1-D array of type `char` is used to store a **string**.

Initialization of String

```
#include <stdio.h>
#define MAXLEN 100
int main() // stringInit.c
{
    char a[MAXLEN]={'B','i','g',' ','
                    'B','a','n','g','\0'},
        b[MAXLEN]="Black Hole",
        c[]="Quantum Gravity",
        *cP="Super String" ;
```

```
printf("a: %s\nb: %s\nc: %s\n*cP: %s\n",
      a, b, c, cP) ;
printf("a[0] = %c, b[1] = %c, c[2] = %c, cP[3] = %c\n",
      a[0], b[1], c[2], cP[3]) ;
return 0;
}
```

Output

```
$ cc -Wall stringInit.c
```

```
$ a.out
```

```
a: Big Bang
```

```
b: Black Hole
```

```
c: Quantum Gravity
```

```
*cP: Super String
```

```
a[0] = B, b[1] = l, c[2] = a, cP[3] = e
```

Note

- All constant strings are stored in the **read-only** memory.
- Space is allocated for **a[]**, **b[]**, **c[]** to store the strings. But the pointer **cP** directly points to the beginning of the string in the **read-only** memory. Any attempt to change that location results in **segmentation violation** in GCC.

Initialization of String

```
#include <stdio.h>
#define MAXLEN 100
int main() // stringInit1.c
{
    char a[MAXLEN]={'B','i','g',' ','
                    'B','a','n','g','\0'},
        b[MAXLEN]="Black Hole",
        c[]="Quantum Gravity",
        *cP="Super String" ;
```

```
a[0] = 'A', b[0] = 'A', c[0] = 'A' ;  
cP[0] = 'A' ; // segmentation fault  
return 0;  
}
```

Reading a String

The library function `scanf()` can be used to read a string. The format conversion specifier for a sequence of non-white-space characters is `%s`.

```
#include <stdio.h>
#define MAXLEN 100
int main() // stringRead1.c
{
    char a[MAXLEN], b[MAXLEN], c[MAXLEN] ;

    printf("Enter the 1st string of char\n");
    scanf("%s",a); printf("a: %s\n",a);
    printf("Enter the 2nd string of char\n");
    scanf("%[^\n]",b); printf("b: %s\n",b);
    printf("Enter the 3rd string of char\n");
    scanf(" %[^\n]",c); printf("c: %s\n",c);
    return 0;
}
```

Output

```
$ cc -Wall stringRead1.c
$ a.out
Enter the 1st string of char
The world is made of facts
a: The
Enter the 2nd string of char
b: world is made of facts
Enter the 3rd string of char
and not of matter.
c: and not of matter.
```

Problem Solving

Write a non-recursive function that will take a character string as a parameter and will return its length. Also write a recursive function to do the same job.

Inductive Definition: length

$$\text{length}(s) = \begin{cases} 0 & \text{if } s = \text{null}, \\ 1 + \text{length}(\text{tail}(s)) & \text{otherwise.} \end{cases}$$

The **tail**(*s*) is the string after removal of the 0th character of *s*.

Non-recursive length()

```
int length(char *s) {  
    int len=0;  
  
    while(s[len]) ++len ;  
    return len;  
} // lengthI.c
```


Recursive length()

```
#define NIL ('\0')
int length(char *s) {
    if(s[0] == NIL) return 0;
    return 1 + length(s+1);
} // lengthR.c
```

string.h

Library functions (`libc`) are available to manipulate strings. The prototypes of these functions are available in the header file `string.h`. The function `size_t strlen(const char *s);` returns the length of the string. Note that the character string pointed by `s` cannot be changed (`const`).

strlen()

```
#include <stdio.h>
#include <string.h>
#define MAXLEN 100
int length(char *);
int main() // lengthL.c
{
    char b[MAXLEN] ;
    printf("Enter a string of char\n") ;
    scanf("%[^\\n]", b) ;
```

```
    printf("length(%s) = %d\n",b,strlen(b));  
    return 0;  
} // lengthL.c
```

Problem Solving

Write a non-recursive and a recursive program that copies a string to another array. We assume that the target array has sufficient space. The function returns the number of character copied.

Non-recursive Function

```
#define NIL ('\0')
int strcpy(char *dst, const char *src){
    int count=-1;
    do{
        ++count ;
        dst[count] = src[count];
    } while(src[count] != NIL);
    return count ;
} // stringCopyI.c
```

Recursive Function

```
#define NIL ('\0')
int strCopy(char *dst, const char *src){
    dst[0] = src[0] ;
    if(src[0] == NULL) return 0 ;
    return strCopy(src+1, dst+1) + 1 ;
} // stringCopyR.c
```

strcpy()

The function

`char *strcpy(char *dest, const char *src);` copies the source string to the destination string. It also returns the destination string pointer.

Problem Solving

Write a non-recursive C Function that will concatenate a string to the end of another string. As an example, let `s:"IIT "` and `ct:"Kharagpur"`. After the concatenation `s:"IIT Kharagpur"`.

Non-recursive Function

```
#define NIL ('\0')  
  
int concat(char *dst, const char *sec){  
    int count=0, i=0;  
    while(dst[count] != NIL) ++count;  
    do dst[count++] = sec[i];  
    while(sec[i++] != NIL);  
    return count-1;  
} // concatI.c
```



strcat()

The function

`char *strcat(char *dest, const char *src);` concatenates the source string to the destination string. It also returns the destination string pointer.

Problem Solving

Write a non-recursive and a recursive C Function that will compare two strings $s1$ and $s2$. It returns < 0 if $s1 < s2$, 0 if $s1 = s2$, or > 0 if $s1 > s2$.

Non-recursive Function

```
#define NIL ('\0')
int strcmp(const char *s1, const char *s2){
    int i=0;

    while(s1[i]==s2[i] && s1[i] != NIL
           && s2[i] != NIL) ++i;

    if(s1[i] == s2[i]) return 0;
    return (int)(s1[i] - s2[i]);
} // strcmp.c
```

Problem Solving

Write a non-recursive and a recursive C Function that will test whether a pattern string is the prefix of a text string. As an example “IIT” is a prefix of “IIT Kanpur”, but “IIIT” is not.

Inductive definition

$$\text{prefix}(p, t) = \begin{cases} \text{true, if } p = \text{Nil} \\ \text{false, if } p \neq \text{Nil} \text{ and } t = \text{Nil} \\ \text{false, if } \text{head}(p) \neq \text{head}(t) \\ \text{prefix}(\text{tail}(p), \text{tail}(t)), \\ \quad \text{if } \text{head}(p) = \text{head}(t) \end{cases}$$

Steps

A pattern p of length m is a prefix of a text t of length n , if $m \leq n$ and $p[0 \cdots m - 1]$ is same as $t[0 \cdots m - 1]$.

Non-recursive Function

```
#define TRUE 1
#define FALSE 0
static int length(const char *) ;
int isPrefix(const char *t, const char *p) {
    int m = length(p), n = length(t), i;

    if(m > n) return FALSE ;
    for(i=0; i<m; ++i)
        if(p[i] != t[i]) return FALSE;
```

```
    return TRUE;
} // isPrefixI.c
static int length(const char *s) {
    int len=0;

    while(s[len]) ++len ;
    return len;
}
```

Recursive Function

```
#define TRUE 1
#define FALSE 0
#define NIL ('\0')
int isPrefix(const char *t, const char *p) {
    if(*p == NIL) return TRUE;
    if(*p != *t) return FALSE;
    return isPrefix(t+1, p+1);
} // isPrefixR.c
```

Problem Solving

Write a non-recursive C Function that tests whether a pattern string is a substring of a text string. The function returns -1 if the pattern is not a substring, otherwise it returns the index of the starting position (first occurrence) of the pattern in the text.

Similarly write a recursive C function.

Non-recursive C Function

```
#define NIL ('\0')
int isSubString(const char *t, const char *p){
    int index = 0;
    const char *pP, *tP ;

    while (*t != NIL) { // p cannot be a substring
        tP = t ;
        pP = p ;
        do {
            if(*pP == NIL) return index ; // p is a substring
            if(*tP == NIL) return NOTSUBSTR ;
            tP++; pP++;
        } while (*tP != NIL);
        index++; t++;
    }
    // p cannot be a
```

```
    } while (*pP++ == *tP++);           // test next char
    ++index ;
    ++t ;                               // shift in text
}
return NOTSUBSTR ;
} // subStringI.c
```

Recursive C Function

```
#define NIL ('\0')
#define TRUE 1
#define FALSE 0
int isSubString(const char *t, const char *p){
    int n ;

    if(length(t) < length(p)) return -1 ;
    if(isPrefix(t, p)) return 0 ;
    n = isSubString(t+1, p) ;
    if(n == -1) return -1 ;
    else return n + 1 ;
} // subStringR.c
```

`strstr()`

The library function `char *strstr(const char *t, const char *p)`; This function finds the first occurrence of the substring `p` in the string `t`.

Problem Solving

Write a non-recursive function that will **left-shift** a non-null string. A null string will remain as it is.

Write a recursive function to solve this problem.

what()

What does the following function do?

```
int what() {  
    char not ;  
    int but ;  
  
    if((not = getchar()) == EOF) return 0 ;  
    but = what() + 1 ;  
    putchar(not) ;  
    return but ;  
} // printReverse1.c
```

Input: abc\nEOF

