CS11001/CS11002 Programming and Data Structures (PDS)

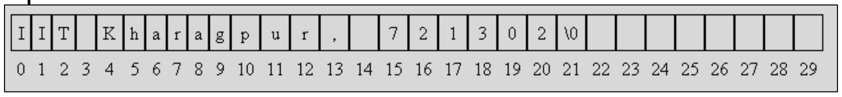
(Theory: 3-1-0)

Strings

De-limiters in C

- In C a *string* is defined to be a null-terminated character array.
- The null character ('\0') is used to indicate the end of the string.
- Like any other arrays, C does not impose range checking of array indices for strings.
- Declaration of an array allocates a fixed space for it. You need not use the entire space.
- Instead you can store your data in the initial portion of the array. It is, therefore, necessary to put a boundary of the *actual data*.
- This is the reason why we pass the size parameter to functions along with arrays.
  - Strings handle it differently, namely by putting an explicit marker at the end of the actual data.

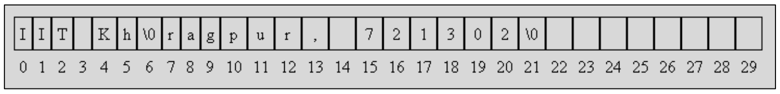
Delimiter \0



- Here we use an array of size 30.
- The string "IIT Kharagpur, 721302" is stored in the first 21 locations.
- This is followed by the null character.
- A total of 22 characters is needed to represent this string of length 21.
- Whatever follows after this null character is irrelevant for defining the string.

Delimiter \0

- Whatever follows after this null character is irrelevant for defining the string.
- If we set the element at location 6 to '\0', the array looks like:



I	I	T		K	h	\0	r	a	g	p	u	r	,		7	2	1	3	0	2	\0								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29

- Considered as a string this stands for "IIT Kh".

- Recall that C allows you to read from and write to the locations at indices 30,31,... of this array.
- These are memory locations not allocated to the array, since its size is 30.
- Writing beyond the allocated space is expected to corrupt memory or even raise fatal run-time errors (Segmentation faults).
- In particular, if you do not put the null character at the end of the string, C keeps on searching for it and may go out of the legal boundary and create troubles.

- C offers some built-in functions for working with strings. They assume (null-terminated) strings as input and create (null-terminated) strings.
- You do not have to append the null character explicitly. For example, the statement
  - strcpy~~(A,"IIT Kharagpur"); copies the string "IIT Kharagpur"~~ to the character array A and also appends the required null character at the end of it.
- In order to use these string functions you should #include <string.h>. No additional libraries need be linked during compilation time.

Reading and writing a string

```
#include<stdio.h>
main()
{
 char month[15];
 printf("Enter the string");
 gets(month);
 printf("The string is %s\n",month);
}
```

Program to illustrate char pointers

```
#include<stdio.h>
main()
{
  char charr[]="Pointers and Strings";
  char *chptr;
  chptr=charr;
  printf("address pointed to by the pointer is
    %x\n",chptr);
  printf("contents pointed by the pointer chptr is:
    %c\n",*chptr);
}
```

Reading a printing an array of pointers to strings-dynamic memory allocation

```
ramshyamjadvi string2.c
#include<stdio.h>
#include<malloc.h>

main()
{
 char *names[10];
 int i;

 for(i=0;i<3;i++)
   names[i]=(char*)malloc(10*sizeof(char));

 for(i=0;i<3;i++)
   scanf("%s",names[i]);

 for(i=0;i<3;i++)
   printf("%s",names[i]);

}
```

String Library Functions

- int strlen (const char s[]);
  - Returns the length (the number of characters before the first null character) of the string s.

mystrlen function

```
#include<stdio.h>
#include<string.h>
int mystrlen(char  *);
main()
{
  char text[10];
  printf("Enter string:");
  scanf("%s",text);
  printf("%s\n",text);
  printf(text);
  printf(": length is %d\n",mystrlen(text));

}
int mystrlen(char *ptr)
{
 int cnt=0;
 while(*ptr!='\0')
 {   cnt++;   ptr++;
 }
return(cnt);
}
```

Using pointers to write mystrlen()

```
#include<stdio.h>
main()
{
 char string [80], *ptr;
 ptr=string;
 printf("Enter the string:");
 while((*ptr++=getchar())!='\n');
 *--ptr = '\0';
 printf("string is %s\n",string);
 printf("Length is %d\n",ptr-string);
}
```

mystrcpy function

```
#include<stdio.h>
#include<string.h>
void mystrcpy(char *, char *);

main()
{
  int len;
  char s1[]="Good";
  char s2[10];
  mystrcpy(s2,s1);//library function
  printf("Copied string is %s\n",s2);
}

void mystrcpy(char *p, char *q)
{
 while(*p++=*q++);
}
```

mystrcmp function

```
#include<stdio.h>
int mystrcmp(char *, char *);

main()
{
  char str1[20], str2[20];
  int k;
  gets(str1);
  gets(str2);
  k=mystrcmp(str1,str2);
 if(!k)
   printf("Both are the same strings\n");
  else if(k>0)
    printf("Str1 is lesser than Str2\n");
  else
    printf("Str1 is greater than Str2\n");
}
```

```
int mystrcmp(char *str1, char
*str2)
{
 char *p, *q;

for(p=str1,q=str2;((*p==*q)&&(*
p!='\0')&&(*q!='\0'));p++,q++);

 if((*p=='\0')&&(*q=='\0'))
   return 0;
 else if(*p < *q) return 1;
 else return -1;
}
```