Programming & Data Structure

CS 11002

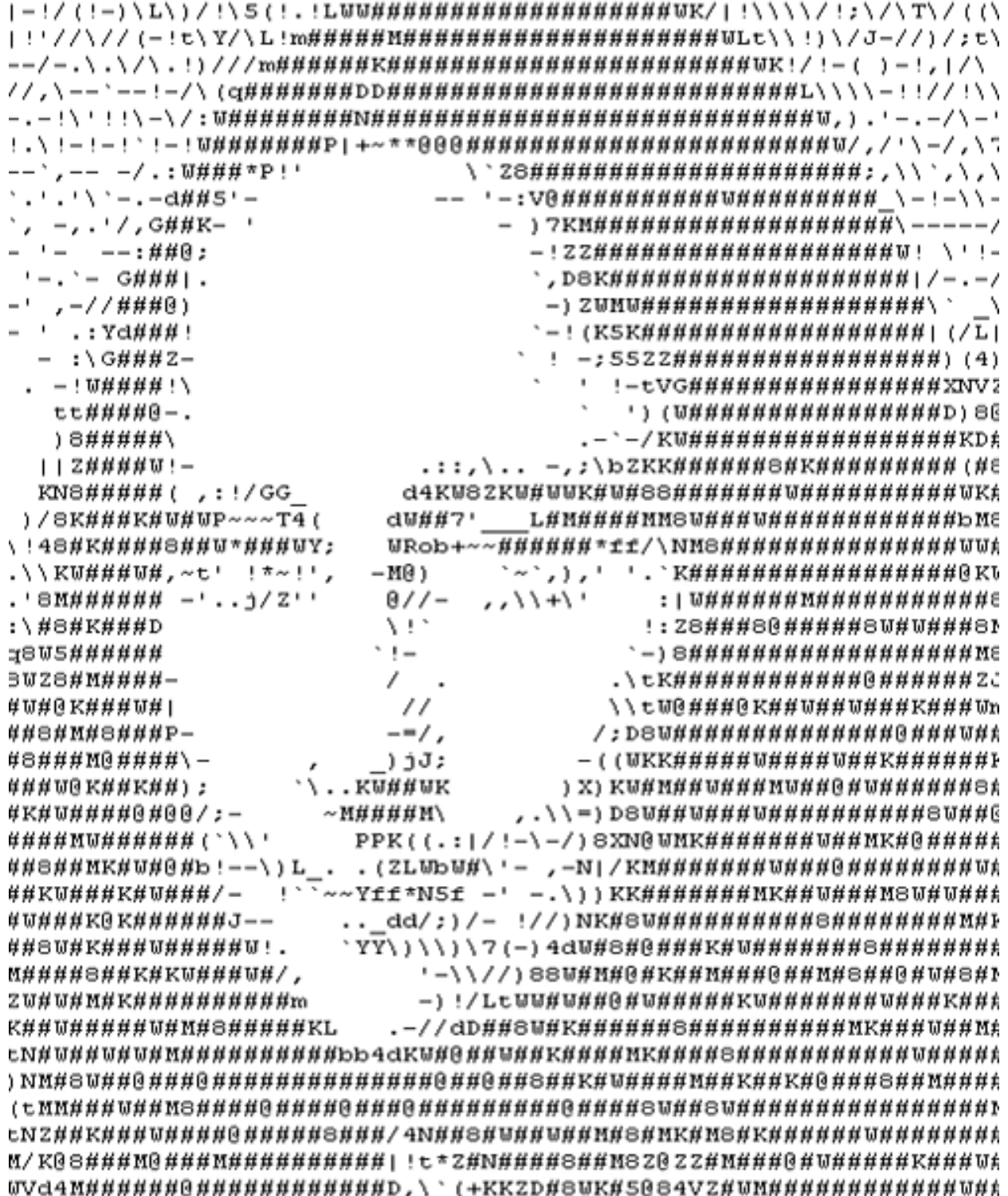
Partha Bhowmick

http://cse.iitkgp.ac.in/~pb

CSE Department
IIT Kharagpur

Spring 2012-2013

Characters & Strings



String (1)

A **string of characters** is a sequence of data of type `char` (ASCII codes) stored in a 1-D array comprising consecutive memory locations and *terminated* by the NULL character (`\0` of ASCII value = 0).

A string constant is written within a pair of *double quotes*, e.g., `"IIT Kharagpur"`.

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

Initialization of a String (1)

```
#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;}
```

Initialization of a String (2)

```
$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
```

Initialization of a String (3)

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 a String (4)

```
#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'; //OK
  cP[0] = 'A';  // segmentation fault
  return 0;}
```

Reading a String (1)

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");
```

Reading a String (2)

```
  scanf(" %[^\n]",c); printf("c: %s\n",c);
  return 0;}
```

```
$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.
```

String Length (1)

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.

String Length (2)

Non-recursive Function

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

String Length

Recursive Definition

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

where $\textit{tail}(s)$ is the string after removal of the 0th character of s.

String Length

Recursive Function

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

String Length

The function `size_t strlen(const char *s)` returns the length of the string.

Note

- `string.h` is the header file that contains the library functions to manipulate strings.
- The null character (`\0`) is not counted.
- The character string pointed by `s` cannot be changed, as it is `const`.

To learn more, type on the terminal: `man strlen`.

To see your computer's manual on a library function, you should type:

`man <function name>`.

String Length


```
#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;}
```

String Copy

Write a non-recursive and a recursive program to copy a string to another array.
Assume that the target array has sufficient space.
The function should return the number of characters copied.

String Copy (2)

Non-recursive Function

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

String Copy

Recursive Function

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

String Copy

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

Example

```
strcpy(mycity, "Calcutta");
strcpy(city, mycity);
```

`mycity = "Calcutta";` *is an invalid assignment, since assignment operator does not work for strings.*

String Concatenation (1)

Write a non-recursive C Function that will concatenate a string to the end of another string.

Example

Let `s="IIT"`, `t="Kharagpur"`.
After concatenation, `s="IITKharagpur"`.

String Concatenation (2)

Non-recursive Function

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

String Concatenation (3)

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

String Comparison (1)

Write a non-recursive and a recursive C Function to compare two strings `s1` and `s2`.
They should return `< 0` if `s1 < s2`, `0` if `s1 = s2`, or `> 0` if `s1 > s2`.

String Comparison (2)

Non-recursive Function

```
#define NULL ('\0')
int strComp(const char *s1, const char *s2){
    int i=0;
    while(s1[i]==s2[i]
          && s1[i] != NULL
          && s2[i] != NULL) ++i;
    if(s1[i] == s2[i]) return 0;
    return (int)(s1[i] - s2[i]);
} // strCompI.c
```

String Prefix (1)

Pattern

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

Example

`"I"` or `"II"` or `"IIT"` or `"IIT "` or `"IIT K"` or ... is a prefix of `"IIT Kanpur"`, but `"IIIT"` is not.

Write a non-recursive and a recursive C Function to test whether a pattern string is the prefix of a text string.

String Prefix

Inductive definition

$$prefix(p,t) = \begin{cases} \textsc{True}, \text{ if } p = \text{NULL} \\ \textsc{False}, \text{ if } p \neq \text{NULL and } t = \text{NULL} \\ \textsc{False}, \text{ if } head(p) \neq head(t) \\ prefix(tail(p),\ tail(t)), \\ \qquad\qquad \text{if } head(p) = head(t) \end{cases}$$

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

String Prefix

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;}
```

String Prefix

Recursive Function

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

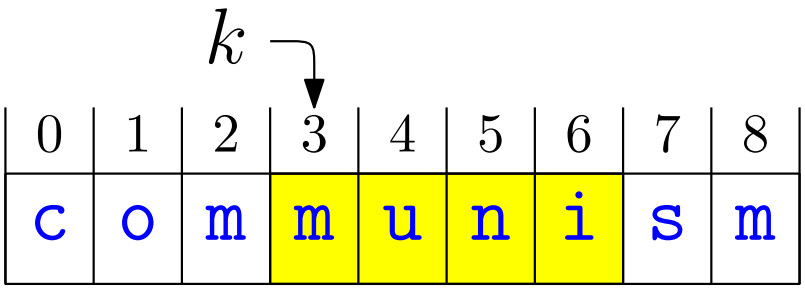
Substring Matching (1)

Substring

A pattern $p[0 \cdots m-1]$ is a **substring** of a text $t[0 \cdots n-1]$ if $m \leqslant n$ and $p[0 \cdots m-1]$ is same as $t[k \cdots k+m-1]$ for some $k \geqslant 0$.

Example

`"muni"` is a substring of `"communism"` with $k = 3$.



Substring Matching (2)

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.

Substring Matching (3)

Non-recursive Function

```
#define NULL ('\0')
int isSubString(const char *t, const char *p){
  int index = 0; const char *pP, *tP;
  while (*t != NULL) { //p cannot be a substring
    tP = t; pP = p;
    do {
      if(*pP == NULL) return index; //p is a substring
      if(*tP == NULL) return -1; //p not a substring
    } while (*pP++ == *tP++); //test next character
    ++index; ++t; } //end while
  return NOTSUBSTR;} //subStringI.c
```

Substring Matching (4)

Recursive Function

```
#define NULL ('\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
```

Substring Matching

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