

第四章 串



字符串 (string)

- 字符串是零个或多个字符的有限序列，记作：

$$S = 'a_1 a_2 a_3 \cdots a_n', \quad n \geq 0$$

- 其中， S 是串名字

' $a_1 a_2 a_3 \cdots a_n$ ' 是串值

a_i 是串中字符

n 是串的长度(串中字符的个数)


- 例如， $S = \text{'Shenzhen University'}$

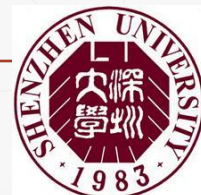


字符串术语

- 空串：不含任何字符的串，串长为零
- 空格串：仅由一个或多个空格组成的串，它的长度为串中空格的个数
- 子串：由串中任意个连续的字符组成的子序列
- 主串：包含子串的串。

[例] a= 'BEI' , b= 'JING' ,
c= 'BEIJING' , d= 'BEI JING'

- ❑ a和b都是c和d的子串
- ❑ c是d的子串吗? 



字符串术语

- 子串位置：子串的**第一个**字符在主串中第一次出现的位置
- 字符位置：字符在主串中第一次出现的位置
- **串相等**的条件：当两个串的**长度相等**且各个**对应位置的字符都相等**时才相等。
- 模式匹配：确定子串在主串中首次出现位置的运算

[例] a= 'BEI' , b= 'JING' ,

c= 'BEIJING' , d= 'BEI JING'

□ c和d是相等的串吗？✗



串的抽象数据类型描述

- ADT String {

数据对象: $D = \{a_i | a_i \in CharacterSet, i = 1, 2, 3, \dots, n, n \geq 0\}$

数据关系: $R = \{ \langle a_{i-1}, a_i \rangle | a_{i-1}, a_i \in D \}$

基本操作: StrAssign (&T, chars) // 串赋值

StrCompare(S, T) // 串比较

StrLength(S) // 求串长

Concat (&T, S1, S2) // 串联接

Substring (&Sub, S, pos, len) // 求子串

.....

} ADT String



字符串与线性表的关系

- 串的**逻辑结构和线性表**极为相似，它们都是**线性结构**
- 区别主要表现为：
 - ✓ 串的数据对象约定是**字符集**
 - ✓ 在线性表的基本操作中，以“单个元素”作为操作对象，而在串的基本操作中，通常以“**串的整体**”作为操作对象，如：**在串中查找每个子串、求取一个子串、在串的某个位置上插入一个子串以及删除一个子串等。**

串 的表示与实现

- 串的存储结构:
 - 顺序存储
 - ✓ 串的定长顺序存储表示
 - ✓ 串的堆分配存储表示
 - 链式存储
 - ✓ 串的块链存储表示



串的定长顺序存储

- 用一组地址**连续**的存储单元存储字符序列
- 按照预定义的大小，分配一个固定长度的存储区，可用定长数组来描述：

```
#define MAXSTRLEN 255    //定义了长度为MAXSTRLEN字符存储空间
```

```
char Str[MAXSTRLEN+1];  //以“\0”为串结束标志
```

- 字符串长度可以是小于MAXSTRLEN的任何值, 超过部分将被**截断**



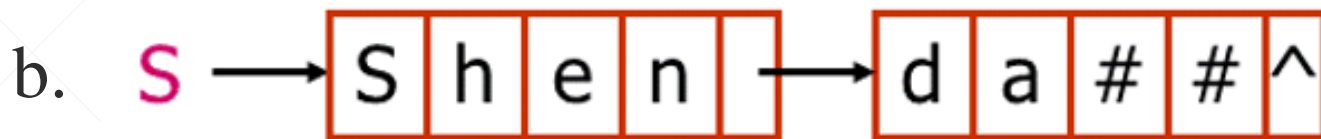
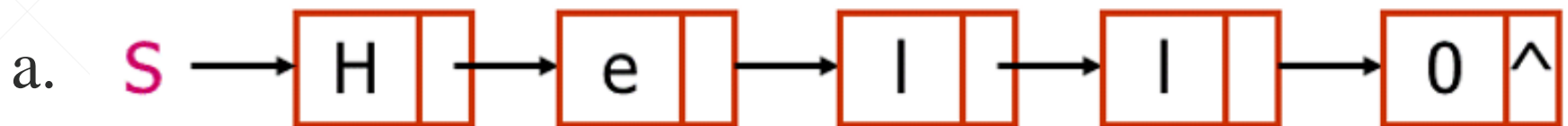
串的堆分配存储表示

- 在程序执行过程中，**动态分配**（**malloc**）一组**地址连续**的存储单元存储字符序列
- 在C语言中，由**malloc()**和**free()**动态分配与回收的存储空间称为堆
- 堆分配存储结构的串既有顺序存储结构的特点，处理方便，操作中对串长又没有限制，更显灵活



串的链式存储表示

- 采用链表方式存储串值
- 每个结点中，可以放一个字符，也可以存放多个字符



a. 结点大小为1的链表 b. 结点大小为4的链表

求子串位置函数 Index(S, T, pos)

- 子串的**定位**操作通常称做串的**模式匹配**

初始条件：串S和T存在，T为非空串（S为主串，T为模式）
串采用**顺序**存储结构，串的第**0位置存放串长**
 $1 \leq \text{pos} \leq \text{StrLength}(S)$

操作结果：若主串S中存在和串T值相同的子串，返回它的主串S中第**pos个字符**之后第一次出现的位置；否则函数值为0

基本思想：从主串S的第**pos个字符**起和模式T的第一个字符比较，若相等，则逐个比较后续字符；否则从主串的**下一个字符**起再重新和模式的字符比较



求子串位置函数 Index(S, T, pos)

- 算法(穷举法):

从主串的**指定位置**开始，将主串与模式 (要查找的子串) 的第一个字符比较

- ❑ 若相等，则继续逐个比较后续字符；
- ❑ 若不相等，从主串的**下一个字符**起再**重新**和模式的字符比较



求子串位置函数

```
■ int Index(Sstring S, Sstring T, int pos){  
    i = pos; j = 1;           // 从第一个位置开始比较  
    while (i<=S[0] && j<=T[0]){ // S[0]、T[0]为串长  
        if (S[i] == T[j]) {++i; ++j} // 继续比较后继字符  
        else {i = i - j + 2; j = 1;} // 指针后退重新开始匹配  
    }  
    if (j > T[0]) return i-T[0]; // 返回与模式第一字符相等的字符在主串中的序号  
    else return 0;               // 匹配不成功  
}
```



求子串位置函数 Index()

- [illegible]

