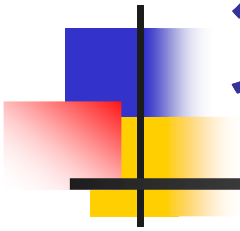
数据结构——串

主讲：张昱

yuzhang@ustc.edu

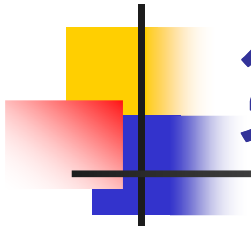
0551-3603804



第四章 串

知识点：串的定义、表示与实现，定位函数

说明：KMP算法移至<算法基础>课中讲



第四章 串

4.1 串类型的定义

- 串

C中的字符串：一对双引号括起的字符序列，如“abcd ef”

一般地，串是由零个或多个字符组成的有限序列

一些概念：串的长度、空串、子串、主串、
字符/子串在串/主串中的位置

如：S=“Data Structure”，S为串名，“Data Structure”为串值，
长度为14；“ata Str”为主串S的子串，它在S中的位置为2

称两个串是相等的，当且仅当这两个串的值相等。

空格串 vs. 空串Ø

是特殊的线性表 1)元素类型为字符；
2)操作对象：个体(字符)与整体(子串)

4.1 串类型的定义-ADT String

- **ADT String** 串的整体操作
  - 赋值 StrAssign(S, “Data Structure”)
  - 复制 StrCopy(T, S) // T<=S, T为“Data Structure”
  - 比较 StrCompare(S, T)
  - 连接 Concat(T, “Data”, “Structure”) //T为“DataStructure”
  - 取子串 SubString(sub, S, 2, 5) // sub为“ata S”
  - 子串在主串中的定位 Index(S, “a”, 3) // 4
  - 子串置换 Replace(S, “a”, “b”) // S为“Dbtb Structure”
  - 子串插入 StrInsert(S, 3, “aha”) // “Daahata Structure”
  - 子串删除 StrDelete(S, 3, 5) // “Daructure”

4.1 串类型的定义-操作间的关系

- 串的最小操作子集

  赋值、求串长、比较、联接、取子串

- 定位函数 Index(S, T, pos)

```
int  Index(String S, String T, int pos) {
   if (pos>0){              // pos的合法性
      n = StrLength(S); m = StrLength(T);  i = pos;
      while( i<=n-m+1) {
         SubString(sub, S, i, m);
         if (StrCompare(sub, T) != 0)   ++i;
         else  return i;   // 返回子串在主串中的位置
      } // while
   } // if
   return 0;              // S中不存在与T相等的子串
} // Index
```


4.2 串的表示和实现-定长顺序存储

- 定长顺序存储表示----顺序映像
  - 类型定义

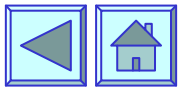
    typedef unsigned char SString[MAXSTRLEN+1];

    约定：1) 下标为0的分量存放串的长度

    或 2)串值后加入一个不计入串长的结束标记字符，如C语言中的'\0'
  - 串联接Concat(&T, S1, S2)

    ∵是定长存储，联接后T的串长为S1和S2串长之和，该长度可能会超出MAXSTRLEN

    ∴分情况处理，超出部分要“截断”



4.2 串的表示和实现-定长顺序存储

- 串联接Concat(&T, S1, S2)
  - S1[0]>=MAXSTRLEN
    T[1..MAXSTRLEN]=S1[1..MAXSTRLEN];
    T[0] = MAXSTRLEN; //S2全被截去
  - S1[0]<MAXSTRLEN && S1[0]+S2[0] > MAXSTRLEN
    T[1..S1[0]]=S1[1..S1[0]];
    T[S1[0]+1.. MAXSTRLEN]=S2[1..MAXSTRLEN-S1[0]];
    T[0]=MAXSTRLEN; //S2部分截断
  - S1[0]+S2[0] <= MAXSTRLEN
    T[1..S1[0]]=S1[1..S1[0]];
    T[S1[0]+1..S1[0]+S2[0]]=S2[1..S2[0]];
    T[0]=S1[0]+S2[0];

4.2 串的表示和实现-定长顺序存储

- 求子串SubString(&Sub, S, pos, len)
  - 合法的pos和len
    pos>0 && len>=0 && pos<= S[0]-len+1
  - 串的复制
    Sub[1..len]=S[pos..pos+len-1];
    Sub[0]=len;
- 评价
  - 串长超出最大长度，约定采用截尾法
  - 串长过小，则串空间浪费较大

4.2 串的表示和实现-堆分配存储

- 堆分配存储表示----顺序映像
  - 类型定义

```
typedef  struct {
     char  *ch;              // 串值所在的存储区的起始地址
     int    length;          // 串长度
} HString;
```

  - 一些操作的实现
    - 基于复制

      利用malloc()分配一块足够的空间，再按要求完成复制；
      如StrAssign(&T,chars), Concat(&T, S1, S2),
      SubString(&Sub, S, pos, len)

4.2 串的表示和实现-堆分配存储

- 堆分配存储表示----顺序映像
  - 一些操作的实现
    - 基于链接
      建立串名和串值的映射关系，
      如: StrAssign(&T, chars)
  - 评价
    - 基于动态存储管理
    - 建立串名和串值的映射关系
    - 处理方便、串值共享

4.2 串的表示和实现-块链存储

- 块链存储表示----链式映像
  - sizeof(char) < sizeof(链域) → 存储密度不高

    存储密度 = 串值所占的存储位/实际分配的存储位
  - 类型定义

```
typedef   struct  Chunk {
      char  ch[CHUNKSIZE];
      struct Chunk  *next;
} Chunk;
typedef   struct {
      Chunk  *head, *tail;   // 串的头和尾指针
      int      curlen;           // 串的当前长度
} LString;
```


4.2 串的表示和实现-块链存储(续)

- **存储密度的影响**

  存储密度小，运算处理方便，但存储占用量大

- **评价**
  - 存储密度较低
  - 块链使串的操作复杂化
- **顺序映像的缺点**
  - 空间利用率低
  - 空间不可扩充，使串的某些操作（联接、置换）受到限制
  - 插入、删除、置换操作带来的移动

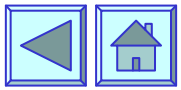
4.3 串的模式匹配算法

- 求子串位置的定位函数 Index(S, T, pos)

  ——模式匹配(T:模式串)

  基于堆分配存储的算法

```
int Index( HString S, HString T, int pos ){
   i = pos;  j = 1;
   while ( i<=S.length && j<=T.length ){
      if ( S.ch[i]==T.ch[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.length) return (i – T.length);
   else return 0;
}
```

4.4 串操作应用举例

- 文本编辑
  - 处理规则：行插入/删除，页插入/删除，……
  - 数据结构：页表、行表(行号，起始地址，长度)
- 建立词索引表
  - 数据结构
    - 词表——书名中的关键词集合
    - 关键词索引表