第4章 串

4.1 串类型的定义

- 字符串是 $n\ (\geq 0)$ 个字符的有限序列，记作：

$$S = \text{“}c_1c_2c_3\ldots c_n\text{”}$$

其中：

- S 是串名字；
- “$c_1c_2c_3\ldots c_n$”是串值；
- c_i 是串中字符；
- n 是串的长度，n = 0 称为空串。

- 例如，S = “Qingdao University”。
- 注意：空串和空白串不同，例如“ ”和“”分别表示长度为1的空白串和长度为0的空串。

- 串中任意个连续字符组成的子序列称为该串的子串，包含子串的串相应地称为主串。
- 通常将子串在主串中首次出现时，该子串首字符对应的主串中的序号，定义为子串在主串中的位置。例如，设A和B分别为

  A = “This is a string” B = “is”

  则 B 是 A 的子串，A 为主串。B 在 A 中出现了两次，首次出现所对应的主串位置是2（从0开始）。因此，称 B 在 A 中的位置为2。
- 特别地，空串是任意串的子串，任意串是其自身的子串。

- 通常在程序中使用的串可分为两种：串变量和串常量。
- 串常量在程序中只能被引用但不能改变它的值，即只能读不能写。通常串常量是由直接量来表示的，例如语句 Error (“overflow”) 中 “overflow”是直接量。但有的语言允许对串常量命名，以使程序易读、易写。如C中可定义

  char path[] = “dir/bin/appl”;

  这里path是一个串常量。
- 串变量和其它类型的变量一样，其取值可以改变。

在C语言中常用的字符串操作

- 字符串初始化
  - **char** name[10] = “Qingdao”;
  - **char** name[ ] = “Qingdao”;
  - **char** name[10] = {‘Q’,‘i’,‘n’,‘g’,‘d’,‘a’,‘o’};
  - **char** name[ ] = {‘Q’,‘i’,‘n’,‘g’,‘d’,‘a’,‘o’,‘\0’};
  - **char** *name = “Qingdao”;
  - **char** name[10];

    name = “Qingdao”; × 因数组名是地址常量

- 单个字符串的输入函数 gets (str)

  例　char name[10];

  gets (name);

- 字符串输出函数 puts (str)

  例　char name[10];

  gets (name);

  puts (name);

- 字符串求长度函数 strlen(str)

  字符串长度不包括 “\0”和分界符

  例　int m = strlen (“University”);

  printf (“%d\n”, m);　//输出10

- 字符串连接函数 strcat (str1, str2)

  例 str1 “Qingdao \0” //连接前

  str2 “University\0” //连接前

  str1 “QingdaoUniversity\0” //连接后

  str2 “University\0” //不变

- 字符串比较函数 strcmp (str1, str2)

  //从两个字符串第 1 个字符开始，逐个对应字符进

  //行比较，全部字符相等则函数返回0，否则在不

  //相等字符处停止比较，函数返回其差值

  //（比较基于 ASCII代码）

  例 str1 “University” i 的代码值105

  str2 “Universal” a 的代码值97，差8

4.2 串的表示和实现

- 除 C 语言提供的字符串库函数外，可以自定义字符串。

  适用于自定义字符串数据类型的有三种存储表示：

  - 定长顺序存储表示；
  - 堆分配存储表示；
  - 块链存储表示。

4.2.1 定长顺序存储表示

- 顺序串：使用静态分配的字符数组存储字符串中的字符序列。
  - 字符数组的长度预先用 MAXSTRLEN 指定，一旦空间存满不能扩充。
- 有两种实现定长顺序存储表示：
  - 字符存放于字符数组的 0～ MAXSTRLEN-1 单元，另外用整数length记录串中实际存放的字符个数；
  - 字符存放于字符数组的 1～ MAXSTRLEN-1 单元，用 0 号单元记录串中实际存放的字符个数。
  - 本讲座采用前者（即按照C语言方式）。

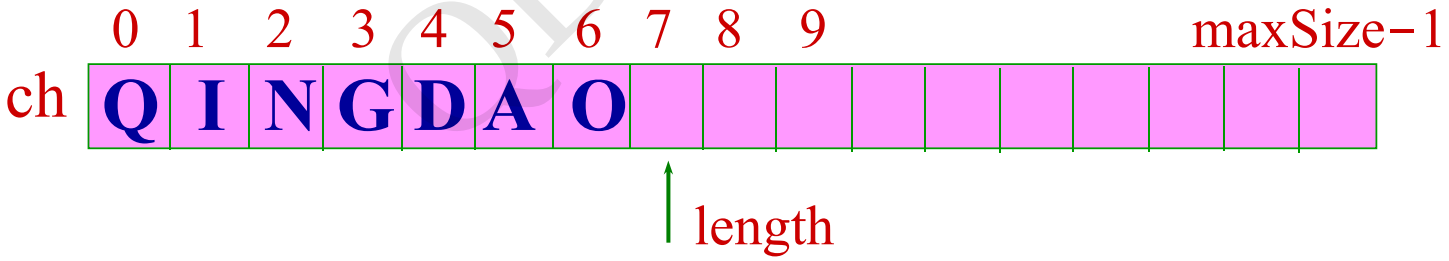
- 按照C语言规定，在字符串值最后有一个特殊的“\0”表示串值的结束。因此，在存放串值时要求为它留一个位置。
- 定长存储表示的定义如下：

```
#define   MAXSTRLEN   256          //顺序串的预设长度
typedef   struct {                 //顺序串的定义
     char SString[MAXSTRLEN];      //存储字符数组
     int  length;                  //串中实际字符个数
} SqString;
```

4.2.2 堆分配存储表示

- **堆式串**：字符数组的存储空间是通过malloc( )函数动态分配的。串的最大空间数 MAXSTRLEN 和串中实际字符个数length保存在串的定义中。
- 可以根据需要，随时改变字符数组的大小。

```
//---------- 串的堆分配存储表示 -------
#define MAXSTRLEN  256;
typedef struct {
    char *ch;              //串的存储数组
    int maxSize;           //串数组的最大长度
    int length;            //串的当前长度
} HString;
```



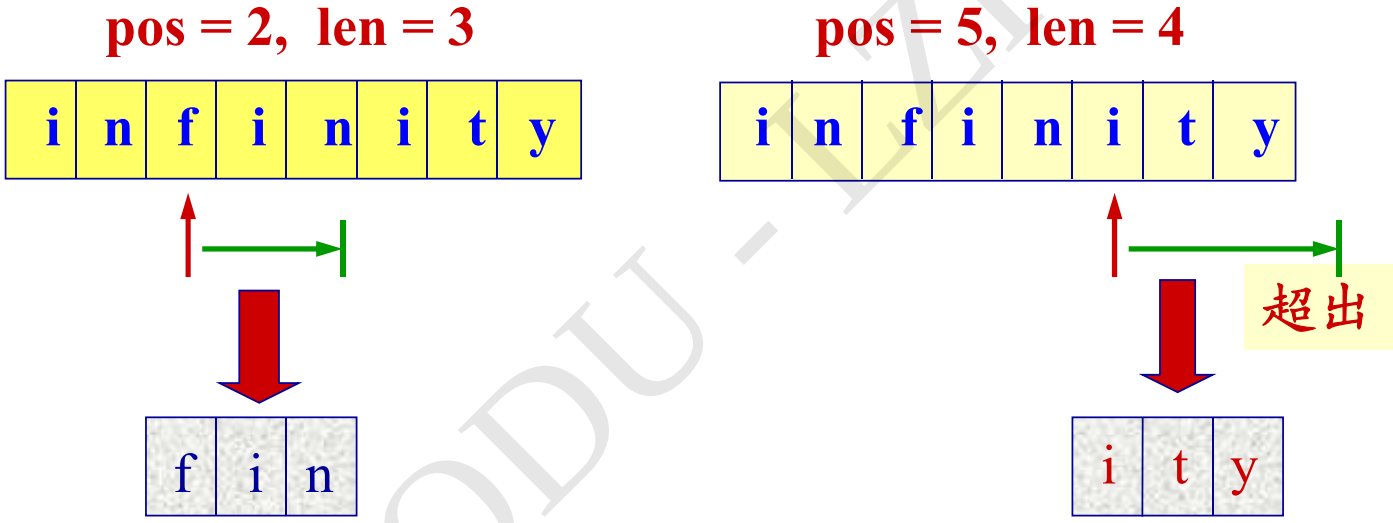
堆存储部分操作的实现

（1）初始化空串算法

```
void initString (Hstring &S)
{  //初始化：创建字符串 S 的存储空间并置空串
    //分配字符数组空间
    S.ch = (char *)malloc(MAXSTRLEN*sizeof(char));
    if ( S.ch == NULL ) exit (1);    //判断分配成功与否
    S.ch[0] = '\0';                  //置空串
    S.maxSize = MAXSTRLEN;           //置串的最大字符数
     S.length = 0;                   //实际字符数置0
}
```

（2）提取子串算法

提取子串的算法示例



pos+len-1
≤ curLen-1
可以全部提取

pos+len -1
≥ curLen
只能从pos取到串尾

（2）提取子串算法

```
HString subString ( HString& s, int pos, int len )
{
   //在串 s 中连续取从 pos 开始的 len 个字符，构成子串
   //返回。若提取失败则函数返回NULL
  HString tmp;
  //创建子串空间
  tmp.ch =(char *)malloc(MAXSTRLEN*sizeof(char));
  tmp.maxSize = MAXSTRLEN;
  //参数不合理，返回空串
  if (pos<0||len<0||pos+len-1 >= s.maxSize )
  {
      tmp.length = 0;
      tmp.ch[0] = '\0';
  }
   else {
```

```
    //若提取个数超出串尾，修改个数
    if ( pos+len-1 >= s.length )
        len = s.length-pos;
    for ( int i = 0, j = pos; i < len; i++, j++ )
        tmp.ch[i] = s.ch[j];          //复制子串的字符
    tmp.length = len;  tmp.ch[len] = '\0';
  }
  return tmp;                         //返回复制的子串
}
```

- 【例】串 st = “university”，pos = 3，len = 4

  使用示例　HString t = subString(st, 3, 4)

  提取子串　t = “vers”

（3）串的连接算法

```
void concat ( HeapString &s, HeapString &t )
{  // 函数将串t复制到串s之后，通过串s返回结果，串t不变。
    if (s.length+t.length <= s.maxSize )
    {  //原空间可容纳连接后的串
       for(int i = 0; i < t.n; i++)
           s.ch[s.length+i] = t.ch[i];    //串t复制到串s后
       s.length = s.length+t.length;
       s.ch[s.length] = '\0';
    }
    else { //原空间容不下连接后的串
        char *tmp = s.ch;
       s.maxSize = s.length + t.length;
```

```
        //按新的大小分配存储空间
        s.ch =(char*)malloc((s.maxSize+1)*sizeof(char));
        strcpy (s.ch, tmp);                    //复制原串 s 数组
        strcat (s.ch, t.ch);              //连接串 t 数组
        s.length = s.length+t.length;
        free ( tmp );
    }
}
```

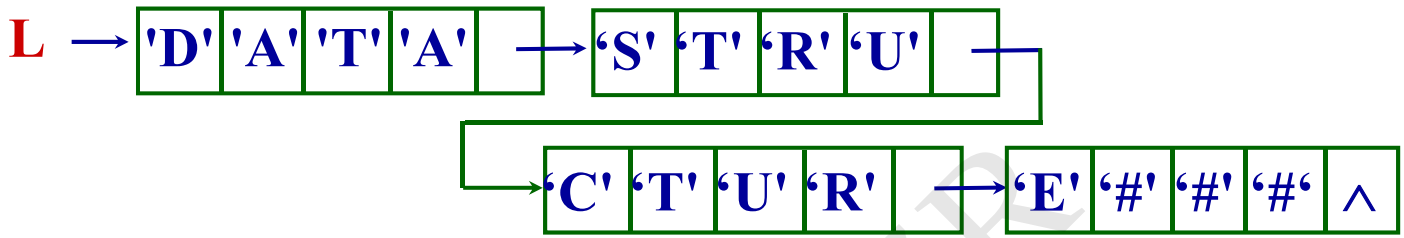
- 【例】串 st1 = “beijing ”， st2 = “university”,
  使用示例　concat (st1, st2);
  连接结果　st1 = “beijing university”
  　　st2 = “university”

4.2.3 串的块链存储表示

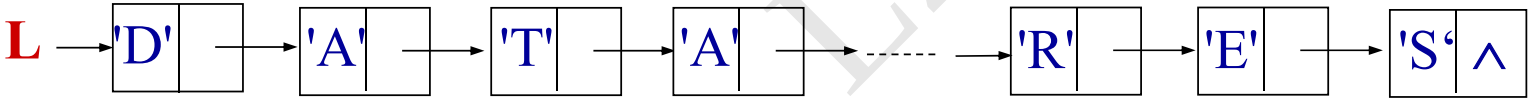
- 使用单链表作为字符串的存储表示，此即字符串的链接存储表示。
- 链表的每个结点可以存储 1 个字符，称其“块的大小”为 1，也可以存储 n 个字符，称其“块的大小”为 n。
- 定义存储密度为：

$$\text{存储密度} = \frac{\text{串值所占的存储位}}{\text{实际分配的存储位}}$$

- 显然，存储密度越高，存储利用率越高。



(a) 结点大小为4



(b) 结点大小为1

- 结点大小为 4 时，存储利用率高，但操作复杂，需要析出单个字符；结点大小为 1 时，存储利用率低，但操作简单，可直接存取字符。
- 块链存储表示一般带头结点，设置头、尾指针。

块链存储表示的结构定义

```
#define blockSize 4   //由使用者定义的结点大小
typedef struct block {      //链表结点的结构定义
  char ch[blockSize];
  struct block *next;
} Chunk;

typedef struct {  //链表的结构定义
  Chunk *first, *last;    //链表的头指针和尾指针
  int length;             //串的当前长度
} LString;
```

END