数据结构与算法

第四章　串

串类型的定义
串的表示和实现
串的模式匹配算法

第一节 串类型的定义

- 串(String)，是由零个或多个字符组成的有限序列，也称字符串。一般记为：

  $s = 'a_1a_2a_3...a_n'$

  其中s称为串名，是用单引号括起来的字符序列

  C语言中字符串是用双引号括起来的字符序列

  a_i ($1 \le i \le n$)是字符，可以是字母、数字或其它符号

  n是串s的长度。
- 零个字符的串称为空串null ->C语言NULL
- 串中任意个连续的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。通常称字符在序列中的序号为该字符在串中的位置。子串在主串中的位置则以子串的第一个字符在主串中的位置来表示。

串的例子

- 称两个串是相等的，当且仅当这两个串的值相等。也就是说，只有当两个串的长度相等，并且各个对应位置的字符都相等时才相等。

  a = 'WU', b = 'HAN'

  c = 'WUHAN' d = 'WU HAN'

- 值得一提的是，串值必须用一对单引号括起来，但单引号本身不属于串，它的作用只是为了避免与变量名或数的常量混淆而已。

串的操作

基本操作：

StrAssign (&T, chars)

初始条件：chars 是字符串常量。

操作结果：生成一个其值等于 chars 的串 T。

StrCopy (&T, S)

初始条件：串 S 存在。

操作结果：由串 S 复制得串 T。

StrEmpty (S)

初始条件：串 S 存在。

操作结果：若 S 为空串，则返回 TRUE，否则返回 FALSE。

StrCompare (S, T)

初始条件：串 S 和 T 存在。

操作结果：若 S > T，则返回值 > 0；
若 S = T，则返回值 = 0；
若 S < T，则返回值 < 0。

StrLength (S)

初始条件：串 S 存在。

操作结果：返回 S 的元素个数，称为串的长度。

ClearString (&S)

初始条件：串 S 存在。

操作结果：将 S 清为空串。

Concat (&T, S1, S2)

初始条件：串 S1 和 S2 存在。

操作结果：用 T 返回由 S1 和 S2 联接而成的新串。

SubString (&Sub, S, pos, len)

初始条件：串 S 存在，$1 \leqslant pos \leqslant StrLength(S)$
且 $0 \leqslant len \leqslant StrLength(S) - pos + 1$。

操作结果：用 Sub 返回串 S 的第 pos 个字符起
长度为 len 的子串。

Index (S, T, pos)

初始条件：串 S 和 T 存在，T 是非空串，
$1 \leqslant pos \leqslant StrLength(S)$。

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

Replace (&S, T, V)

初始条件：串 S，T 和 V 存在，T 是非空串。

操作结果：用 V 替换主串 S 中出现的所有与 T 相等
的不重叠的子串。

StrInsert (&S, pos, T)

初始条件：串 S 和 T 存在，$1 \leqslant pos \leqslant StrLength(S) + 1$。

操作结果：在串 S 的第 pos 个字符之前插入串 T。

StrDelete (&S, pos, len)

初始条件：串 S 存在，$1 \leqslant pos \leqslant StrLength(S) - len + 1$。

操作结果：从串 S 中删除第 pos 个字符起长度为 len 的子串。

DestroyString (&S)

初始条件：串 S 存在。

操作结果：串 S 被销毁。

串的基本操作

- **最小操作子集：**StrAssign、StrCompare、StrLength、Concat、SubString

```
int Index(String S, String T, int pos) {
    if(pos > 0) {
        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;
        } //while
    } // if
    return 0;
} //Index
```

第二节 串的表示和实现

- **定长顺序存储表示**

```
#define MAXSTRLEN 255;
typedef unsigned char SSTRING[MAXSTRLEN+1];
```

```
Status Concat(SString &T, SString S1, SString S2) {
    if(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];
    }
    else if(S1[0] < MAXSTRLEN) {              //截断
        …
    }
    …
}
```

第三节 串的堆分配存储表示

- **堆分配存储表示**

```
Typedef struct {
   char *ch;
   int length;
} Hstring;
```

```
Status Concat(Hstring &T, HString S1, HString S2) {
    if(T.ch) free(T.ch);
    if(!(T.ch = (char *)malloc((S1.length+S2.length)*sizeof(char))))
        exit(OVERFLOW);
    T.ch[0..S1.length-1] = S1.ch[0..S1.length-1];
    T.length = S1.length + S2.length;
    T.ch[S1.length..T.length-1] = S2.ch[0..S2.length-1];
    return OK;
}
```

```
Status StrInsert(HString &S, int pos, HString T) {
    if(pos<1||pos>S.length+1) return ERROR;
    if(T.length) {
        if(!(S.ch = (char *)realloc(S.ch, S.length+T.length)*sizeof(char))))
            exit(OVERFLOW);
        for(i=S.length-1; i>=pos-1; --i)
            S.ch[i+T.length] = S.ch[i];
        S.ch[pos-1..T.length-2] = T.ch[0..T.length-1];
        S.length += T.length;
    }
    return OK;
}
```

串的块链存储表示

```
#define CHUNKSIZE 80
typedef struct Chunk {
    char ch[CHUNKSIZE];
    struct Chunk *next;
} Chunk;
typedef struct {
    Chunk *head, *tail;
    int curlen;
}LString;
```

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

串的模式匹配算法

- 串的模式匹配：子串的定位操作。
- 子串称为模式串。
- 定长顺序存储结构的匹配算法

```
int Index(SString S, SString T, int pos) {
    i = pos; j = 1;
    while(i<=S[0] && j<=T[0]) {
        if(S[i]==T[j]) {++i; ++j;}
        else {i = i – j + 2;  j = 1;}
    }
    if(j>T[0]) return i-T[0];
    else return 0;
}
```