链滴

摸一摸数据结构（串）

作者：stillwarter

原文链接：https://ld246.com/article/1630589857371

来源网站：链滴

计算机上的非数值处理对象基本上是字符串数据。在较早的程序设计中，字符串是作为输入和输出的量出现。

字符串一般简称为串。在汇编和语言的编译程序中，源程序和目标程序都是字符串数据。

串的定义和实现

定义

串（string）是零个或多个字符组成的有限序列，记为 str='abcd...'。

串中字符数目是串的长度，0 个字符称为空串，其长度为 0。

子串

串中任意连续的字符组成称为该串的子串

主串

包含子串的串相应的称为主串

空格串

由一个或多个空格组成的串 ‘’ 称为空格串。

定长的顺序串实现

数据结构

```
#include <stdio.h>
#include <string.h>
#include "Status.h"
//顺序串相关操作
#define maxstrlen 255
typedef unsigned char sstring[maxstrlen+1];
```

串我们直接用字符数组去表示就行。

注意我们定义的串 sstring[0]存放的是串的长度

串的赋值

```
Status strassignsq(sstring T,const char *chars){
    int i, len;

    len = strlen(chars);

    if(len>maxstrlen)
        return ERROR;
```

```
    else
    {
        T[0] = len;
        for(i=1; i<=len; i++)
            T[i] = chars[i-1];

        return OK;
    }
};//生成一个其值等于常量chars的串T（串赋值）
```

对目标串赋值，串入一个常量串，先判断常量串长度是否超过串结构的最大规模，再进行循环赋值。

串的复制，判空，长度，清空

```
void strcopysq(sstring T,sstring S){
    int i;

    for(i=0; i<=S[0]; i++)
        T[i] = S[i];
};
Status stremptysq(sstring S){
    if(S[0]==0)
        return TRUE;
    else
        return FALSE;
};
int strlensq(sstring S){
    return S[0];
};

void clearstrsq(sstring S){
    S[0] = 0;
};
```

略

串比较

```
int strcomparesq(sstring S,sstring T){
    int i = 1;

    while(i<=S[0] && i<=T[0])
    {
        if(S[i]==T[i])
            i++;
        else
            return S[i] - T[i];
    }

    return S[0] - T[0];
};若S>T，返回值>0；若S<T，返回值<0；否则，返回值=0
```

对两个串进行判断是否一样，若为 0，则串等。

串连接

```
Status concatsq(sstring T,sstring S1,sstring S2){
    int i;

    for(i=1; i<=S1[0]; i++)
        T[i] = S1[i];

    if(S1[0]+S2[0]<=maxstrlen)
    {
        for(i=1; i<=S2[0]; i++)
            T[S1[0]+i] = S2[i];

        T[0] = S1[0]+S2[0];

        return OK;
    }
    else
    {
        for(i=1; S1[0]+i<=maxstrlen; i++)
            T[S1[0]+i] = S2[i];

        T[0]=maxstrlen;

        return ERROR;
    }
};//用T返回由S1和S2联接而成的新串
```

传入串 T，s1，s2；这里用 T 来输出连接后的串。这里要处来连接后的长度超出了规模，因为是定长，所以无法变长。

若连接后不超出规模，则在 s1 后面接上 s2 就行；若超出，则循环的上界需要变为 maxstrken。

子串

```
Status substrsq(sstring sub,sstring S,int pos,int len){
    int i;

    if(pos<1 || pos>S[0] || len<0 || pos+len-1>S[0])
        return ERROR;

    for(i=1; i<=len; i++)
        sub[i] = S[pos+i-1];

    sub[0] = len;

    return OK;
}//用Sub返回串S的第pos个字符起长度为len的子串
```

传入值是串 sub（用于返回子串），主串 S，子串长度 len 和位次 pos。

首先对传入值进行判断是否合理。之后对子串进行截取。

串的模式匹配

```
int indexsq1(sstring S,sstring T,int pos){
    int s, t;
    sstring sub;

    if(pos>0)
    {
        s = strlensq(S);
        t = strlensq(T);

        if(s && t)
        {
            while(pos+t-1<=s)
            {
                substrsq(sub, S, pos, t);

                if(!strcomparesq(sub, T))
                    return pos;

                pos++;
            }
        }
    }

    return 0;
};//返回T在S中第pos个字符后第一次出现的位置，不存在则返回0。

int indexsq2(sstring S,sstring T,int pos){
    int i = pos;
    int j = 1;

    if(pos<1)    return 0;

    while(i<=S[0] && j<=T[0])
    {
        if(S[i]==T[j])
        {
            i++;
            j++;
        }
        else
        {
            i = i - (j-1) +1;
            j = 1;
        }
    }

    if(j>T[0] && T[0]) return i-T[0];
    else return 0;

};//串的模式匹配算法，不依赖其他操作的匹配算法
```

第一个比较简单，通过已有的功能函数，根据 T 串的长度在 S 生成对应位次的子串然后对比即可。

我们主要看第二个。传入的是串 S，T 和位次。

判断 T 是否是 S 的子串，不使用其他操作的情况下，只能通过对比串元素来得到答案。

对位次进行合理性判断，然后进入循环（判断是否存在子串关系）；

以位次 1 为例（从第一位进行判断），若相等，则更新位次为下一位；再次判断。

若不等，则需要重来（T 串与 S 串若第一个字母相等而第二个不等则需要重头开始判断，所以 j 需要写为 1）。

那么若 T，S 前 2 个字母相等，最后一个字母不懂，则又需要重新判断，我们称 i 是无效的更新。需回到原来的后一位在次判断。猫猫愚钝，是穷举才明白一点这个关系。这也是编码的魅力，用最简洁话就能完成复杂的操作，构筑有力的工具！

这里可能有点难度，需要揣摩一下。

删除，插入与输出

```
Status strdelsq(sstring S,int pos,int len){
    int i;

    if(pos<1 || pos+len-1>S[0] || len<0)
        return ERROR;

    for(i=pos+len; i<=S[0]; i++)
        S[i-len] = S[i];

    S[0] -= len;

    return OK;
};//从串S中删除第pos个字符起长度为len的子串。
Status strinsertsq(sstring S,int pos,sstring T){
    int i;

    if(pos<1 || pos>S[0]+1 || S[0]+T[0]>maxstrlen)
        return ERROR;

    for(i=S[0]; i>=pos; i--)
        S[i+T[0]] = S[i];

    for(i=pos; i<=pos+T[0]-1; i++)
        S[i] = T[i-pos+1];

    S[0] += T[0];

    return OK;
};//在串S的第pos个字符之前插入串T。可以完全插入返回OK,否则返回ERROR
void strprintsq(sstring S){
    int i;
    for(i=1; i<=S[0]; i++)
        printf("%c", S[i]);
};
```

略

替换

```
Status replacesq(sstring S,sstring T,sstring V){
    int i;

    i = indexsq2(S, T, 1);                          //寻找第一个匹配的位置

    while(S[0]-T[0]+V[0]<=maxstrlen &&& i)              //有匹配的字符串且可以被完全替换
    {

        strdelsq(S, i, T[0]);                       //删除T
        strinsertsq(S, i, V);                         //插入V

        i += V[0];                                  //i切换到下一个位置

        i = indexsq2(S, T, i);                      //定位下一个匹配的字符串
    }

    if(i==0)                                        //S中的T已全部被替换
        return OK;
    else                                           //S中尚有T，但是V已经插不进去了
        return ERROR;
};//用V替换主串S中出现的所有与T相等的不重叠的子串，可以被完全替换才返回OK
```

对之间功能函数的组装。

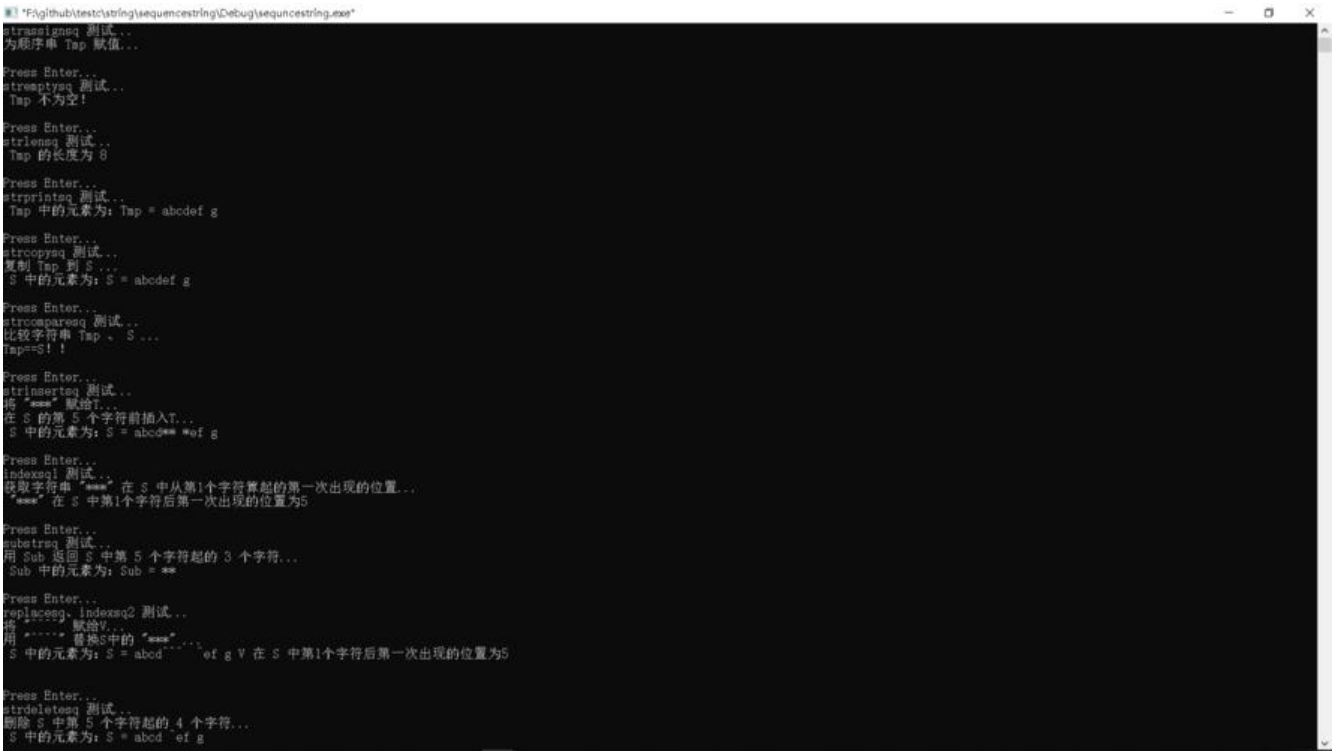
测试

```
#include <stdio.h>
#include "sequncestring.h"
int main()
{
    char *chars = "abcdef g";
    char *t = "** *";
    char *v = "^^^  ^";
    sstring S, Tmp, T, V, Sub;
    int i,j;

    printf("strassignsq 测试...\n");
    {
        printf("为顺序串 Tmp 赋值...\n");
        strassignsq(Tmp, chars);
        printf("\n");
    }
    PressEnter;


    printf("stremptysq 测试...\n");
    {
        stremptysq(Tmp) ? printf(" Tmp 为空！！\n") : printf(" Tmp 不为空！\n");
```

```
    printf("\n");
}
PressEnter;

printf("strlensq 测试...\n");
{
    i = strlensq(Tmp);
    printf(" Tmp 的长度为 %d \n", i);
    printf("\n");
}
PressEnter;

printf("strprintsq 测试...\n");
{
    printf(" Tmp 中的元素为：Tmp = ");
    strprintsq(Tmp);
    printf("\n\n");
}
PressEnter;

printf("strcopysq 测试...\n");
{
    printf("复制 Tmp 到 S ...\n");
    strcopysq(S, Tmp);
    printf(" S 中的元素为：S = ");
    strprintsq(S);
    printf("\n\n");
}
PressEnter;

printf("strcomparesq 测试...\n");
{
    printf("比较字符串 Tmp 、 S ...\n");
    i = strcomparesq(Tmp, S);
    i==0 ? printf("Tmp==S！！\n") : (i<0 ? printf("Tmp<S！！\n") : printf("Tmp>S！！\n"));
    printf("\n");
}
PressEnter;

printf("strinsertsq 测试...\n");
{
    printf("将 \"***\" 赋给T...\n");
    strassignsq(T, t);
    printf("在 S 的第 5 个字符前插入T...\n");
    strinsertsq(S, 5, T);
    printf(" S 中的元素为：S = ");
    strprintsq(S);
    printf("\n\n");
}
PressEnter;

printf("indexsq1 测试...\n");
{
    printf("获取字符串 \"***\" 在 S 中从第1个字符算起的第一次出现的位置...\n");
```

```
    i = indexsq1(S, T, 1);
    printf(" \"***\" 在 S 中第1个字符后第一次出现的位置为%d\n", i);
    printf("\n");
}
PressEnter;

printf("substrsq 测试...\n");
{
    printf("用 Sub 返回 S 中第 5 个字符起的 3 个字符...\n");
    substrsq(Sub, S, 5, 3);
    printf(" Sub 中的元素为：Sub = ");
    strprintsq(Sub);
    printf("\n\n");
}
PressEnter;

printf("replacesq、indexsq2 测试...\n");
{
    printf("将 \"^^^^\" 赋给V...\n");
    strassignsq(V, v);
    printf("用 \"^^^^\" 替换S中的 \"***\" ...\n");
    replacesq(S, T, V);
    printf(" S 中的元素为：S = ");
    strprintsq(S);
    j = indexsq2(S, V, 1);
    printf(" V" 在 S 中第1个字符后第一次出现的位置为%d\n", j);
    printf("\n\n");
}
PressEnter;

printf("strdeletesq 测试...\n");
{
    printf("删除 S 中第 5 个字符起的 4 个字符...\n");
    strdelsq(S, 5, 4);
    printf(" S 中的元素为：S = ");
    strprintsq(S);
    printf("\n\n");
}
PressEnter;

printf("clearstringsq 测试...\n");
{
    printf("清空 S 前：");
    stremptysq(S) ? printf(" S 为空！！\n") : printf(" S 不为空！\n");
    clearstrsq(S);
    printf("清空 S 后：");
    stremptysq(S) ? printf(" S 为空！！\n") : printf(" S 不为空！\n");
    printf("\n");
}
PressEnter;

printf("concatsq 测试...\n");
{
    printf("连接 T 和 V 形成 Tmp ...\n");
```

```
        concatsq(Tmp, T, V);
        printf(" Tmp 中的元素为：Tmp = ");
        strprintsq(Tmp);
        printf("\n\n");
    }
    PressEnter;

    return 0;
}
```



串的其他储存

1. 定长顺序存储表示（就是定长串如上）
2. 堆分配储存表示

```
typedef struct{
    char *ch;
    int len;
}hstring;
```

堆存储任然是一组地址连续的存储单元，但他们存储空间是在执行程序中动态分配的。（鸽）

3. 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;

类似于线性表的链式存储，也看采用链表方式存储串值。（鸽）

串的模式匹配

1. 求子串位置的定位函数（顺序串已实现）
2. kmp 算法（鸽）以及进一步优化

串的应用

串可以用于文本编辑以及建立词索引，有机会试试实现
at2

思考

这篇没有思考，串的应用很多，其模式匹配在很多地方都有应用，而且这是我们利用计算机算力的有手段，字符不仅仅是表达字面意思，更可以压缩图片，声音等信息，通过转码再呈现出来。

这是时代进步的产物，这也是未来的基石。

顺便把上节的问题回答了

1. 很明显，栈与队限制条件不一样，一个先进后出，一个先进先出。
2. 根据卡兰特数， (1/4)*6*5*4/(3*2*1)=5，所以有 5 中不同的排列。

ps: 摸一摸数据结构(目录)