§4. 串

4.1. 串类型的定义

串(字符串)是由零个或多个字符组成的有限序列

★ 一般记为 $s='a_1a_2...a_n'$ $(n\geq 0)$

或 $s=''a_1a_2...a_n''$ $(n\geq 0)$

★ 形式化定义中，单双引号无区别，但C/C++语言中单双引号有严格的区别

(单-字符 双-字符串)

★ 基本术语

串的长度	：	串中字符的个数
串的值	：	字符序列
子串	：	串中任意连续字符组成的子序列
主串	：	包含子串的串
字符在串中的位置	：	字符在序列中的序号
空串	：	字符个数为0的串，记作∅(null string)
空格串	：	由若干(1..n)空格组成的串
串相等	：	当且仅当两个串的值相等 { 长度相等；对应位置字符相等

★ 串与线性表的区别

- 串是元素为字符集合的线性表
- 串一般以子串/整体(多个元素)做为操作对象线性表一般以单个元素做为操作对象

★ 串的形式化定义及基本操作

P.71 - 72

§4. 串

4.2. 串的表示和实现

4.2.1. 定长顺序存储表示

P.73 定义

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

★ 字符串长度的表示方法有两种 ┌ [0]中存放长度，[1..255]存放串值
                                └ 以'\0'等形式隐形表示长度('\0'不是合法的串值)

★ unsigned char可当做1字节整数，范围 0 - 255，即这种方式表示的字符串最大不超过255

★ 数组形式，不方便扩充

★ 淘汰此方法，不做任何进一步的讨论

4.2.2. 堆分配存储表示

P.75 定义

```
typedef struct {
    char *ch;
    int length;
} HString;
```

★ 与顺序表、顺序栈相同，将动态内存申请空间当做数组来使用，并可按需扩充

★ 因为C/C++的动态内存是存放在称为“堆”的自由存储区，因此称为堆分配

★ C语言方式的定义与实现

```
/* hstring.h 的组成 */

/* P.10 的预定义常量和类型 */
#define TRUE                1
#define FALSE               0
#define OK                  1
#define ERROR               0
#define INFEASIBE -1
#define LOVERFLOW -2  //避免与<math.h>中的定义冲突

typedef int Status;

/* P.75 串的堆分配存储表示 */
typedef struct {
    char *ch;
    int   length;
} HString;
```

/* hstring.h 的组成 */

/* P.71 - 72 串的形式化定义 */

暂时隐藏

```
Status StrAssign(HString *T, char *chars);
Status StrCopy(HString *T, HString S);
Status StrEmpty(HString S);
int    StrCompare(HString S, HString T);
int    StrLength(HString S);
Status ClearString(HString *S);
Status Concat(HString *T, HString S1, HString S2);
Status SubString(HString *Sub, HString S, int pos, int len);
int    Index(HString S, HString T, int pos);
Status Replace(HString *S, HString T, HString V);
Status StrInsert(HString *S, int pos, HString T);
Status StrDelete(HString *S, int pos, int len);
Status DestroyString(HString *S);
```

问题：定义了串变量后，最先执行哪个函数？

答：StrAssign

问题：哪个函数用来遍历(打印)？

答：未提供

```
/* hstring.c 的组成 */
#include <stdio.h>
#include <stdlib.h>                    //malloc/realloc函数
#include <unistd.h>                    //exit函数
#include "hstring.h"                   //形式定义

/* ---------------------------------------- */
/* 规定所有的实现过程中不能用到现有的字符串函数 */
/* ---------------------------------------- */
```

```
/* hstring.c 的组成 */

/* 生成串(chars是一个符合C/C++规则的字符串，即以尾零结束) */
Status StrAssign(HString *T, char *chars)
{   int   len, i;
    char *c;
    /* 如果原来有空间则释放 */
    if (T->ch)
        free(T->ch);
    /* 求chars的长度，此处必须符合C/C++的语法规则 */
    for(len=0, c=chars; chars && *c; len++, c++)
        ;
    if (!len) {  //chars长度为0（chars是NULL或"")
        T->ch = NULL;
        T->length = 0;
        }
    else { //chars长度不是0
        /* 申请空间（没有预留尾零的位置）*/
        if (!(T->ch = (char *)malloc(len*sizeof(char))))
            exit(LOVERFLOW);
         /* 逐个字符赋值
            书上是 T.ch[0..len-1] = chars[0..len-1] */
        for(i=0; i<len; i++)
            T->ch[i] = chars[i];
        /* 置长度 */
        T->length = len;
        }
    return OK;
}
```

```
int main()
{  HString h;
   StrAssign(&h, "abc");
   StrAssign(&h, "");
   StrAssign(&h, NULL);
   ...
}
```

chars!=NULL && *c!='\0'

问：这句存在什么问题？
答：若初始调用此函数，则T->ch
可能是未定值，直接free会错

```
HString t1;  //对
int main()
{   HString t2={NULL, 0};//对
    HString t3; //错
    ...
}
```

问:假设允许使用字符串处理函数，能否用strcpy?
答:不行，因为T->ch没有预留尾零的位置

```
/* hstring.h 的组成 */
/* P.71 - 72 串的形式化定义 */
Status StrInit(HString *T);
Status StrVisit(HString T, char *title);
Status StrAssign(HString *T, char *chars);
Status StrCopy(HString *T, HString S);
Status StrEmpty(HString S);
int    StrCompare(HString S, HString T);
int    StrLength(HString S);
Status ClearString(HString *S);
Status Concat(HString *T, HString S1, HString S2);
Status SubString(HString *Sub, HString S, int pos, int len);
int    Index(HString S, HString T, int pos);
Status Replace(HString *S, HString T, HString V);
Status StrInsert(HString *S, int pos, HString T);
Status StrDelete(HString *S, int pos, int len);
Status DestroyString(HString *S);
```

问题：定义了串变量后，最先执行哪个函数？

答：StrAssign

问题：哪个函数用来遍历(打印)？

答：未提供

```
/* hstring.c 的组成 */

/* 初始化串，补充，必须要有，否则StrAssign会出错 */
Status StrInit(HString *T)
{
    T->ch     = NULL;
    T->length = 0;

    return OK;
}
```

```
/* hstring.c 的组成 */
/* 打印串，补充，便于观察操作后的结果
   以 <串名>:<内容>:<长度> 的格式打印 */
Status StrVisit(HString T, char *title)
{   int i;
    if (!T.length) //空串
        printf("<%s>:<NULL>:<0>\n", title);
    else { //非空串
        printf("<%s>:", title);
        for (i=0; i<T.length; i++)
            printf("%c", T.ch[i]);
        printf(":<%d>\n", T.length);
        }
    return OK;
}
```

假设HString中是"Hello"
调用StrVisit(T, "串T");
则打印为
<串T>:Hello:<5>

问:能否直接
printf("%s",T.ch);
的方式打印?
答:

```
/* hstring.c 的组成 */

/* 串拷贝 */
Status StrCopy(HString *T, HString S)
{   int i;
    /* 释放T原有的空间 */
    if (T->ch)
        free(T->ch);

    if (!S.length) { //源串为空
        T->ch = NULL;
        T->length = 0;
        }
    else { //源串不为空
        /* 申请空间（没有预留尾零的位置）*/
        if (!(T->ch=(char *)malloc(S.length*sizeof(char))))
            exit(LOVERFLOW);
        /* 逐个字符赋值 */
        for(i=0; i<S.length; i++)
            T->ch[i] = S.ch[i];
        /* 置长度 */
        T->length = S.length;
        }
    return OK;
}
```

处理方式与StrAssign相同，
一个是HString = char*
一个是HString = HString

同前理，不能用strcpy
后面都相同，不再重复

```
/* hstring.c 的组成 */

/* 判断串是否为空 */
Status StrEmpty(HString S)
{
    if (S.length==0)
        return TRUE;
    else
        return FALSE;
}
```

```
/* hstring.c 的组成 */
/* 串比较(要求是 >0、0、<0即可，具体是多少无要求) */
int StrCompare(HString S, HString T)
{
    int i;

    /* 若两串均未结束则循环，逐个比较字符 */
    for(i=0; i<S.length && i<T.length; i++)
        if (S.ch[i]!=T.ch[i])
            return S.ch[i] - T.ch[i]; //对应位置不等
                                      //字符ASCII码差值
    /* 包括了长度相等、小于、大于等三种情况 */
    return S.length - T.length;  //返回长度的差值
}
```

例1:"horse"和"house" -3

例2："abcd" 和 "abcd" 0
例3："abcde" 和 "abcd" 1
例4："abcd" 和 "abcde" -1

```
/* hstring.c 的组成 */

/* 求串长 */
int StrLength(HString S)
{
    return S.length;
}
```

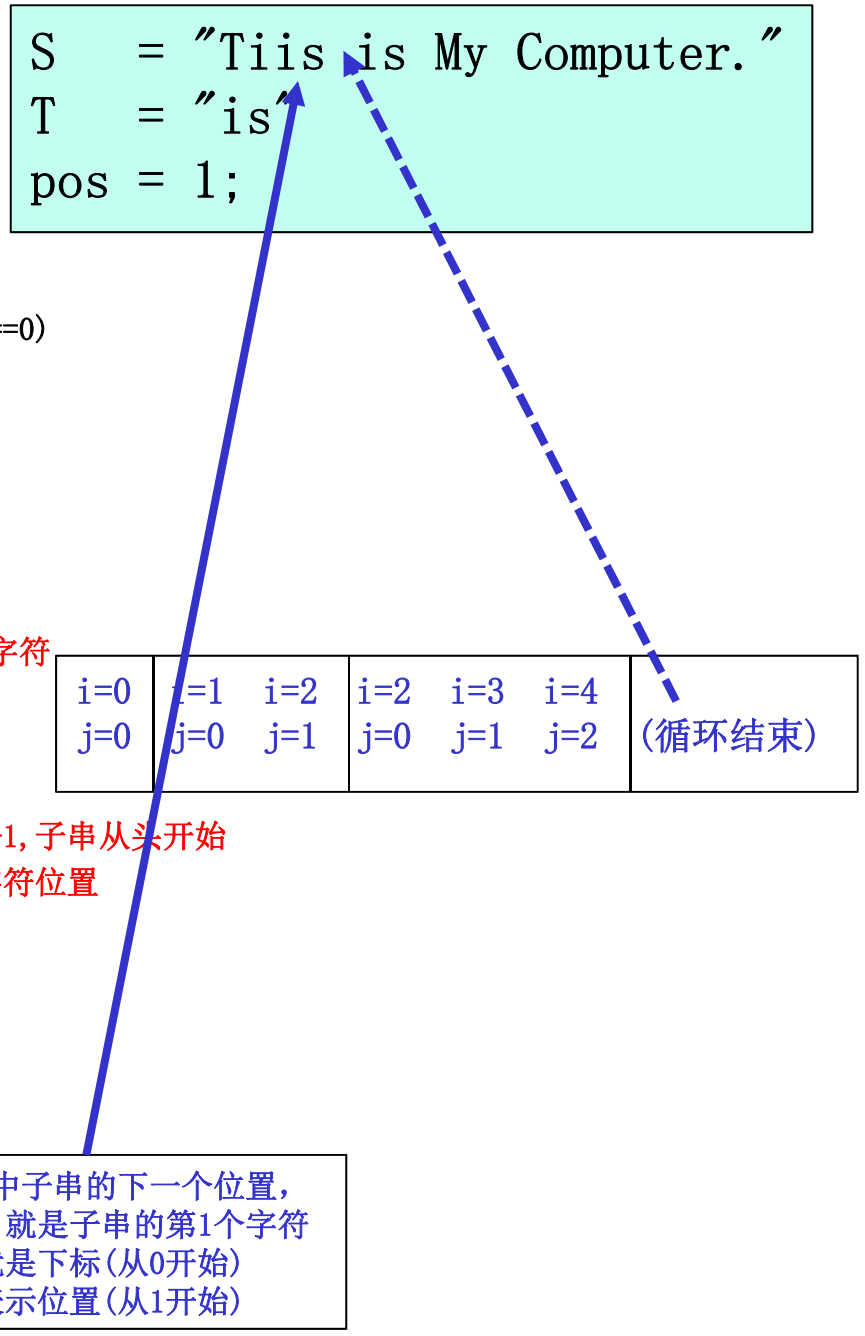
```
/* hstring.c 的组成 */

/* 清空串 */
Status ClearString(HString *S)
{
    /* 如果已申请了空间则释放 */
    if (S->ch) {
        free(S->ch);
        S->ch = NULL; //必须要，否则是非NULL
                      //再做StrAssign等操作时会再次释放
        }
    S->length = 0; //长度置0
    return OK;
}
```

```
/* hstring.c 的组成 */
/* 串连接 */
Status Concat(HString *T, HString S1, HString S2)
{
    int i;
    /* 释放T原有空间 */
    if (T->ch)
        free(T->ch);

    /* 按新长度为T申请新空间 */
    T->length = S1.length + S2.length;
    if (!(T->ch = (char *)malloc(T->length*sizeof(char))))
        exit(LOVERFLOW);

    /* 先复制S1到T */
    for (i=0; i<S1.length; i++)
        T->ch[i] = S1.ch[i];
    /* 再把S2接在T后面 */
    for (i=0; i<S2.length; i++)
        T->ch[S1.length+i] = S2.ch[i];
    return OK;
}
```

```
/* hstring.c 的组成 */
/* 取子串 */
Status SubString(HString *Sub, HString S, int pos, int len)
{   int i;

    /* 判断pos和len的值是否合法，其中len是否合法取决于pos */
    if (pos<1 || pos > S.length || len < 0 || len > S.length-pos+1)
        return ERROR;

    /* 释放原有子串 */
    if (Sub->ch)
        free(Sub->ch);

    if (!len) {    /* len=0是允许的，表示取的子串长度为0 */
        Sub->ch = NULL;
        Sub->length = 0;
        }
    else { /* 取的子串长度不为0 */
         /* 申请新空间 */
        if (!(Sub->ch = (char *)malloc(len * sizeof(char))))
            exit(LOVERFLOW);
        /* 复制子串 */
        for (i=0; i<len; i++)
            Sub->ch[i] = S.ch[pos-1+i];
        /* 置子串长度 */
        Sub->length = len;
        }
    return OK;
}
```

因为从[0]开始存放，pos-1为第pos个字符，即起始位置

```
/* hstring.c 的组成 */
/* 在主串中查找子串在指定位置后第1次出现的位置 */
int Index(HString S, HString T, int pos)
{
    int i, j;

    /* 位置非法或T、S为空串则直接返回0 */
    if (pos < 1 || pos > S.length || T.length==0 || S.length==0)
        return 0;

    /* 从主串的第pos个位置开始，主串/子串都有内容则循环 */
    i = pos-1;  //主串从第pos个字符开始，就是下标[pos-1]
    j = 0;       //子串从第1个字符开始，就是下标[0]
    while(i<S.length && j<T.length) {
        if (S.ch[i]==T.ch[j]) { //对应字符相等则继续比较下一字符
            i++;
            j++;
            }
        else {          //对应字符不等则主串回到上次比较位置+1,子串从头开始
            i = i-j+1;  //i = i-(j-1)，j指向子串的第1个不等字符位置
            j = 0;
            }
        }

    if (j>=T.length) //表示已超过子串的长度，即找到了
        return (i-T.length+1);
    else
        return 0; //未找到
}
```



```
/* hstring.c 的组成 */
/* 子串替换 */
Status Replace(HString *S, HString T, HString V)
{   int i, j, k;
    /* 主串或子串为空则直接返回(替换串V允许为空串，表示删除) */
    if (S->length==0 || T.length==0)
        return ERROR;
    i=0;
    while(1) {
       在主串的[i]处开始查找子串是否存在;
       if (存在) {
           替换 (长度T==V相等：直接替换)
                (长度T>V：替换，后续字符再向前移动若干)
                (长度T<V：先扩展空间，后续字符向后移动，再替换)
           i指向主串中查找到子串的后续位置，再次循环查找
           }
       else
           return OK;
       }
}
```

算法描述

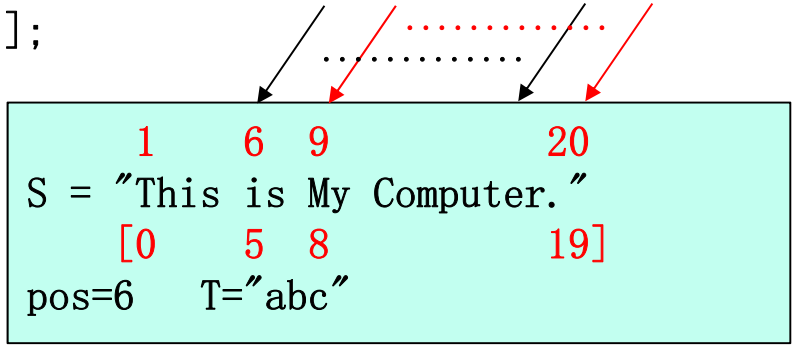
```
/* hstring.c 的组成 */
/* 子串替换 */
Status Replace(HString *S, HString T, HString V)
{   int i, j, k;
    /* 主/子串为空则直接返回(V允许为空串，表示删除) */
    if (S->length==0 || T.length==0)
        return ERROR;

    i = 0; //主串位置，循环外赋值，从头开始找子串
    while(1) {
        j = 0; //子串位置从头开始
        while(i<S->length && j<T.length) {
            if (S->ch[i]==T.ch[j]) { //对应字符相等
                i++;
                j++;
                }
            else { //对应字符不等,主串回上次位置+1
                i = i-(j-1); //j指向子串第1个不等字符
                j = 0;
                }
            } //end of while                    与Index类似

        if (j>=T.length) {//j超过子串长度，表示找到
            ...
            }
        else //没找到，直接返回即可
            return OK;
        } // end of while(1)
}
```

分三种情况讨论

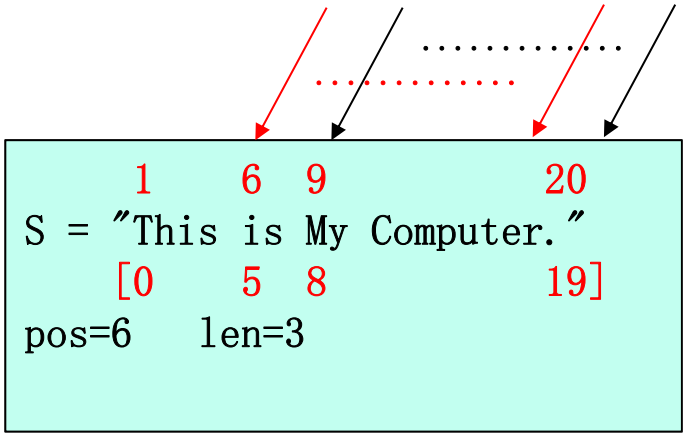
```
i -= T.length; //i指向主串中子串的最后，要回到首部
if (V.length == T.length) {//替换串长度==被替换串长度
    /* 直接替换 */
    for (k=0; k<V.length; k++)
        S->ch[k+i] = V.ch[k];
    }
else if (V.length<T.length) {//替换串长度<被替换串长度
    /* 先替换 */
    for (k=0; k<V.length; k++)
        S->ch[k+i] = V.ch[k];
    /* 主串中被替换后剩余的部分前移 */
    for(k=i+V.length; k<S->length; k++)
        S->ch[k] = S->ch[k + T.length - V.length];
    /* 主串长度减少并重新申请空间 */
    S->length -= T.length - V.length;
    if (!(S->ch = (char *)realloc(S->ch,
                                  S->length*sizeof(char))))
        exit(LOVERFLOW);
    }
else {  //替换串长度>被替换串长度
    /* 主串长度增加并重新申请空间 */
    S->length += V.length - T.length;  //V-T是正数
    if (!(S->ch = (char*)realloc(S->ch,
                                 S->length*sizeof(char))))
        exit(LOVERFLOW);
    /* 被替换子串后面的字符先向后移动
       (注意：S->length是已增加过的新值) */
    for (k=S->length-1; k>i+T.length; k--)
        S->ch[k] = S->ch[k-(V.length-T.length)];
    /* 替换 */
    for (k=0; k<V.length; k++)
        S->ch[k+i] = V.ch[k];
    }
i += V.length; //i移动到本次替换后的第1个字符
               //做为下次的查找位置
```

此句作用?

```
/* hstring.c 的组成 */
/* 在指定位置插入子串 */
Status StrInsert(HString *S, int pos, HString T)
{   int i;
    if (pos<1 || pos>S->length+1)
        return ERROR; //位置不对直接返回
    if (T.length) { //要插入的串不为空
        /* 重新申请空间 */
        if (!(S->ch = (char *)realloc(S->ch, (S->length+T.length)*sizeof(char))))
            exit(LOVERFLOW);
        /* 从pos - S->length之间的字符均向后移动T.length个位置 */
        for (i=S->length-1; i>=pos-1; i--)
            S->ch[i+T.length] = S->ch[i];
        /* 在pos位置处插入 */
        for (i=0; i<T.length; i++)
            S->ch[pos-1+i] = T.ch[i];
        /* 主串长度增加 */
        S->length += T.length;
        }
    //如果T.length==0，相当于插入空串，此时什么也不做，但返回OK
    return OK;
}
```



```
/* hstring.c 的组成 */
Status StrDelete(HString *S, int pos, int len)
{   int i;
    /* 判断pos和len的值是否合法，其中len是否合法取决于pos */
    if (pos<1 || pos>S->length || len<1 || len>S->length-pos+1)
        return ERROR;

    /* 从pos+len - S->length 之间的字符每个向前移动len个位置 */
    for (i=pos-1+len; i<S->length; i++)
        S->ch[i-len] = S->ch[i];
    /* 长度缩小 */
    S->length -= len;
    if (S->length>0) { //删除后长度不为0则缩小空间
        if (!(S->ch = (char *)realloc(S->ch, S->length*sizeof(char))))
            exit(LOVERFLOW);
        }
    else { //若删除后长度为0，则置空
        free(S->ch);
        S->ch = NULL;
        }
    return OK;
}
```



```
/* hstring.c 的组成 */

/* 销毁串 */
Status DestroyString(HString *S)
{
    /* 释放已申请的空间 */
    if (S->ch) {
        free(S->ch);
        S->ch = NULL;
        }
    S->length = 0;
    return OK;
}
```

和ClearString一样

§4. 串

4.2.2. 堆分配存储表示

★ C语言方式的定义与实现

- 多个函数中用到了realloc，该操作包含了大量的数据移动，效率较低
- 上学期TString类中曾经做过一个循环+直到100M的测试，效率很低，就是因为不断进行类似realloc的操作

★ C++语言方式的定义与实现

```
StrInit        => 构造函数
StrVisit       => cout << 重载
StrAssign      => =重载(HString = char *)
StrCopy        => =重载(HString = HString)
StrEmpty       =>
StrCompare     => 重载6个比较运算符(含义扩展)
StrLength      =>
ClearString    => =重载 (HString = NULL)
Concat         => +和=重载，T = S1+S2
SubString      =>
Index          =>
Replace        =>
StrInsert      =>
StrDelete      =>
DestroyStrint  => 析构函数
```

结合上学期10-b2作业的TString类自己完成

§4. 串

4.2. 串的表示和实现

4.2.3. 串的块链存储表示

P.78 定义

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

★ 未采用一个字符一个结点的方式，
因为空间浪费太大

```
typedef struct {
   char ch;
   struct truck *next;
   }
```

ch ：8字节(7字节填充)
next：8字节指针

★ 本质仍是单链表
★ 大部分基本操作(复制、较等)都可能跨结点

4.3. 串的模式匹配算法(讲课作业)

4.4. 串操作应用举例(略)