数据结构

第四章

4.2

`StrAssign` , `StrLength` , `SubString` , `Concat` , `StrCompare` 构成了基本操作

4.3

`StrLength(s) = 14` , `StrLength(s) = 4` ,

`SubString(s, 8, 7) = ERROR` , `SubString(t, 2, 1) = 'O'` ,

`Index(s, 'A') = 2` , `Index(s, t) = 0` ,

`Replace(s, 'STUDENT', q) = 'I AM A WORKER'` ,

`Concat(SubString(s, 6, 2), Concat(t, SubString(s, 7, 8))) =ERROR`

4.8

j	1	2	3	4	5	6	7	8	9	10
next[j]	0	1	1	2	1	1	2	3	4	3

Step1:

ADBADABBAABADABBADADA i=3

ADABBADADA j=3 , next[j] = 1

Step2:

ADBADABBAABADABBADADA i=3

ADABBADADA j=1 , next[j] = 0

Step3:

ADB**ADABBA**ABADABBADADA i=10

ADABBADADA j=7 , next[j] = 2

Step4:

ADBADABB**(A)**ABADABBADADA i=10

(A)DABBADADA j=2 , next[j] = 1

Step5:

ADBADABBA**A**BADABBADADA i=11

ADABBADADA j=1 , next[j] = 0

Step6:

ADBADABBAABADABBADADA　　i=11

　　ADABBADADA　　j=1 , next[j] = 0

Step7:

ADBADABBAAB**ADABBADADA**　　OK

　　ADABBADADA　　OK return 12

4.10

```
StringType StrReverse(StringType* S){
    int len = StrLength(*S);
    StringType temp;
    for (int i=0; i<len; i++){
        temp = Concat(temp, SubString(*S, len-i-1, 1));
    }
    return temp
}
```

4.21

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct node{
    char c;
    struct node* next;
}node;

typedef struct str{
    node* head;
    int len;
}str;

void StrAssign(str* S, char* chars);
void StrCopy(str* S, str H);
int StrCompare(str S, str H);
int StrLength(str S);
void Concat(str* S, str H, str T);
void SubString(str* S, str H, int start, int len);
//为了方便以后用，多实现了几个
str StrCreate();
void StrInit(str* S);
node* StrPos(str S, int pos);
void StrPrint(str S);

//返回一个结构体
str StrCreate(){
    str S;
    S.head = (node*)malloc(sizeof(node));
    S.head->next = NULL;
    S.len = 0;
    return S;
}

//字符串初始化，释放掉所有空间
void StrInit(str* S){
    node* p = S->head->next;
    node* temp;
    while (p){
        temp = p->next;
        free(p);
        p = temp;
    }

    S->len = 0;
    S->head->next = NULL;
}


//返回S中pos位置的节点，允许返回头节点
node* StrPos(str S, int pos){
    node* ps = S.head;

    for (int i=0; i<pos; i++){
```

```
        ps = ps->next;
    }
    return ps;
}

//输出字符串
void StrPrint(str S){
    node* p = S.head->next;
    while(p){
        printf("%c", p->c);
        p = p->next;
    }
    printf("\n");
}

void StrAssign(str* S, char* chars){
    S->len = 0;
    S->head = (node*)malloc(sizeof(node));
    S->head->next = NULL;
    node* p = S->head;

    while(*chars){
        node* temp = (node*)malloc(sizeof(node));
        temp->c = *(chars++);
        temp->next = NULL;
        p->next = temp;
        p = p->next;
        S->len ++;
    }
}

void StrCopy(str* S, str H){
    //若S是一个未初始化的或者是空串，进行初始化
    StrInit(S);
    node* ps = S->head;
    node* ph = H.head->next;
    S->len = H.len;
    //复制
    while(ph){
        node* temp = (node*)malloc(sizeof(node));
        temp->c = ph->c;
        temp->next = NULL;
        ps->next = temp;
        ps = ps->next;
        ph = ph->next;
    }
}

int StrLength(str S){
    return S.len;
}


int StrCompare(str S, str H){
```

```
    node* ps = S.head->next;
    node* ph = H.head->next;

    while(ps && ph){
        if (ps->c != ph->c){
            return ps->c - ph->c;
        }
        else{
            ps = ps->next;
            ph = ph->next;
        }
    }
    return S.len - H.len;
}

void Concat(str* S, str H, str T){
    //先让S=H
    StrCopy(S, H);
    //再生成一个空的temp
    str temp = StrCreate();
    //然后temp=T
    StrCopy(&temp, T);
    //找到S的尾节点
    node* p = StrPos(*S, S->len);
    //S尾节点接上temp头节点的next
    p->next = temp.head->next;
    //释放temp头节点
    free(temp.head);
    S->len = H.len + T.len;
}

//把H从start开始长为len的子串用S返回
void SubString(str* S, str H, int start, int len){
    if (start<1 || start>=H.len || len<0 || len>H.len-start+1){
        printf("Fail to get substring, out of bounds\n");
        exit(-1);
    }
    //初始化S
    StrInit(S);
    S->len = len;
    node* ps = S->head;
    node* ph = StrPos(H, start);

    for (int i=0; i<len; i++){
        node* temp = (node*)malloc(sizeof(node));
        temp->c = ph->c;
        temp->next = NULL;
        ps->next = temp;
        ps = ps->next;
        ph = ph->next;
    }
}
```

4.29

```c
#include "link_str.h"                    //也就是上一题的代码

node* StrPos(str S, int pos);
int KMP(str S, str H, int pos);
int KMP_next(str H, int* next);

void main(){
    str S = StrCreate();
    str H = StrCreate();
    StrAssign(&S, "ASBADABBAABADABBADADA");
    StrAssign(&H, "ADABBADADA");
    printf("%d", KMP(S, H, 1));
    return;
}

int KMP(str S, str H, int pos){
    //边界判断
    if (pos<1 || pos>StrLength(S)){
        printf("Fail to get substring, out of bounds\n");
        exit(-1);
    }
    //获取主串位置pos
    int i = pos;
    int j = 1;
    node* ps = StrPos(S, i);
    node* ph = StrPos(H, j);
    //计算next
    int next[H.len + 1];
    KMP_next(H, next);
    //开始匹配
    while(i<=S.len && j<=H.len){
        if (j==0 || ps->c == ph->c){
            i++;
            j++;
            ps = ps->next;
            ph = ph->next;
        }
        else{
            j = next[j];
            ph = StrPos(H, j);
        }
    }
    if (j>H.len){
        return i - H.len;
    }
    return 0;
}

int KMP_next(str H, int* next){
    next[0] = -1;
    next[1] = 0;


    int i = 1;
```

```
    int j = 0;
    node* ph = StrPos(H, i);
    node* _ph = StrPos(H, j);
    while (i < H.len){
        if (j == 0 || ph->c == _ph->c){
            i++;
            j++;
            ph = ph->next;
            _ph = _ph->next;
            if (ph->c != _ph->c){
                next[i] = j;
            }
            else{
                next[i] = next[j];
            }
        }
        else{
            j = next[j];
            _ph = StrPos(H, j);
        }
    }
}
```

4.30

```c
#include "link_str.h"

int StrMaxRep(str S, str* H);

void main(){
    str S = StrCreate();
    str H = StrCreate();

    StrAssign(&S, "aaaabbbdccdddd3ddde");
    printf("The string is:\n");
    StrPrint(S);

    printf("The max repeat %d times, it is:\n", StrMaxRep(S, &H));
    StrPrint(H);
    return;
}

//求S最大重复子串，用H返回该串，用返回值返回位置
//代码是在链表结构的string下写成并调试的，然后修改到顺序存储
//四个斜杠注释的表示修改的地方
int StrMaxRep(str S, str* H){
    int len = StrLength(S);
    int num_now = 0;
    int num_max = 0;

    int pos_now = 1;
    int pos_max = 0;

    ////node* p = StrPos(S, pos_now);
    ////char char_now = p->c;
    char char_now = S[1];
    char char_max;

    for (int i=0; i<len; i++){
        ////if (p->c == char_now){
        if (S[i+1] == char_now){
            num_now++;
        }
        else{
            if (num_now > num_max){
                num_max = num_now;
                char_max =char_now;
                pos_max = pos_now;
            }
            ////char_now = p->c;
            char_now = S[i+1];
            num_now = 1;
            pos_now = i;
        }
        ////p = p->next;
    }
    if (num_now > num_max){

        num_max = num_now;
```

```
            pos_max = pos_now;
            char_max =char_now;
        }
        //生成一个字符串，也就是最大重复子串
        char temp[num_max + 1];
        for (int i=0; i<num_max; i++){
            temp[i] = char_max;
        }
        temp[num_max] = '\0';
        //给H赋值为刚才生成的字符串
        StrAssign(H, temp);
        //返回重复次数
        return pos_max;
    }
```

算法时间复杂度为 $o(n)$.