串

1.顺序串

```
#include <stdio.h>
#define MaxSize 100

typedef struct
{
    char data[MaxSize];
    int length;         //串长
} SqString;

void StrAssign(SqString &s,char cstr[]) //字符串常量赋给串s
{
    int i;
    for (i=0;cstr[i]!='\0';i++)
        s.data[i]=cstr[i];
    s.length=i;
}

void DestroyStr(SqString &s)
{  }

void StrCopy(SqString &s,SqString t)    //串复制
{
    int i;
    for (i=0;i<t.length;i++)
        s.data[i]=t.data[i];
    s.length=t.length;
}

bool StrEqual(SqString s,SqString t)    //判串相等
{
    bool same=true;
    int i;
    if (s.length!=t.length)             //长度不相等时返回0
        same=false;
    else
        for (i=0;i<s.length;i++)
            if (s.data[i]!=t.data[i])   //有一个对应字符不相同时返回0
            {   same=false;
                break;
            }
    return same;
}

int StrLength(SqString s)   //求串长
{
    return s.length;
}
```

```
SqString Concat(SqString s,SqString t)  //串连接
{
    SqString str;
    int i;
    str.length=s.length+t.length;
    for (i=0;i<s.length;i++)    //将s.data[0..s.length-1]复制到str
        str.data[i]=s.data[i];
    for (i=0;i<t.length;i++)    //将t.data[0..t.length-1]复制到str
        str.data[s.length+i]=t.data[i];
    return str;
}

SqString SubStr(SqString s,int i,int j) //求子串
{
    SqString str;
    int k;
    str.length=0;
    if (i<=0 || i>s.length || j<0 || i+j-1>s.length)
        return str;                 //参数不正确时返回空串
    for (k=i-1;k<i+j-1;k++)         //将s.data[i..i+j]复制到str
        str.data[k-i+1]=s.data[k];
    str.length=j;
    return str;
}

SqString InsStr(SqString s1,int i,SqString s2)  //插入串
{
    int j;
    SqString str;
    str.length=0;
    if (i<=0 || i>s1.length+1)  //参数不正确时返回空串
        return str;
    for (j=0;j<i-1;j++)             //将s1.data[0..i-2]复制到str
        str.data[j]=s1.data[j];
    for (j=0;j<s2.length;j++)       //将s2.data[0..s2.length-1]复制到str
        str.data[i+j-1]=s2.data[j];
    for (j=i-1;j<s1.length;j++)     //将s1.data[i-1..s1.length-1]复制到str
        str.data[s2.length+j]=s1.data[j];
    str.length=s1.length+s2.length;
    return str;
}

SqString DelStr(SqString s,int i,int j)     //串删去
{
    int k;
    SqString str;
    str.length=0;
    if (i<=0 || i>s.length || i+j>s.length+1) //参数不正确时返回空串
        return str;
    for (k=0;k<i-1;k++)             //将s.data[0..i-2]复制到str
        str.data[k]=s.data[k];
    for (k=i+j-1;k<s.length;k++)    //将s.data[i+j-1..s.length-1]复制到str
        str.data[k-j]=s.data[k];
    str.length=s.length-j;
    return str;
```

```
}

SqString RepStr(SqString s,int i,int j,SqString t)  //子串替换
{
    int k;
    SqString str;
    str.length=0;
    if (i<=0 || i>s.length || i+j-1>s.length) //参数不正确时返回空串
        return str;
    for (k=0;k<i-1;k++)             //将s.data[0..i-2]复制到str
        str.data[k]=s.data[k];
    for (k=0;k<t.length;k++)        //将t.data[0..t.length-1]复制到str
        str.data[i+k-1]=t.data[k];
    for (k=i+j-1;k<s.length;k++)    //将s.data[i+j-1..s.length-1]复制到str
        str.data[t.length+k-j]=s.data[k];
    str.length=s.length-j+t.length;
    return str;
}

void DispStr(SqString s)    //输出串s
{
    int i;
    if (s.length>0)
    {   for (i=0;i<s.length;i++)
            printf("%c",s.data[i]);
        printf("\n");
    }
}
```

2.链串

```
#include <stdio.h>
#include <malloc.h>

typedef struct snode
{
    char data;
    struct snode *next;
} LinkStrNode;

void StrAssign(LinkStrNode *&s,char cstr[]) //字符串常量cstr赋给串s
{
    int i;
    LinkStrNode *r,*p;
    s=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    r=s;                            //r始终指向尾结点
    for (i=0;cstr[i]!='\0';i++)
    {   p=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        p->data=cstr[i];
        r->next=p;r=p;
    }
    r->next=NULL;
}

void DestroyStr(LinkStrNode *&s)
```

```
{   LinkStrNode *pre=s,*p=s->next;  //pre指向结点p的前驱结点
    while (p!=NULL)                 //扫描链串s
    {   free(pre);                  //释放pre结点
        pre=p;                      //pre、p同步后移一个结点
        p=pre->next;
    }
    free(pre);                      //循环结束时,p为NULL,pre指向尾结点,释放它
}

void StrCopy(LinkStrNode *&s,LinkStrNode *t)    //串t复制给串s
{
    LinkStrNode *p=t->next,*q,*r;
    s=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    r=s;                            //r始终指向尾结点
    while (p!=NULL)                 //将t的所有结点复制到s
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
}

bool StrEqual(LinkStrNode *s,LinkStrNode *t)    //判串相等
{
    LinkStrNode *p=s->next,*q=t->next;
    while (p!=NULL && q!=NULL && p->data==q->data)
    {   p=p->next;
        q=q->next;
    }
    if (p==NULL && q==NULL)
        return true;
    else
        return false;
}

int StrLength(LinkStrNode *s)   //求串长
{
    int i=0;
    LinkStrNode *p=s->next;
    while (p!=NULL)
    {   i++;
        p=p->next;
    }
    return i;
}

LinkStrNode *Concat(LinkStrNode *s,LinkStrNode *t)  //串连接
{
    LinkStrNode *str,*p=s->next,*q,*r;
    str=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    r=str;
    while (p!=NULL)                 //将s的所有结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
```

```
        r->next=q;r=q;
        p=p->next;
    }
    p=t->next;
    while (p!=NULL)             //将t的所有结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
    return str;
}

LinkStrNode *SubStr(LinkStrNode *s,int i,int j) //求子串
{
    int k;
    LinkStrNode *str,*p=s->next,*q,*r;
    str=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    str->next=NULL;
    r=str;                      //r指向新建链表的尾结点
    if (i<=0 || i>StrLength(s) || j<0 || i+j-1>StrLength(s))
        return str;             //参数不正确时返回空串
    for (k=0;k<i-1;k++)
        p=p->next;
    for (k=1;k<=j;k++)          //将s的第i个结点开始的j个结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
    return str;
}

LinkStrNode *InsStr(LinkStrNode *s,int i,LinkStrNode *t)        //串插入
{
    int k;
    LinkStrNode *str,*p=s->next,*p1=t->next,*q,*r;
    str=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    str->next=NULL;
    r=str;                                  //r指向新建链表的尾结点
    if (i<=0 || i>StrLength(s)+1)           //参数不正确时返回空串
        return str;
    for (k=1;k<i;k++)                       //将s的前i个结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    while (p1!=NULL)                        //将t的所有结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p1->data;
        r->next=q;r=q;
        p1=p1->next;
```

```
    }
    while (p!=NULL)                         //将结点p及其后的结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
    return str;
}

LinkStrNode *DelStr(LinkStrNode *s,int i,int j) //串删去
{
    int k;
    LinkStrNode *str,*p=s->next,*q,*r;
    str=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    str->next=NULL;
    r=str;                          //r指向新建链表的尾结点
    if (i<=0 || i>StrLength(s) || j<0 || i+j-1>StrLength(s))
        return str;                 //参数不正确时返回空串
    for (k=0;k<i-1;k++)             //将s的前i-1个结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    for (k=0;k<j;k++)                   //让p沿next跳j个结点
        p=p->next;
    while (p!=NULL)                     //将结点p及其后的结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
    return str;
}

LinkStrNode *RepStr(LinkStrNode *s,int i,int j,LinkStrNode *t)  //串替换
{
    int k;
    LinkStrNode *str,*p=s->next,*p1=t->next,*q,*r;
    str=(LinkStrNode *)malloc(sizeof(LinkStrNode));
    str->next=NULL;
    r=str;                              //r指向新建链表的尾结点
    if (i<=0 || i>StrLength(s) || j<0 || i+j-1>StrLength(s))
        return str;                     //参数不正确时返回空串
    for (k=0;k<i-1;k++)                 //将s的前i-1个结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;q->next=NULL;
        r->next=q;r=q;
        p=p->next;
    }
    for (k=0;k<j;k++)                   //让p沿next跳j个结点
        p=p->next;
```

```
    while (p1!=NULL)                //将t的所有结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p1->data;q->next=NULL;
        r->next=q;r=q;
        p1=p1->next;
    }
    while (p!=NULL)                 //将结点p及其后的结点复制到str
    {   q=(LinkStrNode *)malloc(sizeof(LinkStrNode));
        q->data=p->data;q->next=NULL;
        r->next=q;r=q;
        p=p->next;
    }
    r->next=NULL;
    return str;
}

void DispStr(LinkStrNode *s)    //输出串
{
    LinkStrNode *p=s->next;
    while (p!=NULL)
    {   printf("%c",p->data);
        p=p->next;
    }
    printf("\n");
}
```

串的模式匹配

1. BF（Brute-Force）算法/朴素模式匹配算法

```
int index(SqString s,SqString t)
{
    int i=0,j=0;
    while (i<s.length && j<t.length)
    {
        if (s.data[i]==t.data[j])   //继续匹配下一个字符
        {
            i++;                    //主串和子串依次匹配下一个字符
            j++;
        }
        else                        //主串、子串指针回溯重新开始下一次匹配
        {
            i=i-j+1;                //主串从下一个位置开始匹配
            j=0;                    //子串从头开始匹配
        }
    }
    if (j>=t.length)
        return(i-t.length);         //返回匹配的第一个字符的下标
    else
        return(-1);                 //模式匹配不成功
}
```

2. KMP算法

```
// 求next数组
void GetNext(SqString t,int next[])
{
    int j,k;
    j=0;k=-1;next[0]=-1;
    while (j<t.length-1)
    {
        if (k==-1 || t.data[j]==t.data[k])  //k为-1或比较的字符相等时
        {
            j++;k++;
            next[j]=k;
            //printf("(1) j=%d,k=%d,next[%d]=%d\n",j,k,j,k);
        }
        else
        {
            k=next[k];
            //printf("(2) k=%d\n",k);
        }
    }
}

//KMP算法
int KMPIndex(SqString s,SqString t)
{
    int next[MaxSize],i=0,j=0;
    GetNext(t,next);
    while (i<s.length && j<t.length)
    {
        if (j==-1 || s.data[i]==t.data[j])
        {
            i++;j++;            //i,j各增1
        }
        else j=next[j];         //i不变,j后退
    }
    if (j>=t.length)
        return(i-t.length);     //返回匹配模式串的首字符下标
    else
        return(-1);             //返回不匹配标志
}
```

3. 王道版KMP算法

```
// 求next数组
void GetNext(SqString t,int next[])
{
    int j=0,k=0;
    next[1]=0;
    while (j<t.length)
    {
        if (k==0 || t.data[j]==t.data[k])   //k为0或比较的字符相等时
        {
            j++;k++;
            next[j]=k;
            //printf("(1) j=%d,k=%d,next[%d]=%d\n",j,k,j,k);
        }
```

```
        else
        {
            k=next[k];
            //printf("(2) k=%d\n",k);
        }
    }
}

// KMP算法
int KMPIndex(SqString s,SqString t, int next[])
{
    int i=1,j=1;
    while (i<=s.length && j<=t.length)
    {
        if (j==0 || s.data[i]==t.data[j])
        {
            i++;j++;
        }
        else j=next[j];
    }
    if (j>t.length)
        return(i-t.length);
    else
        return 0;
}
```

4. 改进的KMP算法

```
// 求nextval数组
void GetNextval(SqString t,int nextval[])  //由模式串t求出nextval值
{
    int j=0,k=-1;
    nextval[0]=-1;
    while (j<t.length)
    {
        if (k==-1 || t.data[j]==t.data[k])
        {
            j++;k++;
            if (t.data[j]!=t.data[k])
                nextval[j]=k;
            else
                nextval[j]=nextval[k];
        }
        else  k=nextval[k];
    }

}

//修正的KMP算法
int KMPIndex1(SqString s,SqString t)
{
    int nextval[MaxSize],i=0,j=0;
    GetNextval(t,nextval);
    while (i<s.length && j<t.length)
    {
```

```
        if (j==-1 || s.data[i]==t.data[j])
        {
            i++;j++;
        }
        else j=nextval[j];
    }
    if (j>=t.length)
        return(i-t.length);
    else
        return(-1);
}
```