**第四章 串**

**一、教学基本要求**

1．掌握串的有关概念。

2．掌握串的基本运算及实现。

3．掌握串的存储结构。

4． 熟悉串的静态存储结构和动态存储结构以及在这两种存储结构上实现串的各种运算的方法。

5．了解串的各种基本运算的意义。

本章目的是介绍串的逻辑结构、存储结构及其中上的基本运算，由于高级语言均已具备了较强的串处理功能。

重点：掌握串上实现的模式匹配算法，这也是本章的难点。

**二、学时安排 总学时：4**

|  |  |
| --- | --- |
| 教 学 内 容 | 学 时 |
| 1．串及其操作 串的存储结构 | 2 |
| 2. 串的模式匹配算法 | 2 |

**三、教学内容**

一、串类型的定义

1.串(string)：是由零个或多个字符组成的有限序列。记为：S=’a1 a2…an’（n>=0）

串值：用单引号括起来的字符序列。

长度：串中字符的数目。

空串：长度为零。

子串：子序列。

位置：字符在序列中的序号。

相等：当且仅当两个串的值相等。

空格串：由一个或多个空格组成的串。

2.串的操作：以串的整体为操作对象。

串赋值strassign()

串比较strcompar()

求串长strlength()

串连接concat()

求子串substring()

二、串的表示和实现

1.定长顺序存储表示

⑴ 表示：用一组地址连续的存储单元存储串值的字符序列。可用定长数组描述。

#define MAXSTRLEN 255

typedef unsigned char Sstring[MAXSTRLEN +1];

其中0号单元存放串长

⑵ 串操作的实现：

实现串操作的原操作为“字符序列的复制”。操作的时间复杂度基于复制字符序列的长度。

例1：串连接：复制、截尾。

【算法4.1】

Status concat(sstirng &T,sstring s1,sstring s2)

If(两个串的长度小于T串的长度)

{将两个串的数据拷到T串中；}

elseif(第一个串的长度小于T串的长度)

{将第一个串的内容拷到T前部分，T后半部分拷第二个串的前半部分；}

else

{将第一个串的前半部分拷到T串的前半部分；}}

例2：求子串：复制。

已知一个字符串S，求另一个字符串sub，其中sub是串s的第pos个字符长度为len的子串。

【算法4.2】

Status substring(sstring &sub,sstring s,int pos,intlen){

//判断len 的值和pos的值是不是合法的

if(pos<1||pos>s[0]||len<0||len>s[0]-pos+1)

return error;

sub[1..len]=s[pos..pos+len-1];

sub[0]=len;

return ok;

}

2.堆分配存储表示

⑴ 表示：以一组地址连续的存储单元存放串值字符序列，但它们的存储空间是在程序执行的过程中动态分配而得。在C语言中有一个称为堆的自由空间，用函数malloc()和函数free()来管理。

堆分配存储结构有顺序顺序存储结构的特点，处理方便，且操作中对串长没有限制。

typedef struct{

char \*ch;

int length;

}Hstring;

⑵ 串操作的实现：也是基于字符序列的复制。

例1：串插入

【算法4.3】

status strinsert(hstring &s,int pos,hsrting t){

if(pos<1||pos>s.length+1) return error;

if(t.length){//T非空串

if((s.ch=(char )realloc(s.ch,(s.length+t.length)sizeof(char))))

exit(overflow);//分配失败

for(I=s.length-1;I>=pos-1;I--)//从第pos个串开始将字符后移

s.ch[I+t.length]=s.ch[i];

s.ch[pos-1,pos+t.length-2]=t.ch[0..t.length-1];//将T串插入到S串中

s.length+=t.length;//改变串长

}

return ok;

}

基本操作

（1）生成一个其值等于串常量chars的串T

【算法4.4】

status stressing(hstring &t,char \*chars){

if(t.ch) free(t);//T已经存在

for(I=0,c=chars;c;++I,++c);//求串长

if(!i)//串长为零

{t.ch=null;

t.length=0;

}

else{//开辟空间，为新串赋值

t.ch=(\*char)malloc(I\*sizeof(char));

t.ch[0..i-1]=chars[0..i-1];

t.length=I;

}

return ok;

}

（2）串比较

【算法4.5】

int strcompare(hstring s,hstring t){

for(I=0;I<s.length&&I<t.length;I++)

if(s.ch[i]!=t.ch[i])

return s.ch[i]-t.ch[i];

return s.length-t.length;//两个串前部分相等

}

（3）连接串(用T返回由S1和S2联接而成的新串)

（4）求子串（返回串S第pos个字符起长度为len的子串）

3.串的块链存储表示

⑴ 表示：用链表方式存储串值，每个结点可以存放一个字符，也可以存放多个字符。不了便于操作，除头指针外还要附设一个尾指针，指向链表的最后一个结点，同时连要存储链表的长度，这些结构是基于对串经常进行的操作而设定的。

#DEFINE CHUNKSIZE 80

typedef struct Chunk{

char ch[CHUNKSIZE];

struct Chunk \*next;

}Chunk;//串的结点的结构

typedef struct{

Chunk \*head, \*tail;

int curlen;

}Lstring;

存储密度=串值所占的存储位/实际分配的存储位。

串的模式匹配与应用

一、 串的模式匹配算法

1. 求子串位置的定位函数 Index(S, T, pos)

模式匹配：子串的定位操作称作串的模式匹配。T：模式串

例：定长顺序存储结构的匹配算法（不依赖于串的其他操作）

思想：从主串S的第pos个字符起和模式的第一个字符比较之，若相等，则继续比较后续字符，否则从主串的下一个字符起再重新和模式的字符比较之。

【算法4.6】

Int index(sstring s,sstring t, int pos){

//返回子串T在主串S中第pos个字符之后的位置，若不存在，则函数值为0

i=pos;j=1;

While(i<=s[0]&&j<=t[0]

{

if (s[i]=t[j]) {++i;++j;}

else {i=j-j+2; j=1;}

}

if(j>t[0]) return i-t[0];

else return 0;

}

算法缺点：有时效率低，回溯指针次数多。

2.模式匹配的一种改进算法

KMP算法的改进：每当一趟匹配过程中出现字符比较不等时，不需指针回溯，而是利用得到的部分匹配的结果，将模式向右滑动尽可能远的一段距离后，继续进行比较。

二、串存在应用举例

1.文本编辑

文本编辑程序：面向用户的系统服务程序。

文本编辑的功能：原程序的输入和修改。

文本编辑的实质：修改字符序列的形式或格式。

文本编辑的基本操作：串的查找、插入、删除。

文本=一个字符串，文本串。

文本=页+换页符，文本子串。

页=行+换行符，文本子串。

页表：页号、起始行号

行表：行号、起始地址、该行的子串长度。

页指针、行指针和字符指针分别指示当前的页、行和字符。

文本的操作：就是修改页表和行表。