**江西师范大学计算机信息工程学院学生实验报告（4）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **专业：** | **数据科学与大数据技术2班** | **姓名：** | **赖丽婷** | **学号：** |  | **日期：** | **2021.10.14** |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 数据结构 | 实验室名称 | 计算机综合实验室 |
| 实验名称 | 栈与字符串 | | |
| 指导教师 |  | 成绩 |  |

**1.实验目的**（结出本次实验所涉及并要求掌握的知识点）

1. 掌握栈的结构与基本运算实现方法
2. 掌握栈表达式计算的基本技术
3. 掌握应用栈进行问题求解的基本方法
4. 理解掌握串的相关概念和运算实现
5. 掌握快速模式匹配等串的典型算法

**2.实验内容**（结出实验内容具体描述）

1. 利用顺序栈实现十进制向16进制之间的转换
2. 利用链式栈实现转换
3. 利用字符顺序栈实现表达式求值
4. 在字符串S中从第i的位置取长度为len的子串
5. 从字符串s中删除从第i个位置开始，长度为len的子串
6. 查找子串t在主串s中第一次出现的位置
7. 用朴素模式匹配查找查找子串t在主串s中出现的所有位置
8. 编写KMP算法中的next数组与快速匹配算法

**3.算法描述及实验步骤**（用适当的形式表达算法设计思想与算法实现步骤）

1. 根据进制转换算法与栈结构，得知需要吧m%16的值放入栈中，然后m=m/16执行循环至m==0，因为16进制的后6位由字母表示这时我们需要把数字转换位字符，如果栈中的数大于十根据ascii码，我们将栈值加55，如果小于十，栈中取出的值加44
2. 第三题先将中缀表达式转换为后缀表达式，然后在计算后缀表达式的函数中定义一个数组用栈来保存数字，如果遇到了运算符将栈顶与栈顶前一位进行运算，然后栈顶减一，最终返回栈顶保存的数就是最后的计算结果
3. 4-5先找到位置，如果找不到则返回NULL,然后进行读取与删除操作
4. 6-7用朴素的模式匹配，终止条件位i小于长串长度减去模式子串长度如果匹配到了保存i，然后再往后匹配,如果匹配失败如果，则又从i+1位开始重新与子串匹配
5. 8kmp算法，我们也可以用kmp算法先找到要删除的字符的位置，进行删除，提高效率

kmp算法中的核心是next数组，next[i]记录当前位前面数组的匹配对数，定义next[0]=-1;

next数组是从i=0,j=0开始找模式匹配串的，当i=1时前面的匹配对数为j，再往后增，如果前一位与后一位匹配成功，模式匹配串继续加，如果不成功，则根据当前位不成功的匹配串来定也就是变为j = next[j],所以说，在next数组不匹配的过程中也是在进行kmp算法

1. **调试过程及运行结果**（详细记录在调试过程中出现的问题及解决方法。记录实验执行的结果）

**5. 总结**（对实验结果进行分析，问题回答，实验心得体会及改进意见）

1. 1-3题要熟练掌握栈相关内容，也要区分顺序栈与链式栈的不同，知道算术表达式怎么利用后缀表达式与栈进行运算
2. 4-7其实总的来说都算是朴素的模式匹配算法，要注意在匹配位到主串减去子串位时，如果这个时候没有匹配就没有必要再去找匹配了，还要注意&&与||的运算规则，不然可能会导致内存泄漏
3. 8掌握next数组的由来，其实next数组到后面没有匹配到时也可以看成kmp算法，然后最后返回的应该是最后匹配到的i位减去子串的长度
4. kmp算法极大的提高了模式匹配的效率，没用重复干相同的事情，我们在学习的过程中也应该常常思考，避免每一题都用相同的方法，要常改进代码提高效率。
5. **附录**（程序源代码等）
6. void Dto16(int m)

{ seqstack s; /\*定义顺序栈\*/

init(&s);

printf("十进制数%u对应的十六进制数是：",m);

while (m)

{

push(&s, m%16);

m = m/16;

}

while (!empty(&s))

putchar(read(&s) < 10 ? pop(&s) + 48 : pop(&s) + 55);

printf("\n");

}

1. void linkDto16(int m)

{

linkstack top;

top = init(top);

while(m)

{

top = push(top, m%16);

m = m/16;

}

while(!empty(top))

{

putchar(read(top) < 10 ? read(top) + 48 : read(top) + 55);

top = pop(top);

}

}

1. double evalpost(char f[])

{ double obst[50]; //obst数组用来保存数字

int i=0,top=-1;

//i先用来访问数字将数保存到obst数组中，后用来往后查找运算符，

double x;

while (f[i]!='\0')

{

if (f[i]>='0' && f[i]<='9')

obst[++top]=readnumber(f,&i);

else if (f[i]==' ') i++;

else if (f[i]=='+')

{ x=obst[top--];

obst[top]=x+obst[top];

i++;

}

else if (f[i]=='-')

{ x=obst[top--];

obst[top]=obst[top]-x;

i++;

}

else if (f[i]=='\*')

{ x=obst[top--];

obst[top]=obst[top]\*x;

i++;

}

else if (f[i]=='/')

{ x=obst[top--];

obst[top]=obst[top]/x;

i++;

}

}

return obst[top];

}

1. //赖丽婷

//取出从i位开始的len长度子串,带头节点的链表

linkstring substring(linkstring s, int i, int len)

{

linkstring head, q, r, p;

head = (linkstring)malloc(sizeof(linknode));

head ->next = NULL;

r = head;

int k;

k = 1;

p = s->next;

while(p && k < i)

{

p = p->next;

k++;

}

k = 0;

while(p && k < len)

{

q = (linkstring)malloc(sizeof(linknode));

q->data = p->data;

r->next = q;

r = q;

p = p->next;

k++;

}

r -> next = NULL;

return head;

}

1. void delstring(linkstring str, int i, int len){

linkstring p, q, pre;

int k = 1;

p = str->next;

pre = str;

while(p && k < i)

{

pre = p;

p = p->next;

k++;

}

k = 0;

while(p && k < len)

{

q = p;

pre->next = p->next;

p = p->next;

k++;

free(q);

}

}

1. //返回第一次出现的结点

linkstring index(linkstring s, linkstring t){

linkstring p, q, r;

p = s->next;

while(p){

r = p;

q = t->next;

//这里要把r->data与q->data的访问放到后面，不然会造成泄漏

while(q && r && r ->data == q->data){

r = r->next;

q = q->next;

}

if(q==NULL){

return p;

}

p = p->next;

}

return NULL;

}

1. int index(char \*s, char \*t)

{

int i, j, k, m, n;

i = 0;

m = strlen(s);

n = strlen(t);

while(i <= m - n)

{

k = i;

j = 0;

while( j < n&& s[k] == t[j])

{

k++;

j++;

}

if(j == n){

return i;

}

i++;

}

return -1;

}

//关键词:查找使用，返回值存入带头节点的单链表中

linklist indexall(char \*s, char \*t)

{

linklist head, p, q;

int i, j, k, m, n;

head = (linklist) malloc (sizeof(linknode));

head->next = NULL;

p = head;

m = strlen(s);

n = strlen(t);

i = 0;

while(i <= m - n)

{

k = i;

j = 0;

while(j < n && s[k] == t[j])

{

k++;

j++;

}

if(j == n)

{

q = (linklist)malloc(sizeof(linknode));

q->data = i;

p->next = q;

p = q;

}

i++;

}

p->next = NULL;

return head;

}

1. //求next值的算法

//定义一个next数组，第0位代表没有匹配字符j第一位没有前缀值位-1

//之后个代表j中每处的匹配数

//i表示str目前所在的位置,j用来记录匹配对数，如果匹配不成功则相当于对当前字符串对此节点前的字符串进行模式匹配

void getnext(seqstring s, int next[])

{

int i = 0,j;

j = -1;

next[0] = -1;

while(i < s.length)

{

if(j==-1|| s.str[i]==s.str[j])

{

i++;

j++;

next[i] = j;

}else{

j = next[j];

}

}

}

/\*快速模式匹配算法，请将函数补充完整\*/

//kmp匹配匹配成功正常匹配，如果匹配失败根据当前不匹配位的模式配对数来匹配

int kmp(seqstring str1, seqstring str2, int next[])

{

int i, j, k;

i = 0;

j = 0;

while(i <= str1.length-str2.length && j < str2.length )

{

while(j == -1 || str1.str[i]==str2.str[j])

{

i++;

j++;

}

if(j != str2.length)

{

j = next[j];

}

}

if(j == str2.length)

{

return i - str2.length;

}else{

return -1;

}

}