编译原理
作业1
成对的括号

output.txt

ABCDEFGHIJKLMNOP
0123456789
G
P
(A,0,B)
(B,~,F)
(C,1,D)
(D,~,F)
(E,~,C)
(E,~,A)
(F,~,E)
(F,~,H)
(G,~,E)
(G,~,H)
(H,~,I)
(I,0,J)
(J,~,K)
(K,1,L)
(L,~,O)
(M,0,N)
(N,~,M)
(N,~,P)
(O,~,M)
(O,~,P

```
#include<string>
#include<iostream>
#include<fstream>
#include<stack>
#include<map>
#include<vector>
#include<set>
using namespace std;

struct edge
{
char weight;
char next;
};
```

```
// edge：NFA图中的边；
// option：输入的符号集；
// expression：输入的正规式；
// infix：增加连接符的正规式；
// suffix：转换成的后缀表达式；
// postion：状态集；
// precedence：规定运算符优先级；
// nfa：生成的NFA（使用map生成了一个char与edge向量的映射，char代表NFA中的顶点，也就是生成了一个顶点与从顶点出发边的集合的映射）；
// tfunction：转移函数。
string option,expression, suffix, infix, postion;
map<char, int> precedence;
char start='@';
map<char, vector<edge>> nfa;
vector<string> tfunction;


//设置运算符优先级
void setprecedence() {
precedence['*'] = 3;
precedence['.'] = 2;
precedence['|'] = 1;
}

//从文件中读取正规式，并且判断是否有非法输入
void input() {
ifstream in("input.txt");
if (!in) {
cout << "打开文件出错！即将退出！" << endl;
system("Pause");
exit(1);
}
in >> option >> expression;
for (int i = 0; i < expression.length(); i++) {
if (option.find(expression[i])!=option.npos|| expression[i]=='|' || expression[i]=='(' || expression[i]
== ')' || expression[i] == '*' || expression[i] == '.') {
//是正规字符
}
else {
cout << "非法输入！" << endl;
system("Pause");
exit(1);
}
}
}
```

```
//加入连接符
void toinfix() {
for (int i = 0; i < expression.length(); i++) {
char tmp = expression[i];
char next;
if (i == expression.length() - 1) {
next = '\0';
}
else {
next = expression[i + 1];
}
if (((tmp != '('&&tmp != '.'&&tmp != '|') || tmp == ')' || tmp == '*') && (next != ')'&&next != '*'&&next
!= '|'&&next != '.'&&next != '\0')) {
infix = infix + tmp + '.';
}
else {
infix = infix + tmp;
}
}
}

//中缀转后缀
void tosuffix() {
stack<char> cstack;
for (int i = 0; i < infix.length(); i++) {
char tmp = infix[i];
if (tmp == '(') cstack.push(tmp);
else if (tmp == ')') {
while (cstack.top() != '(') {
char t = cstack.top();
cstack.pop();
suffix = suffix + t;
}
cstack.pop();
}
else if(tmp == '|'|| tmp == '*' || tmp == '.')
{
while (!cstack.empty()) {
char t = cstack.top();
if (precedence[t] >= precedence[tmp]) {
suffix = suffix + t;
cstack.pop();
}

else if (precedence[t] < precedence[tmp]) {
cstack.push(tmp);
break;
```

```
}
}
if (cstack.empty())cstack.push(tmp);
}
else {
suffix = suffix + tmp;
}
}
while (!cstack.empty()) {
char t = cstack.top();
cstack.pop();
suffix = suffix + t;
}
}

//找到当前nfa的终态
char findback(char a,set<char> x) {
if (nfa[a].empty()) return a;
else {
x.insert(a);
int i = 0;
while (x.find(nfa[a][i].next) != x.end()) {
i++;
}
char t=findback(nfa[a][i].next,x);
return t;
}
}


//将后缀表达式转换成nfa
char tonfa() {
stack<char>h;
set<char>x;
for (int i = 0; i < suffix.length(); i++) {
char tmp = suffix[i];
if (tmp == '.') {
edge e;
e.weight = '~';
char t = h.top();
h.pop();
x.clear();
char b = findback(h.top(),x);
e.next = t;
nfa[b].push_back(e);
}
else if (tmp == '*') {
edge e;
```

```
e.weight = '~';
start++;
char t = h.top();
h.pop();
x.clear();
char b = findback(t,x);
e.next = t;
nfa[start].push_back(e);
nfa[b].push_back(e);
e.next = start + 1;
nfa[b].push_back(e);
nfa[start].push_back(e);
h.push(start);
postion = postion + start + char(start + 1);
start++;
}
else if (tmp == '|') {
edge e;
e.weight = '~';
start++;
char t = h.top();
h.pop();
x.clear();
char b = findback(t,x);
e.next = t;
nfa[start].push_back(e);
e.next = start+1;
nfa[b].push_back(e);
t = h.top();
h.pop();
x.clear();
b = findback(t,x);
e.next = t;
nfa[start].push_back(e);
e.next = start + 1;
nfa[b].push_back(e);
h.push(start);
postion = postion + start + char(start + 1);
start++;
}
else {
start++;
edge e;
e.weight = tmp;
e.next = start + 1;
nfa[start].push_back(e);
h.push(start);
postion = postion + start + char(start+1);
```

```
start++;
}
}
return h.top();
}

//获得转换方程
void getfunction() {
for (auto inter = nfa.begin(); inter != nfa.end(); inter++) {
for (int i = 0; i < inter->second.size(); i++) {
string tmp;
tmp =tmp+ '(' + inter->first + ',' + inter->second[i].weight + ',' + inter->second[i].next + ')';
tfunction.push_back(tmp);
}
}
}


int main() {
setprecedence();
input();
toinfix();
tosuffix();
char s=tonfa();
ofstream out("output.txt");
printf("正规式为：");
cout << expression << endl;
printf("加入连接点后：");
cout << infix << endl;
printf("转为后缀表达式：");
cout << suffix << endl;
printf("\n\n结果为：（用~表示空）\n状态集为：");
out << postion << endl;
cout << postion << endl;
printf("符号集为：");
out << option << endl;
cout << option << endl;
printf("初态集为：");
printf("%c\n", s);
out << s << endl;
printf("终态集为：");
set<char>x;
cout << findback(s,x) << endl;
out << findback(s, x) << endl;
printf("转移函数为：\n");
getfunction();
for (int i = 0; i < tfunction.size(); i++) {
```

```
cout << tfunction[i] << endl;
out << tfunction[i] << endl;
}
system("Pause");
}
```