EXPERIMENT NO 7

Date of performance :
Date of submission :

Aim: To implement a LR(0) parser.

Software Used: TURBOC3.

Theory:

1. Construc transition relation between states
   - Use algorithms Initial item setand Next item set
   - States are set of LR(0) items.
     - Shift items of the form P→α•Sβ
     - Reduce items of the form P→α•
2. Construct parsing table
   - If every state contains no conflicts use LR(0) parsing algorithm.
   - If states contain conflict.
     - Rewrite grammar or
     - Resolve conflict or
     - Use stronger parsing technique.

Constructing LR(0) parsing tables:
Items
The construction of these parsing tables is based on the notion of LR(0) items (simply called items here) which are grammar rules with a special dot added somewhere in the right-hand side. For example the rule E → E + B has the following four corresponding items:
E → • E + B
E → E • + B
E → E + • B
E → E + B •
Rules of the form A → ε have only a single item A → •. The item E → E • + B, for example, indicates that the parser has recognized a string corresponding with E on the input stream and now expects to read a '+' followed by another string corresponding with B.
Item sets
It is usually not possible to characterize the state of the parser with a single item because it may not know in advance which rule it is going to use for reduction. For example if there is also a rule E → E * B then the items E → E • + B and E → E • * B will both apply after a string corresponding with E has been read. Therefore it is convenient to characterize the state of the parser by a set of items, in this case the set { E → E • + B, E → E • * B }.
Extension of Item Set by expansion of non-terminals
An item with a dot before a nonterminal, such as E → E + • B, indicates that the parser expects to parse the nonterminal B next. To ensure the item set contains all possible rules the parser may be in the midst of parsing, it must include all items describing how B itself will be parsed. This means that if there are rules such as B → 1 and B → 0 then the item set must also include the items B → • 1 and B → • 0. In general this can be formulated as follows:
If there is an item of the form A → v • Bw in an item set and in the grammar there is a rule of the form B → w' then the item B → • w' should also be in the item set.
Closure of item sets
Thus, any set of items can be extended by recursively adding all the appropriate items until all nonterminals preceded by dots are accounted for. The minimal extension is called the closure of an item set and written as

clos(I) where I is an item set. It is these closed item sets that are taken as the states of the parser, although only the ones that are actually reachable from the begin state will be included in the tables.

Augmented grammar
Before the transitions between the different states are determined, the grammar is augmented with an extra rule
Table construction:
(0) S → E
where S is a new start symbol and E the old start symbol. The parser will use this rule for reduction exactly when it has accepted the whole input string.

For this example, the same grammar as above is augmented thus:
(0) S → E
(1) E → E * B
(2) E → E + B
(3) E → B
(4) B → 0
(5) B → 1
It is for this augmented grammar that the item sets and the transitions between them will be determined.
Finding the reachable item sets and the transitions between them[edit]
The first step of constructing the tables consists of determining the transitions between the closed item sets. These transitions will be determined as if we are considering a finite automaton that can read terminals as well as nonterminals. The begin state of this automaton is always the closure of the first item of the added rule: S → • E:
Item set 0
S → • E
+ E → • E * B
+ E → • E + B
+ E → • B
+ B → • 0
+ B → • 1
The boldfaced "+" in front of an item indicates the items that were added for the closure (not to be confused with the mathematical '+' operator which is a terminal). The original items without a "+" are called the kernel of the item set.

Starting at the begin state (S0), all of the states that can be reached from this state are now determined. The possible transitions for an item set can be found by looking at the symbols (terminals and nonterminals) found following the dots; in the case of item set 0 those symbols are the terminals '0' and '1' and the nonterminals E and B. To find the item set that each symbol x leads to, the following procedure is followed for each of the symbols:
Take the subset, S, of all items in the current item set where there is a dot in front of the symbol of interest, x.
For each item in S, move the dot to the right of x.
Close the resulting set of items.
For the terminal '0' (i.e. where x = '0') this results in:
Item set 1
B → 0 •
and for the terminal '1' (i.e. where x = '1') this results in:
Item set 2
B → 1 •
and for the nonterminal E (i.e. where x = E) this results in:
Item set 3
S → E •
E → E • * B
E → E • + B
and for the nonterminal B (i.e. where x = B) this results in:
Item set 4
E → B •

Note that the closure does not add new items in all cases - in the new sets above, for example, there are no nonterminals following the dot. This process is continued until no more new item sets are found. For the item sets 1, 2, and 4 there will be no transitions since the dot is not in front of any symbol. For item set 3, note that that the dot is in front of the terminals '*' and '+'. For '*' the transition goes to:
Item set 5
E → E * • B
+ B → • 0
+ B → • 1
and for '+' the transition goes to:
Item set 6
E → E + • B
+ B → • 0
+ B → • 1
For item set 5, the terminals '0' and '1' and the nonterminal B must be considered. For the terminals, note that the resulting closed item sets are equal to the already found item sets 1 and 2, respectively. For the nonterminal B the transition goes to:
Item set 7
E → E * B •
For item set 6, the terminal '0' and '1' and the nonterminal B must be considered. As before, the resulting item sets for the terminals are equal to the already found item sets 1 and 2. For the nonterminal B the transition goes to:
Item set 8
E → E + B •
These final item sets have no symbols beyond their dots so no more new item sets are added, so the item generating procedure is complete. The finite automaton, with item sets as its states is shown below.

The transition table for the automaton now looks as follows:

Item Set	*	+	0	1	E	B
0			1	2	3	4
1						
2						
3	5	6				
4						
5			1	2		7
6			1	2		8
7						
8						

Program:-
To write a code for LR(0) Parser for following Production:
E->E+T
T->T*F/F
F->(E)/char

```
#include<string.h>
#include<conio.h>
#include<stdio.h>
int axn[][6][2]={
{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
{{-1,-1},{100,6},{-1,-1},{-1,-1},{-1,-1},{102,102}},
{{-1,-1},{101,2},{100,7},{-1,-1},{101,2},{101,2}},
```

```
{{-1,-1},{101,4},{101,4},{-1,-1},{101,4},{101,4}},
{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
{{100,5},{101,6},{101,6},{-1,-1},{101,6},{101,6}},
{{100,5},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}},
{{100,5},{-1,-1},{-1,-1},{100,4},{-1,-1},{-1,-1}},
{{-1,-1},{100,6},{-1,-1},{-1,-1},{100,11},{-1,-1}},
{{-1,-1},{101,1},{100,7},{-1,-1},{101,1},{101,1}},
{{-1,-1},{101,3},{101,3},{-1,-1},{101,3},{101,3}},
{{-1,-1},{101,5},{101,5},{-1,-1},{101,5},{101,5}}
};
int gotot[12][3]={1,2,3,-1,-1,-1,-1,-1,-1,-1,-1,-1,8,2,3,-1,-1,-1,-1,
9,3,-1,-1,10,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
int a[10];
char b[10];
int top=-1,btop=-1,i;
void push(int k) {
if(top<9)
a[++top]=k;
}
void pushb(char k) {
if(btop<9)
b[++btop]=k;
}
char TOS() {
return a[top];
}
void pop() {
if(top>=0)
top--;
}
void popb() {
if(btop>=0)
b[btop--]='\0';
}
void display()
{
for(i=0;i<=top;i++)
printf("%d%c",a[i],b[i]);
}
void display1(char p[],int m) {
int l;
printf("\t\t");
for(l=m;p[l]!='\0';l++)
printf("%c",p[l]);
printf("\n");
}
void error() {
printf("\n\nSyntax Error");
}
void reduce(int p) {
int len,k,ad;
char src,*dest;
switch(p) {
case 1:dest="E+T";
src='E';
break;
case 2:dest="T";
src='E';
```

```
break;
case 3:dest="T*F";
src='T';
break;
case 4:dest="F";
src='T';
break;
case 5:dest="(E)";
src='F';
break;
case 6:dest="i";
src='F';
break;
default:dest="\0";
src='\0';
break;
}
for(k=0;k<strlen(dest);k++) {
pop();
popb();
}
pushb(src);
switch(src)
{
case 'E': ad=0;
break;
case 'T': ad=1;
break;
case 'F': ad=2;
break;
default: ad=-1;
break;
}
push(gotot[TOS()][ad]);
}
int main()
{
int j,st,ic;
char ip[20]="\0",an;
clrscr();
printf("Enter any String :- ");
gets(ip);
push(0);
display();
printf("\t%s\n",ip);
for(j=0;ip[j]!='\0';)
{
st=TOS();
an=ip[j];
if(an>='a'&an<='z')
ic=0;
else if(an=='+')
ic=1;
else if(an=='*')
ic=2;
```

```
else if(an=='(')
ic=3;
else if(an==')')
ic=4;
else if(an=='$')
ic=5;
else
{ error();
break; }
if(axn[st][ic][0]==100) {
pushb(an);
push(axn[st][ic][1]);
display();
j++;
display1(ip,j); }
if(axn[st][ic][0]==101) {
reduce(axn[st][ic][1]);
display();
display1(ip,j);
}
if(axn[st][ic][1]==102) {
printf("Given String is Accepted");
break; }
}
getch();
return 0;
}
```

OUTPUT:-

```
Enter any String :- a+b*c
0 a+b*c
0a5 +b*c
0F3 +b*c
0T2 +b*c
0E1 +b*c
0E1+6 b*c
0E1+6b5 *c
0E1+6F3 *c
0E1+6T9 *c
0E1+6T9*7 c
0E1+6T9*7c5
```

Conclusion:- Thus we have successfully implemented LR(0) parser.

SIGN AND REMARK

DATE

R1	R2	R3	R4	R5	Total (15 Marks)	Signature