LR(1) PARSER PROGRAM USING C/C++

Submited By:
BIJAY KUSLE
M.S. Computer Science

Submited To:
Dr. SANG C. SUH.
Assistant Professor.

Department of Computer Science and Information Systems.
Texas A&M University – Commerce,
Commerce, TX 75429. USA.

LR(1) PARSER PROGRAM

This is a parser programmed in C++. I t uses LR(1) parsing algorithm to parse a string for a grammar defined. For this project the grammar is SMALLG's Grammer and is specified as below:

<Program>	→	{<StatList>}
<StatList>	→	<Stat>; | <StatList>
<Stat>	→	<AssStat> | <CondStat> | <WhileStat>|<InputStat>|<OutputStat>
<AssStat>	→	<Var> := <Exp>
<Var>	→	<Ident>
<Exp>	→	<Term>+<Term>|<Term>-<Term>|<Term>*<Term>|<Term>
<Term>	→	<Var>|<Constant>
<Constant>	→	<Digit>|<Digit>|<Constant>
<Digit>	→	0 | 1 | 2 | 3
<Letter>	→	i | j | k | m | n | x | y |z
<Ident>	→	<Letter> | <Letter><Tail>
<Tail>	→	<Letter> | <Digit> | <Letter><Tail> | <Digit><Tail>
<CondStat>	→	if <Cond>then<StatList>else<StatList>fi | if<Cond>then<StatList>fi
<WhileStat>	→	while<Cond>do<StatList>od
<Cond>	→	(<Exp><RelOp><Exp>)
<RelOp>	→	= | != | <= | < | >= | >
<InputStat >	→	read<Var>
<OutputStat>	→	write<Var>

Sample of valid strings for this grammar is given below:

1)
```
{
x:=20; y := x1;
}
```

2)
```
{
while (2 = x2) do x:=2;
}
```

3)
```
{
if (z+1 = 2) then j23 := 3; else y := y; fi;
}
```

ABOUT THE PROGRAM:

To make the programming easy and convenient it is divided into three parts:

1) Construction of Symbol Table
2) Parser
3) Error Trapping

1. Construction of a symbol table:

All Non-terminals used in the grammar is converted into uppercase alphabet as listed below:

A → <Program>
B → <StatList>
C → <Stat>
D → <AssStat>
E → <Var>
F → <Exp>
G → <Term>
H → <Constant>
I → <Digit>
J → <Letter>
K → <Ident>
L → <Tail>
M → <CondStat>
N → <WhileStat>
O → <Cond>
P → <RelOp>
Q → <InputStat>
R → <OutputStat>

In The same way all the non terminals of the above grammar are converted into lowercase alphabets and special characters as listed below:

If : a
Then : b
Else : c
Fi : d
While : e
Do : f
Od : g
!= : ~
>= : $
<= : &

read	:	r
write	:	w
:=	:	#
eof	:	!

Now the actual grammar looks like:

S	->	A
A	->	{B}
B	->	C;B | ^ |
C	->	D | M | N | Q | R
D	->	E := F
E	->	K
F	->	G + G | G – G | G * G | G
G	->	E | H
H	->	I | IH
I	->	0 | 1 | 2 | 3
J	->	i| j | k | m | n | x | y | z
K	->	J | JL
L	->	J | I | JL | IL
M	->	aObBcBd | aObBd
N	->	eOfBg
O	->	(F P F)
P	->	= | ~| < | & | > | $
Q	->	rE
R	->	wE

The valid strings given before will be decoded as:

1)
{x#20;y#x1;}

2)
{e(2=x2)fx:=2;}

3)
{a(z+1=2)bj23#3;cy#y;d;}

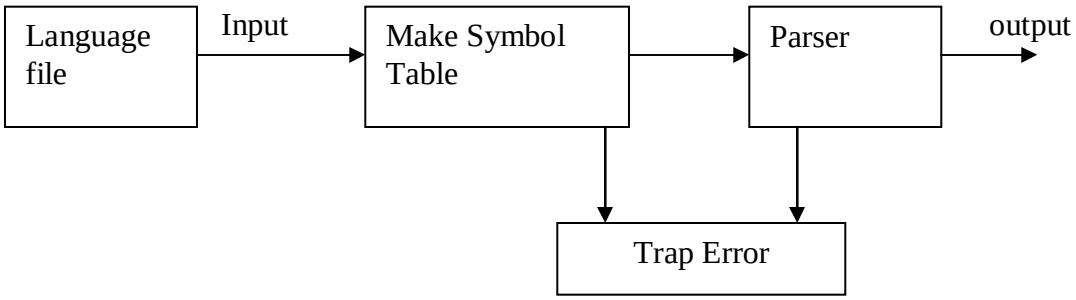
Initially everything will be converted into symbols before passing it to the main parser. At this stage some Errors are also trapped. Errors at this stage are mainly concerned with alphabets not defined in the language. If the language contains “whileb …” then this error is trapped as whileb is not defined in language.

2) Main Parser :

Once the input language is converted into symbols it is passed into the parser to check for the syntax. The syntax is checked according to LR(1) parsing algorithm. It refers to state table to take proper action, i.e shift or goto, and reduction table for reduction of terminals and non terminals.

3) Error Handling:

If the string is not valid then it will display a parse error message and will list the invalid character.



Stages Of Program

IMPORTANT NOTES ON PROGRAM:

Certain files are necessary for the program to execute. Those files are:

1. St_table.txt: This is the data file that contains the entire parsing table. Entries in this file are classified as below:

-1:	this indicates error
500:	sring accepeted
300 and above:	reduction. Eg. 302 means reduction code is 02 and the reduction string is referred from the reduction table.
<300 and >-1:	shift the present symbol into stack and transfer to this state

2. symbols.txt: This file contains all the symbols (terminals and non terminals) used in the program.

3. r_table.txt: This is the reduction table and contains all possible reductions that can take place in the process.

4. lang.txt: This is the language file that contains string to be parsed.

SAMPLE PROGRAMS

1)
{read n;}

OUTPUT:
```
{read n;}_
Encoded String: {rn;}!
SUCCESSFULLY PARSED: STRING ACCEPTED.
```

Reductions:
J<-n
K<-J
E<-K
Q<-rE
C<-Q
B<-C;
A<-{B}

2)
{;}

output
```
{;}_
Encoded String: {;}!
PARSE ERROR: misplaced ; in {;}
```

3)
{
x23 := zzz2212z;}

output:
```
{
x23    := zzz2212z;}
Encoded String: {x23#zzz2212z;}!
SUCCESSFULLY PARSED: STRING ACCEPTED.
```

Reductions:
J<-x
I<-2
I<-3
L<-I
L<-IL
K<-JL
E<-K
J<-z
J<-z

J<-z
I<-2
I<-2
I<-1
I<-2
J<-z
L<-J
L<-IL
L<-IL
L<-IL
L<-IL
L<-JL
L<-JL
K<-JL
E<-K
G<-E
F<-G
D<-E#F
C<-D
B<-C;
A<-{B}

4)
{if (n22 + 1 = 0) then write n;}

output->
```
{if ( n22 + 1 = 0) then write n;}
Encoded String: {a(n22+1=0)bwn;}!

PARSE ERROR: misplaced } in ;}end of file
```

Reductions->
J<-n
I<-2
I<-2
L<-I
L<-IL
K<-JL
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G+G
P<-=

I<-0
H<-I
G<-H
F<-G
O<-(FPF)
J<-n
K<-J
E<-K
R<-wE

5)
{while (i=2) do x:=3;od;}

output->
```
{while (i=2) do x:=3;od;}
Encoded String: {e(i=2)fx#3;g;}!

SUCCESSFULLY PARSED: STRING ACCEPTED.
```

Reductions->
J<-i
K<-J
E<-K
G<-E
F<-G
P<-=
I<-2
H<-I
G<-H
F<-G
O<-(FPF)
J<-x
K<-J
E<-K
I<-3
H<-I
G<-H
F<-G
D<-E#F
C<-D
B<-C;
N<-eOfBg
C<-N
B<-C;
A<-{B}

5)
{
	i:=20;j := 10+i; x := 3-j;
	if (i+1 != j-2) then x23 := x+i; else y11 := 20 fi;
}

output->
```
{
        i:=20;j := 10+i; x := 3-j;
        if (i+1 != j-2) then x23 := x+i; else y11 := 20 fi;
}
Encoded String: {i#20;j#10+i;x#3-j;a(i+1~j-2)bx23#x+i;cy11#20d;}!

PARSE ERROR: misplaced fi in 0fi;
```

Reductions ->
J<-i
K<-J
E<-K
I<-2
I<-0
H<-I
H<-IH
G<-H
F<-G
D<-E#F
C<-D
J<-j
K<-J
E<-K
I<-1
I<-0
H<-I
H<-IH
G<-H
J<-i
K<-J
E<-K
G<-E
F<-G+G
D<-E#F
C<-D
J<-x
K<-J
E<-K
I<-3
H<-I

G<-H
J<-j
K<-J
E<-K
G<-E
F<-G-G
D<-E#F
C<-D
J<-i
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G+G
P<-~
J<-j
K<-J
E<-K
G<-E
I<-2
H<-I
G<-H
F<-G-G
O<-(FPF)
J<-x
I<-2
I<-3
L<-I
L<-IL
K<-JL
E<-K
J<-x
K<-J
E<-K
G<-E
J<-i
K<-J
E<-K
G<-E
F<-G+G
D<-E#F
C<-D
B<-C;
J<-y

I<-1
I<-1
L<-I
L<-IL
K<-JL
E<-K

6)
{
        while (2 = 2) do x:= 30;
                if (x < 10 + M) then x := x+1; else x:=x-1;fi;
                od;
}

output->
{
        while (2 = 2) do x:= 30;
                if (x < 10 + m) then x := x+1; else x:=x-1;fi;
                od;
} _
Encoded String: {e(2=2)fx#30;a(x<10+m)bx#x+1;cx#x-1;d;g;}!

SUCCESSFULLY PARSED: STRING ACCEPTED.

Reductions->
I<-2
H<-I
G<-H
F<-G
P<-=
I<-2
H<-I
G<-H
F<-G
O<-(FPF)
J<-x
K<-J
E<-K
I<-3
I<-0
H<-I
H<-IH
G<-H
F<-G
D<-E#F
C<-D
J<-x
K<-J

E<-K
G<-E
F<-G
P<-<
I<-1
I<-0
H<-I
H<-IH
G<-H
J<-m
K<-J
E<-K
G<-E
F<-G+G
O<-(FPF)
J<-x
K<-J
E<-K
J<-x
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G+G
D<-E#F
C<-D
B<-C;
J<-x
K<-J
E<-K
J<-x
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G-G
D<-E#F
C<-D
B<-C;
M<-aObBcBd
C<-M
B<-C;

B<-C;B
N<-eOfBg
C<-N
B<-C;
A<-{B}

7)
{
	read n;
		if (n+1 <= 0) then write n; else z:= n; m:= n-1;
		while (m > 0) do z := z*n; m := m-1;
			od;
		write z;
	fi;
}

output->
```
{
        read n;
                if (n+1 <= 0) then write n; else z:= n; m:= n-1;
                while (m > 0) do z := z*n; m := m-1;
                        od;
                write z;
        fi;
}_
Encoded String: {rn;a(n+1&0)bwn;cz#n;m#n-1;e(m>0)fz#z*n;m#m-1;g;wz;d;}!

SUCCESSFULLY PARSED: STRING ACCEPTED.
```

reductions->
J<-n
K<-J
E<-K
Q<-rE
C<-Q
J<-n
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G+G
P<-&
I<-0
H<-I
G<-H
F<-G

O<-(FPF)
J<-n
K<-J
E<-K
R<-wE
C<-R
B<-C;
J<-z
K<-J
E<-K
J<-n
K<-J
E<-K
G<-E
F<-G
D<-E#F
C<-D
J<-m
K<-J
E<-K
J<-n
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G-G
D<-E#F
C<-D
J<-m
K<-J
E<-K
G<-E
F<-G
P<->
I<-0
H<-I
G<-H
F<-G
O<-(FPF)
J<-z
K<-J
E<-K
J<-z
K<-J

```
E<-K
G<-E
J<-n
K<-J
E<-K
G<-E
F<-G*G
D<-E#F
C<-D
J<-m
K<-J
E<-K
J<-m
K<-J
E<-K
G<-E
I<-1
H<-I
G<-H
F<-G-G
D<-E#F
C<-D
B<-C;
B<-C;B
N<-eOfBg
C<-N
J<-z
K<-J
E<-K
R<-wE
C<-R
B<-C;
B<-C;B
B<-C;B
B<-C;B
M<-aObBcBd
C<-M
B<-C;
B<-C;B
A<-{B}

********** END**********
```