National Institute Of Technology Surathkal Mangalore Karnataka-575025

Department Of Information Technology

Lab Assignment:- 10

Name:- Chikkeri Chinmaya

Roll Number:- 211IT017

Branch:- Information Technology (B.Tech)

Section :- S13

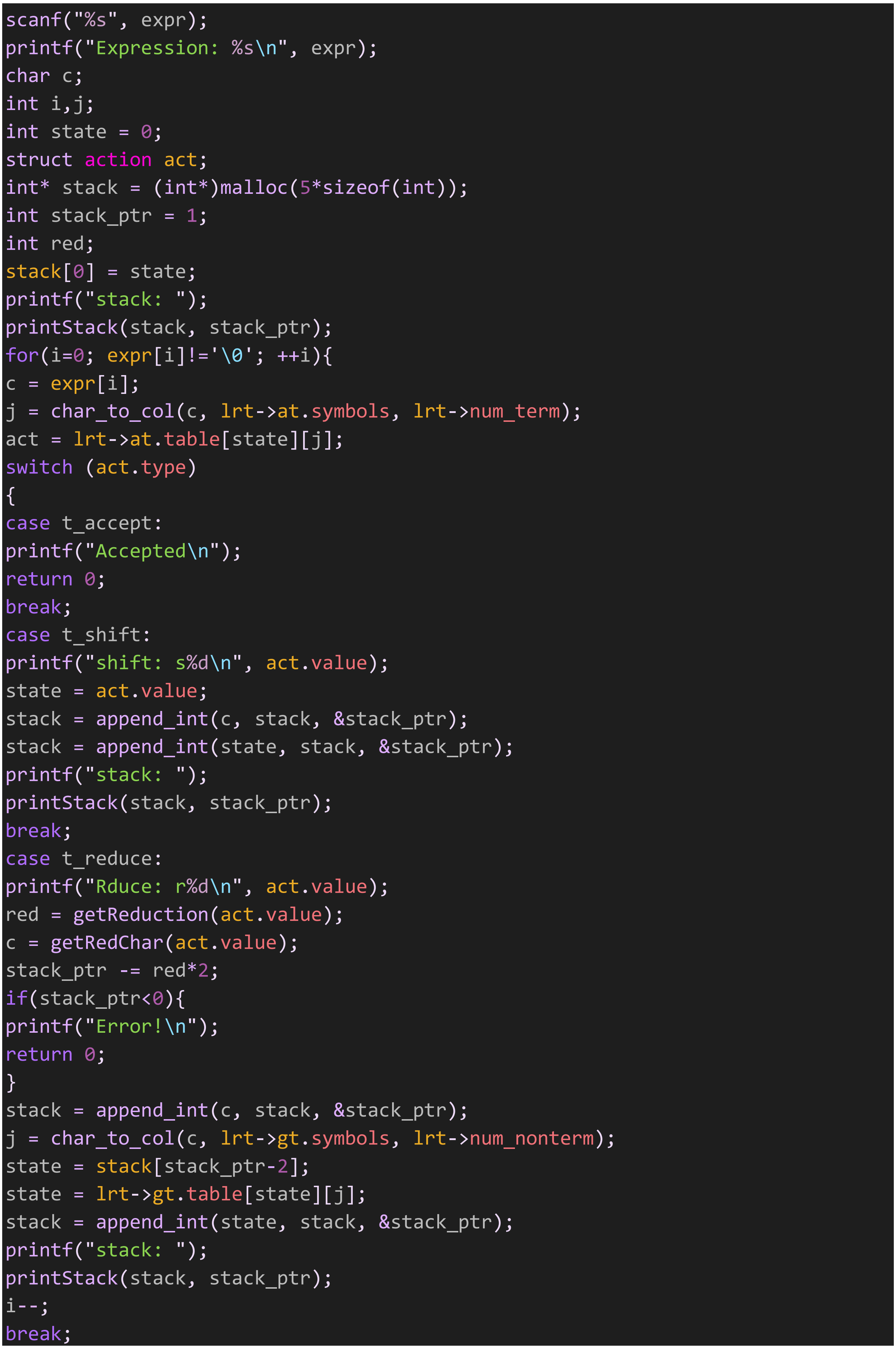
Course:-Automata And Compiler Design (IT252)

Submitted To:-

Anupama H C Mam

Main.cpp

```
#include <stdio.h>
#include "common.h"
#include "table.c"
#define explen 100
struct rule{
char c;
int n;
};
struct rule* rules;
int* append_int(int n, int *arr, int *p);
int getReduction(int k);
char getRedChar(int k);
void printStack(int* stack, int n);
struct rule* appendRule(struct rule r, struct rule* _rules, int p){
if(p>0 && p%5==0){
struct rule* array = (struct rule*)malloc((p+5)*sizeof(struct
rule));
for(int i=0; i<p; ++i){
array[i] = _rules[i];
}
array[p] = r;
return array;
}
_rules[p] = r;
return _rules;
}
int main(){
int num_rules;
printf("How many rules are there ?: ");
scanf("\n%d", &num_rules);
printf("Enter rules properties: left(symbol) right(count). Eg.
{A->Aa}=>{A 2}");
rules = (struct rule*)malloc(5*sizeof(struct rule));
char lhs; int rhs;
struct rule r;
for(int i=0; i<num_rules; ++i){
scanf("\n%c %d", &lhs, &rhs);
r.c = lhs;
r.n = rhs;
rules = appendRule(r, rules, i);
}
struct lr_table* lrt = CreateTable();
// PrintTable(lrt);
PrintTableNice(lrt);
// Scan expression
char expr[explen];
```

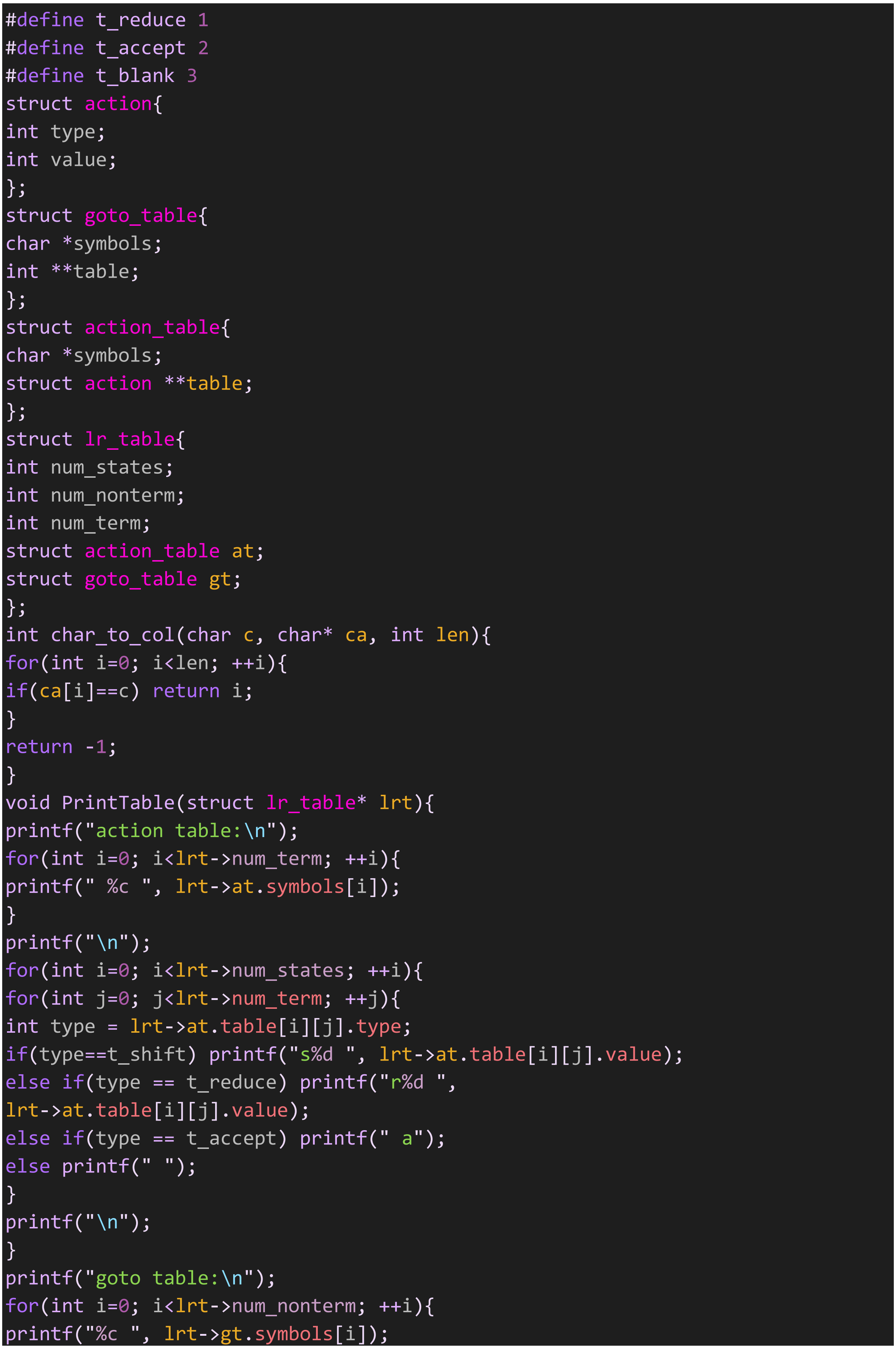


```
scanf("%s", expr);
printf("Expression: %s\n", expr);
char c;
int i,j;
int state = 0;
struct action act;
int* stack = (int*)malloc(5*sizeof(int));
int stack_ptr = 1;
int red;
stack[0] = state;
printf("stack: ");
printStack(stack, stack_ptr);
for(i=0; expr[i]!='\0'; ++i){
c = expr[i];
j = char_to_col(c, lrt->at.symbols, lrt->num_term);
act = lrt->at.table[state][j];
switch (act.type)
{
case t_accept:
printf("Accepted\n");
return 0;
break;
case t_shift:
printf("shift: s%d\n", act.value);
state = act.value;
stack = append_int(c, stack, &stack_ptr);
stack = append_int(state, stack, &stack_ptr);
printf("stack: ");
printStack(stack, stack_ptr);
break;
case t_reduce:
printf("Rduce: r%d\n", act.value);
red = getReduction(act.value);
c = getRedChar(act.value);
stack_ptr -= red*2;
if(stack_ptr<0){
printf("Error!\n");
return 0;
}
stack = append_int(c, stack, &stack_ptr);
j = char_to_col(c, lrt->gt.symbols, lrt->num_nonterm);
state = stack[stack_ptr-2];
state = lrt->gt.table[state][j];
stack = append_int(state, stack, &stack_ptr);
printf("stack: ");
printStack(stack, stack_ptr);
i--;
break;
default:
printf("Error!\n");
return 0;
break;
}
}
return 0;
}
int* append_int(int n, int *arr, int *p){
if(*p>0 && *p%5==0){
int* a = (int*)malloc((*p+5)*sizeof(int));
for(int i=0; i<*p; ++i){
a[i] = arr[i];
}
a[*p] = n;
free(arr);
*p = *p + 1;
return a;
}
arr[*p] = n;
*p = *p + 1;
return arr;
}
int getReduction(int k){
return rules[k-1].n;
}
char getRedChar(int k){
return rules[k-1].c;
}
void printStack(int* stack, int n){
for(int i=0; i<n; ++i){
if((i&1)==0){
printf("%d ", stack[i]);
}
else printf("%c ", stack[i]);
}
printf("\n");
}
```

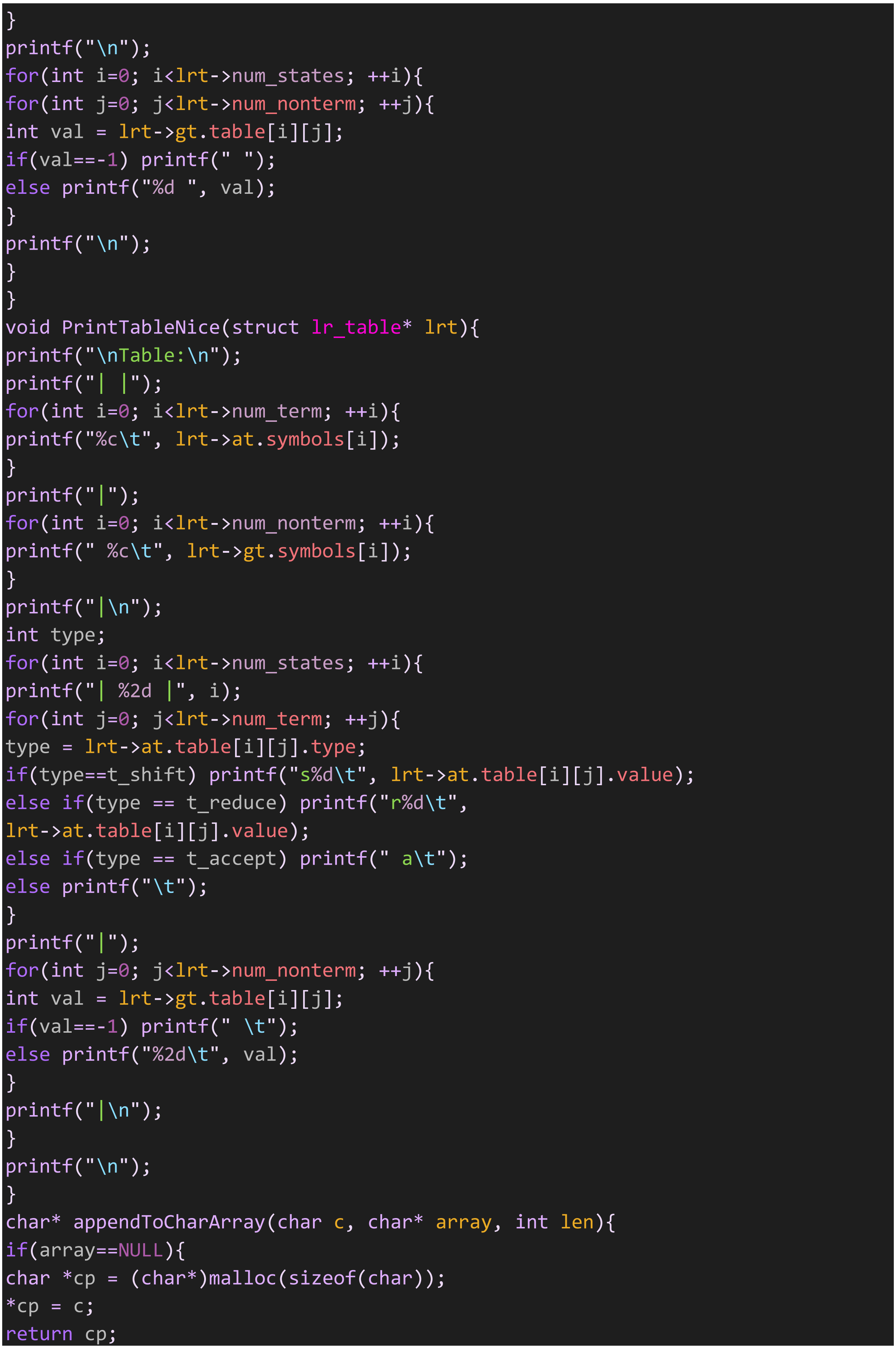
Table. c

Class for taking in the input and creating action and goto table:

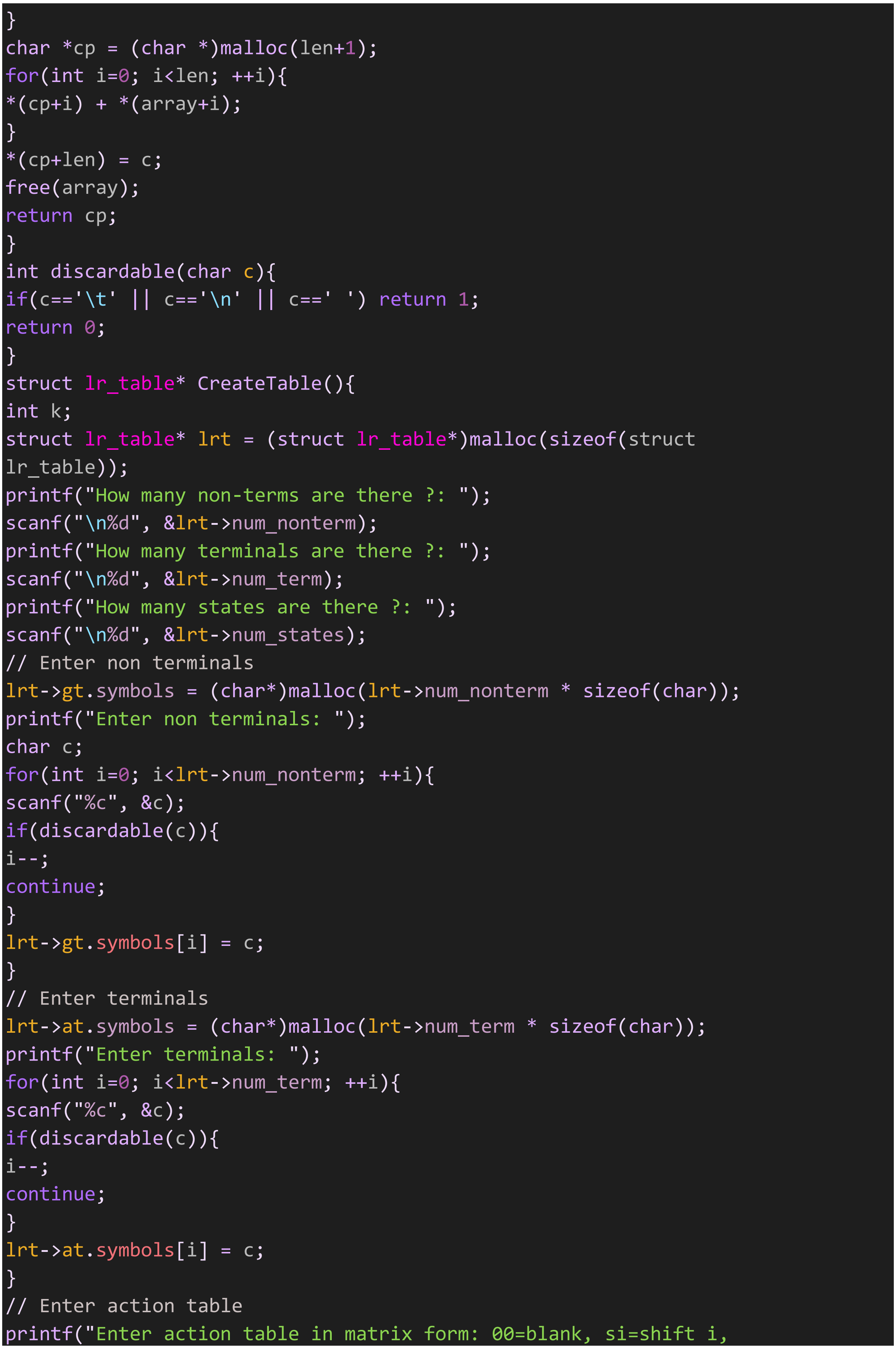
```
#include "common.h"
#define t_shift 0
```



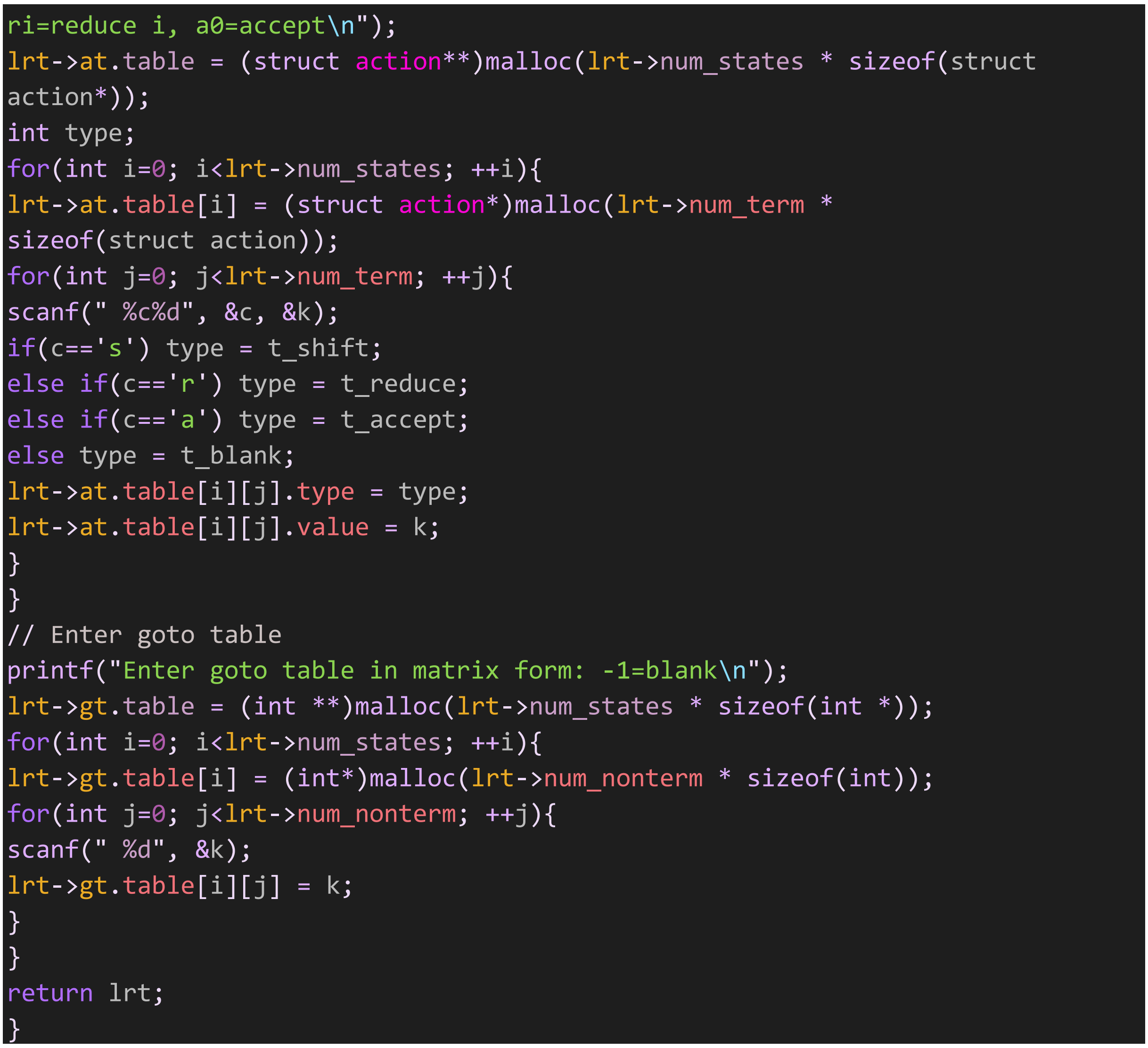
```
#define t_reduce 1
#define t_accept 2
#define t_blank 3
struct action{
int type;
int value;
};
struct goto_table{
char *symbols;
int **table;
};
struct action_table{
char *symbols;
struct action **table;
};
struct lr_table{
int num_states;
int num_nonterm;
int num_term;
struct action_table at;
struct goto_table gt;
};
int char_to_col(char c, char* ca, int len){
for(int i=0; i<len; ++i){
if(ca[i]==c) return i;
}
return -1;
}
void PrintTable(struct lr_table* lrt){
printf("action table:\n");
for(int i=0; i<lrt->num_term; ++i){
printf(" %c ", lrt->at.symbols[i]);
}
printf("\n");
for(int i=0; i<lrt->num_states; ++i){
for(int j=0; j<lrt->num_term; ++j){
int type = lrt->at.table[i][j].type;
if(type==t_shift) printf("s%d ", lrt->at.table[i][j].value);
else if(type == t_reduce) printf("r%d ",
lrt->at.table[i][j].value);
else if(type == t_accept) printf(" a");
else printf(" ");
}
printf("\n");
}
printf("goto table:\n");
for(int i=0; i<lrt->num_nonterm; ++i){
printf("%c ", lrt->gt.symbols[i]);
```



```
}
printf("\n");
for(int i=0; i<lrt->num_states; ++i){
for(int j=0; j<lrt->num_nonterm; ++j){
int val = lrt->gt.table[i][j];
if(val==-1) printf(" ");
else printf("%d ", val);
}
printf("\n");
}
}
void PrintTableNice(struct lr_table* lrt){
printf("\nTable:\n");
printf("| |");
for(int i=0; i<lrt->num_term; ++i){
printf("%c\t", lrt->at.symbols[i]);
}
printf("|");
for(int i=0; i<lrt->num_nonterm; ++i){
printf(" %c\t", lrt->gt.symbols[i]);
}
printf("|\n");
int type;
for(int i=0; i<lrt->num_states; ++i){
printf("| %2d |", i);
for(int j=0; j<lrt->num_term; ++j){
type = lrt->at.table[i][j].type;
if(type==t_shift) printf("s%d\t", lrt->at.table[i][j].value);
else if(type == t_reduce) printf("r%d\t",
lrt->at.table[i][j].value);
else if(type == t_accept) printf(" a\t");
else printf("\t");
}
printf("|");
for(int j=0; j<lrt->num_nonterm; ++j){
int val = lrt->gt.table[i][j];
if(val==-1) printf(" \t");
else printf("%2d\t", val);
}
printf("|\n");
}
printf("\n");
}
char* appendToCharArray(char c, char* array, int len){
if(array==NULL){
char *cp = (char*)malloc(sizeof(char));
*cp = c;
return cp;
```



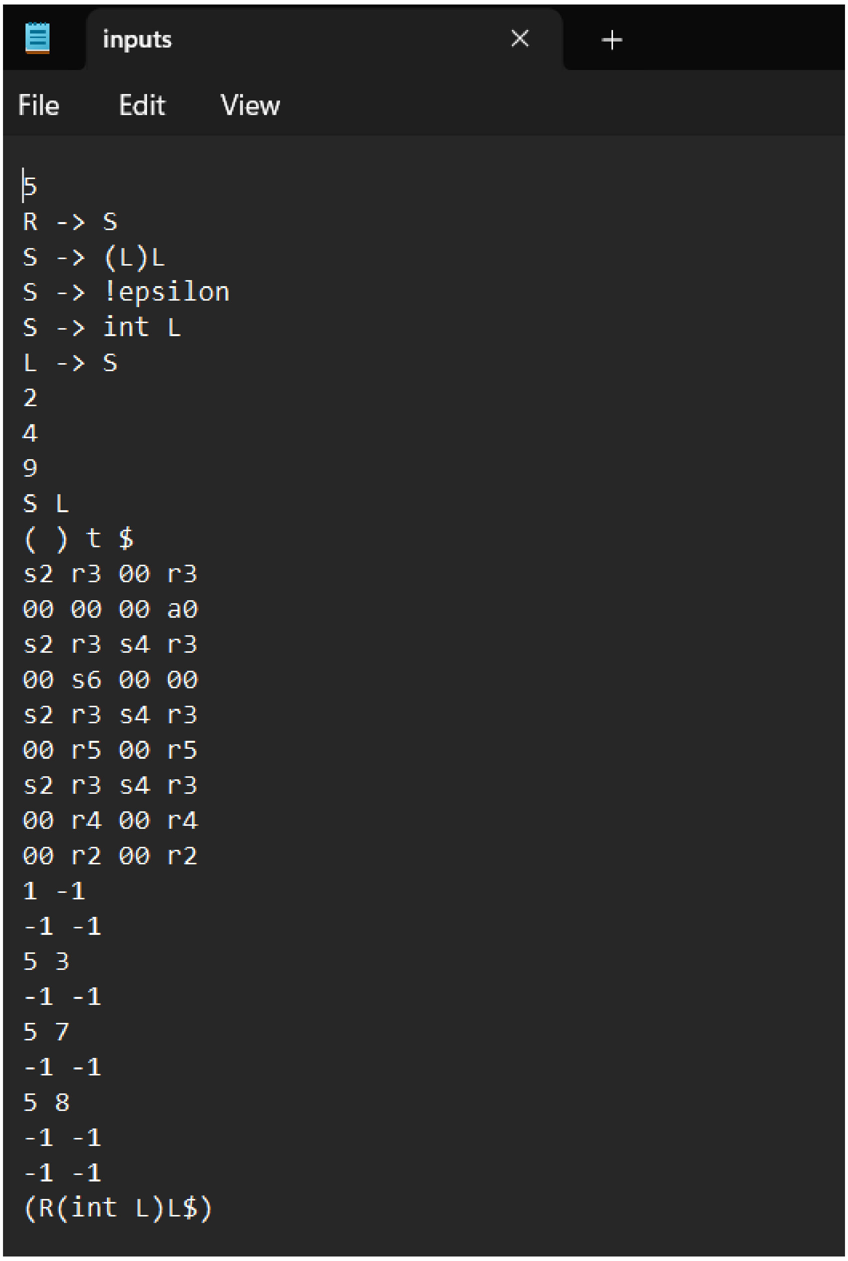
```
}
char *cp = (char *)malloc(len+1);
for(int i=0; i<len; ++i){
*(cp+i) + *(array+i);
}
*(cp+len) = c;
free(array);
return cp;
}
int discardable(char c){
if(c=='\t' || c=='\n' || c==' ') return 1;
return 0;
}
struct lr_table* CreateTable(){
int k;
struct lr_table* lrt = (struct lr_table*)malloc(sizeof(struct
lr_table));
printf("How many non-terms are there ?: ");
scanf("\n%d", &lrt->num_nonterm);
printf("How many terminals are there ?: ");
scanf("\n%d", &lrt->num_term);
printf("How many states are there ?: ");
scanf("\n%d", &lrt->num_states);
// Enter non terminals
lrt->gt.symbols = (char*)malloc(lrt->num_nonterm * sizeof(char));
printf("Enter non terminals: ");
char c;
for(int i=0; i<lrt->num_nonterm; ++i){
scanf("%c", &c);
if(discardable(c)){
i--;
continue;
}
lrt->gt.symbols[i] = c;
}
// Enter terminals
lrt->at.symbols = (char*)malloc(lrt->num_term * sizeof(char));
printf("Enter terminals: ");
for(int i=0; i<lrt->num_term; ++i){
scanf("%c", &c);
if(discardable(c)){
i--;
continue;
}
lrt->at.symbols[i] = c;
}
// Enter action table
printf("Enter action table in matrix form: 00=blank, si=shift i,
```



```
ri=reduce i, a0=accept\n");
lrt->at.table = (struct action**)malloc(lrt->num_states * sizeof(struct
action*));
int type;
for(int i=0; i<lrt->num_states; ++i){
lrt->at.table[i] = (struct action*)malloc(lrt->num_term *
sizeof(struct action));
for(int j=0; j<lrt->num_term; ++j){
scanf(" %c%d", &c, &k);
if(c=='s') type = t_shift;
else if(c=='r') type = t_reduce;
else if(c=='a') type = t_accept;
else type = t_blank;
lrt->at.table[i][j].type = type;
lrt->at.table[i][j].value = k;
}
}
// Enter goto table
printf("Enter goto table in matrix form: -1=blank\n");
lrt->gt.table = (int **)malloc(lrt->num_states * sizeof(int *));
for(int i=0; i<lrt->num_states; ++i){
lrt->gt.table[i] = (int*)malloc(lrt->num_nonterm * sizeof(int));
for(int j=0; j<lrt->num_nonterm; ++j){
scanf(" %d", &k);
lrt->gt.table[i][j] = k;
}
}
return lrt;
}
```

Input file :- (inputs.txt)

Input file :-



```
5
R -> S
S -> (L)L
S -> !epsilon
S -> int L
L -> S
2
4
9
S L
( ) t $
s2 r3 00 r3
00 00 00 a0
s2 r3 s4 r3
00 s6 00 00
s2 r3 s4 r3
00 r5 00 r5
s2 r3 s4 r3
00 r4 00 r4
00 r2 00 r2
1 -1
-1 -1
5 3
-1 -1
5 7
-1 -1
5 8
-1 -1
-1 -1
(R(int L)L$)
```

Grammar:-

R→S

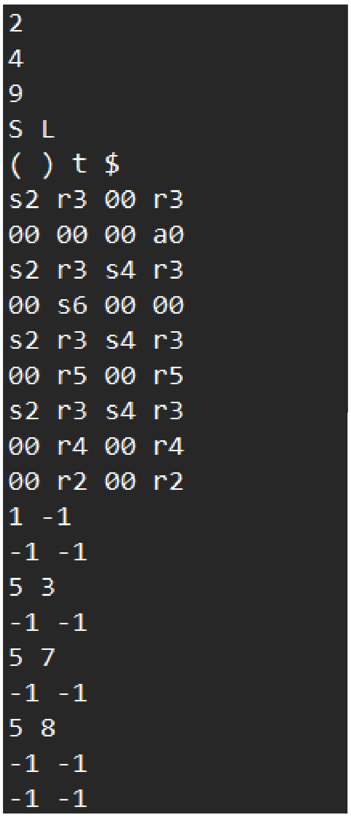
S→(L) L

S→ε

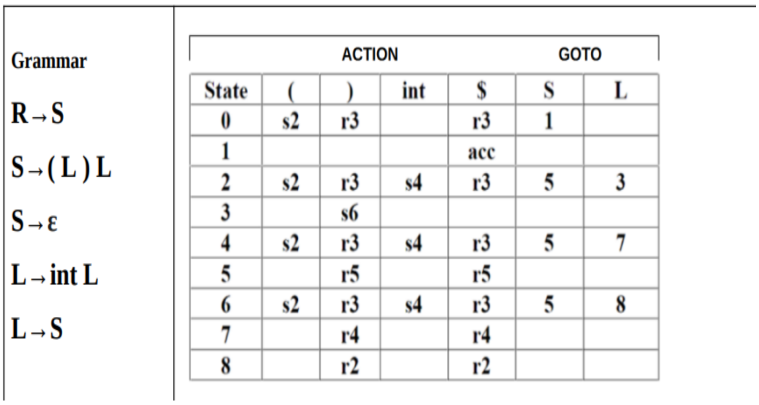
L→int L

L→S

Given The Table As In Put :-



```
2
4
9
S L
( ) t $
s2 r3 00 r3
00 00 00 a0
s2 r3 s4 r3
00 s6 00 00
s2 r3 s4 r3
00 r5 00 r5
s2 r3 s4 r3
00 r4 00 r4
00 r2 00 r2
1 -1
-1 -1
5 3
-1 -1
5 7
-1 -1
5 8
-1 -1
-1 -1
```



Grammar

R→S

S→(L) L

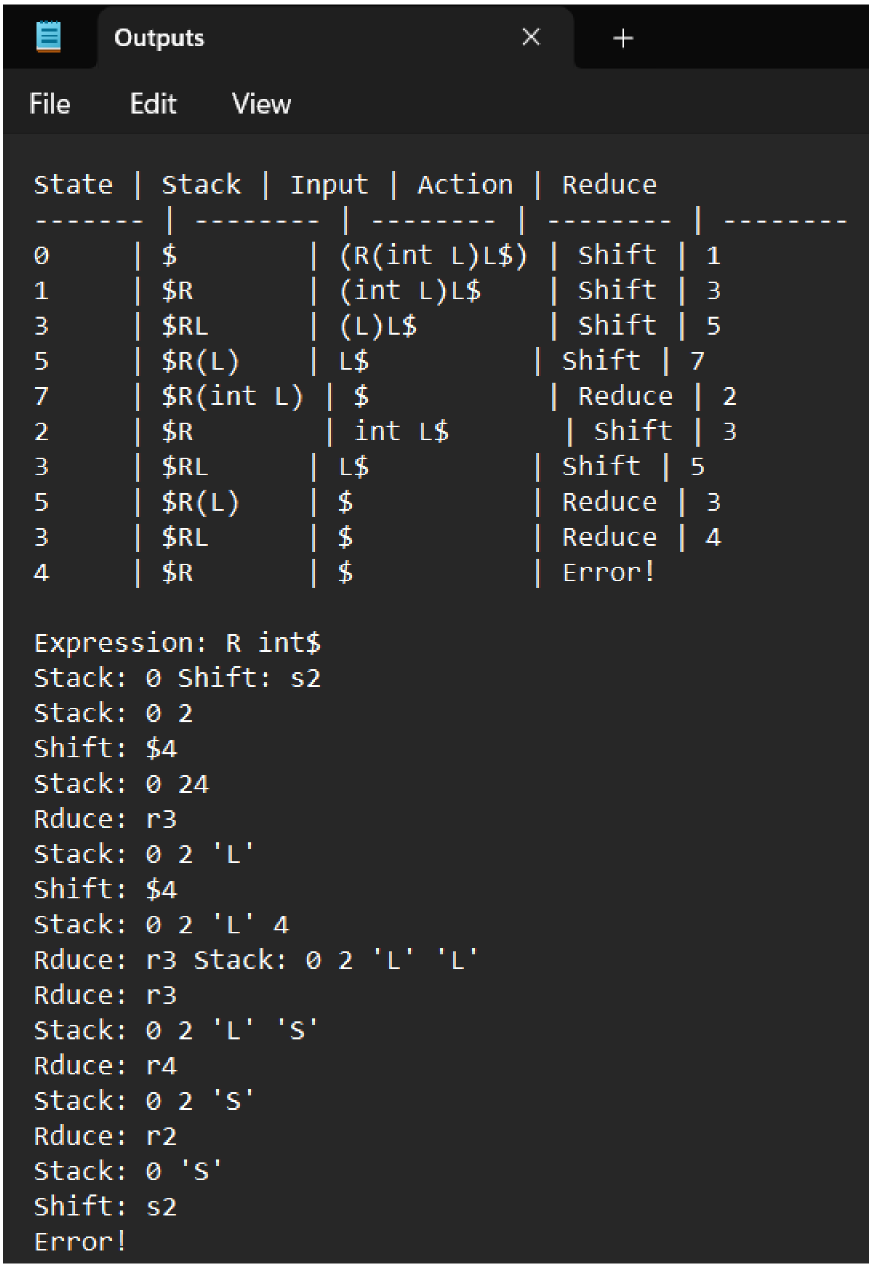
S→ε

L→int L

L→S

	ACTION				GOTO	
State	(	)	int	$	S	L
0	s2	r3		r3	1	
1				acc		
2	s2	r3	s4	r3	5	3
3		s6				
4	s2	r3	s4	r3	5	7
5		r5		r5		
6	s2	r3	s4	r3	5	8
7		r4		r4		
8		r2		r2		

Input string :- R int $



```
State | Stack | Input | Action | Reduce
------- | -------- | -------- | -------- | --------
0     | $        | (R(int L)L$) | Shift | 1
1     | $R       | (int L)L$    | Shift | 3
3     | $RL      | (L)L$        | Shift | 5
5     | $R(L)    | L$          | Shift | 7
7     | $R(int L) | $           | Reduce | 2
2     | $R        | int L$       | Shift | 3
3     | $RL      | L$          | Shift | 5
5     | $R(L)    | $           | Reduce | 3
3     | $RL      | $           | Reduce | 4
4     | $R       | $           | Error!

Expression: R int$
Stack: 0 Shift: s2
Stack: 0 2
Shift: $4
Stack: 0 24
Rduce: r3
Stack: 0 2 'L'
Shift: $4
Stack: 0 2 'L' 4
Rduce: r3 Stack: 0 2 'L' 'L'
Rduce: r3
Stack: 0 2 'L' 'S'
Rduce: r4
Stack: 0 2 'S'
Rduce: r2
Stack: 0 'S'
Shift: s2
Error!
```

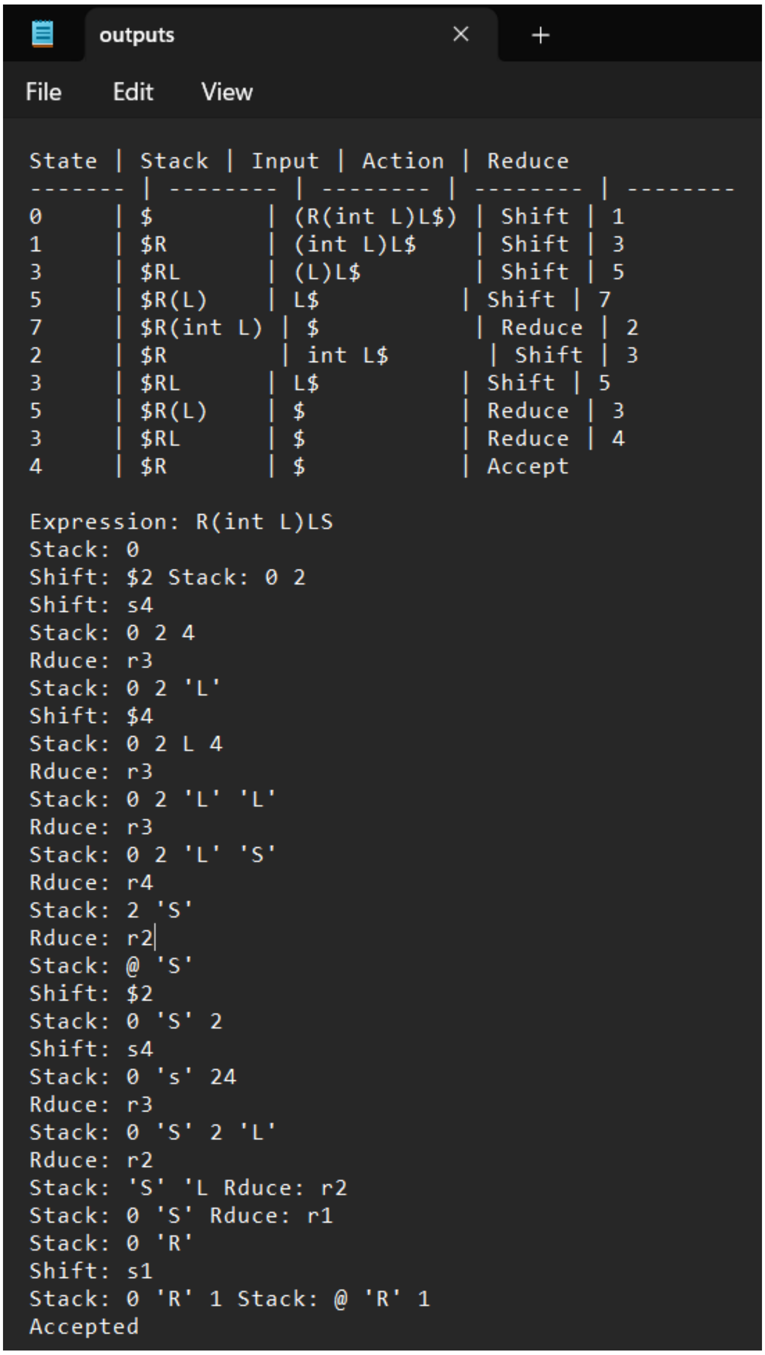
OutPut:-

R int is invalid so the program prints out an error.

Input String :- (R(int L)L$)

OutPut:-

The String R(int id) is accepted.



```
State | Stack | Input | Action | Reduce
------- | -------- | -------- | -------- | --------
0     | $        | (R(int L)L$) | Shift | 1
1     | $R       | (int L)L$    | Shift | 3
3     | $RL      | (L)L$        | Shift | 5
5     | $R(L)    | L$          | Shift | 7
7     | $R(int L) | $           | Reduce | 2
2     | $R        | int L$       | Shift | 3
3     | $RL      | L$          | Shift | 5
5     | $R(L)    | $           | Reduce | 3
3     | $RL      | $           | Reduce | 4
4     | $R       | $           | Accept

Expression: R(int L)LS
Stack: 0
Shift: $2 Stack: 0 2
Shift: s4
Stack: 0 2 4
Rduce: r3
Stack: 0 2 'L'
Shift: $4
Stack: 0 2 L 4
Rduce: r3
Stack: 0 2 'L' 'L'
Rduce: r3
Stack: 0 2 'L' 'S'
Rduce: r4
Stack: 2 'S'
Rduce: r2
Stack: @ 'S'
Shift: $2
Stack: 0 'S' 2
Shift: s4
Stack: 0 's' 24
Rduce: r3
Stack: 0 'S' 2 'L'
Rduce: r2
Stack: 'S' 'L Rduce: r2
Stack: 0 'S' Rduce: r1
Stack: 0 'R'
Shift: s1
Stack: 0 'R' 1 Stack: @ 'R' 1
Accepted
```