/* B10: Write a program to design LALR parsing. */

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
      LALR parser.
      LALR is the same as CLR except two similar states differing only
in lookaheads in CLR are merged in LALR
*/

// @ is null symbol

// structure for representing grammer rules (eg. S -> A)
struct Rules
{
      char var;
      char der[10];
};

// structure for representing LALR items
struct Item
{
      int dotposition;
      struct Rules r;
      int lookahead[255];
      int f;
};

// structure for representing states
struct State
{
      int len;
      struct Item itm[20];
      int transition[255];
};

// Structure for storing a list of states
struct list
{
      struct State data;
      struct list* next;
};

int variables[26] = {0};
int terminals[255] = {0};

int nullable[26] = {0};

char first[26][255] = {{0}}; // Array to store first of each variable
```

```
char follow[26][255] = {{0}};// Array to store follow of each variable

char *var,*term;

char start;

int n,n_var = 0,n_term = 0;
struct Rules* a;

struct list* head,*tail;

// Given a character(variable or terminal) check if its nullable or
not
int is_nullable(char* s)
{
      char* p;
      p = s;
      while(*p!='\0')
      {
            if(*p<'A'||*p>'Z'||!nullable[*p-'A'])
                  return 0;
            p++;
      }
      return 1;
}

// Check if a item is in a given state and return its position
int is_item_in(struct State* l,struct Rules r,int dot)
{
      for(int i=0;i<l->len;i++)
      {
            if((l->itm[i].dotposition==dot)&&(l-
>itm[i].r.var==r.var)&&(strcmp(l->itm[i].r.der,r.der)==0))
                  return i;
      }
      return -1;
}

int is_item_in_advanced(struct State* l,struct Rules r,int dot,int*
bit)
{
      int f = 0;
      for(int i=0;i<l->len;i++)
      {
            f = 1;
            for(int j=0;j<255;j++)
            {
                  if(bit[j]!=l->itm[i].lookahead[j])
                  {
                        f = 0;break;
                  }
```

```
            }
            if(f&&(l->itm[i].dotposition==dot)&&(l-
>itm[i].r.var==r.var)&&(strcmp(l->itm[i].r.der,r.der)==0))
                  return 1;
      }
      return 0;
}

// Fill the look aheads in a gievn item
void fill_lookaheads(int* bit,struct Item* l)
{
      int length = strlen(l->r.der+l->dotposition+1);
      char sto;int f = 0;
      for(int i=l->dotposition+1;i<l->dotposition+length+1;i++)
      {
            if(l->r.der[i]=='\0')
                  continue;
            if(l->r.der[i]<'A'||l->r.der[i]>'Z')
            {
                  bit[l->r.der[i]] = 1;
                  return;
            }
            for(int j=0;j<255;j++)
            {
                  if(first[l->r.der[i]-'A'][j])
                  {
                        bit[j] = 1;
                  }
            }
            sto = l->r.der[i];
            l->r.der[i] = '\0';
            if(!is_nullable(l->r.der+l->dotposition+1))
            {
                  l->r.der[i] = sto;
            }
            else
            {
                  l->r.der[i] = sto;f = 1;break;
            }
      }
      if(!f)
      {
            for(int i=0;i<255;i++)
            {
                  if(l->lookahead[i])
                  bit[i] = 1;
            }
      }
}

// Fill the dot position, look ahead and item of a given state
void build_state(struct State* l)
```

```
{
      int s;
      for(int i=0;i<l->len;i++)
      {
            if(l->itm[i].r.der[l->itm[i].dotposition]>='A'&&l-
>itm[i].r.der[l->itm[i].dotposition]<='Z')
            {
                  //printf("yes\n");
                  for(int j=0;j<n;j++)
                  {
                        if((a[j].var==l->itm[i].r.der[l-
>itm[i].dotposition]))
                        {
                              if((s = is_item_in(l,a[j],0))==-1)
                              {
                                    l->itm[l->len].dotposition = 0;
                                    l->itm[l->len].r = a[j];
                                    l->itm[l->len].f = 0;
                                    memset(l->itm[l-
>len].lookahead,0,255);
                                    fill_lookaheads(l->itm[l-
>len].lookahead,&l->itm[i]);
                                    l->len++;
                              }
                              else
                              {
                                    fill_lookaheads(l-
>itm[s].lookahead,&l->itm[i]);
                              }

                        }
                  }
            }
      }
}

// Print a given list of states
void print_state(struct list* q)
{
      for(int i=0;i<q->data.len;i++)
            {
                  printf("%c :: ",q->data.itm[i].r.var);
                  if(q->data.itm[i].r.der[0]=='@')
                        q->data.itm[i].r.der[0] = '\0';
                  char sto = q->data.itm[i].r.der[q-
>data.itm[i].dotposition];
                  q->data.itm[i].r.der[q->data.itm[i].dotposition] =
'\0';
                  printf("%s.",q->data.itm[i].r.der);
                  q->data.itm[i].r.der[q->data.itm[i].dotposition] =
sto;
```

```
                printf("%s",q->data.itm[i].r.der+q-
>data.itm[i].dotposition);

                printf(" { ");
                for(int j=0;j<255;j++)
                {
                      if(q->data.itm[i].lookahead[j])
                            printf("%c,",(char)j);
                }
                printf(" }\n");
          }
}

// Check if a state is already in the list of states
int state_already_included(struct list* l,struct State* s)
{
      struct list* q;
      q = l;
      int f,rtn = -1;int ind = 0;
      while(q!=NULL)
      {
            f = 0;
            if(q->data.len!=s->len)
            {
                  q = q->next;
                  ind++;
                  continue;
            }
            for(int i=0;i<s->len;i++)
            {
                  if(is_item_in(&q->data,s->itm[i].r,s-
>itm[i].dotposition)==-1)
                  {
                        f = 1;break;
                  }

            }
            if(!f)
            {
                  return ind;
            }
            ind++;q = q->next;
      }
      return -1;
}


int num=0;

// Add look aheads and also merge two states if they are similar
except w.r.t lookaheads
void add_lookaheads(struct list* l,int s,struct State* t)
```

```
{
	struct list* q;
	q = l;
	while(s--)
		q = q->next;
	for(int i=0;i<t->len;i++)
	{
		for(int j=0;j<q->data.len;j++)
		{
			if((t->itm[i].r.var==q->data.itm[j].r.var)&&(strcmp(t-
>itm[i].r.der,q->data.itm[j].r.der)==0)&&(t->itm[i].dotposition==q-
>data.itm[j].dotposition))
			{
				for(int k=0;k<255;k++)
					if(t->itm[i].lookahead[k])
						q->data.itm[j].lookahead[k] = 1;
				break;
			}
		}
	}
}


// Find out all the states and the their transitions
void find_out_states(struct list* l)
{
	if(l==NULL)
		return;
	for(int i=0;i<l->data.len;i++)
	{
		if(l->data.itm[i].f)
			continue;
		else if(l->data.itm[i].dotposition==strlen(l-
>data.itm[i].r.der))
		{
			l->data.itm[i].f = 1;
			continue;
		}

		struct list* t;
		t = (struct list*)malloc(sizeof(struct list));
		for(int ind=0;ind<255;ind++)
		{
			t->data.transition[ind] = -1;
		}
		t->data.len = 1;
		t->data.itm[0].dotposition = l->data.itm[i].dotposition+1;
		t->data.itm[0].r =  l->data.itm[i].r;
		for(int ind=0;ind<255;ind++)
				t->data.itm[0].lookahead[ind] = l-
>data.itm[i].lookahead[ind];
		l->data.itm[i].f = 1;
```

```
            for(int j=i+1;j<l->data.len;j++)
            {
                  if(l->data.itm[j].r.der[l-
>data.itm[j].dotposition]==l->data.itm[i].r.der[l-
>data.itm[i].dotposition])
                  {

                        t->data.itm[t->data.len].dotposition = l-
>data.itm[j].dotposition+1;
                        t->data.itm[t->data.len].r =  l->data.itm[j].r;
                        memset(t->data.itm[t->data.len].lookahead,0,255);
                        for(int ind=0;ind<255;ind++)
                              t->data.itm[t->data.len].lookahead[ind] =
l->data.itm[j].lookahead[ind];
                        l->data.itm[j].f = 1;
                        t->data.len++;

                  }
            }
            build_state(&t->data);
            int s;
            if((s = state_already_included(head,&t->data))==-1)
            {
                  tail->next = t;
                  tail = t;
                  tail->next = NULL;
                  l->data.transition[l->data.itm[i].r.der[l-
>data.itm[i].dotposition]] = num;
                  num++;
                  for(int ii=0;ii<t->data.len;ii++)
                  {
                        if(t->data.itm[i].r.der[0]=='@')
                              t->data.itm[i].r.der[0] = '\0';
                  }

            }
            else
            {
                  l->data.transition[l->data.itm[i].r.der[l-
>data.itm[i].dotposition]] = s;


                  // the follwing function has to be implemented

                  print_state(t);

                  struct list *q = head;

                  add_lookaheads(head,s,&t->data);
            }
      }
      find_out_states(l->next);
```

```
}

struct Table
{
       char op;
       int state_no;
};

// Given a character find if it is terminal or variable
int find(char c)
{
      for(int i=0;i<n_term;i++)
            if(term[i]==c)
                  return i;
      for(int i=0;i<n_var;i++)
            if(var[i]==c)
                  return n_term+i;
}

// Find a gievn rule in the grammer
int find_rule(struct Rules r)
{
      for(int i=0;i<n;i++)
      {
            if(a[i].var==r.var&&strcmp(a[i].der,r.der)==0)
                  return i+1;
      }
      return -1;
}

// Construct LALR table
void construct_table(struct Table** tab,int num)
{
      struct list* q;int k;
      q = head;
      for(int i=0;i<num;i++)
      {
            for(int j=0;j<255;j++)
            {
                  if(q->data.transition[j]!=-1)
                  {
                        k = find(j);
                        if(j>='A'&&j<='Z')
                        {
                              tab[i][k].state_no = q->data.transition[j];
                        }
                        else
                        {
                              tab[i][k].op = 'S';
                              tab[i][k].state_no = q->data.transition[j];
                        }
                  }
```

```
            }

            for(int j=0;j<q->data.len;j++)
            {

                  if(q->data.itm[j].dotposition==strlen(q-
>data.itm[j].r.der))
                  {
                        if(q->data.itm[j].r.var=='#')
                        {
                              k = find('$');
                              tab[i][k].op = 'A';
                              tab[i][k].state_no = 0;continue;
                        }
                        int nn = find_rule(q->data.itm[j].r);
                        for(int l=0;l<255;l++)
                        {
                              if(q->data.itm[j].lookahead[l])
                              {
                                    k = find(l);
                                    if(tab[i][k].state_no==-1)
                                    {
                                          tab[i][k].op = 'R';
                                          tab[i][k].state_no = nn;
                                    }
                                    else
                                    {
                                                printf("A Shift-Reduce
conflict has taken place in state: %d\n",i);
                                                printf("The operators
involved are: %c (for shift), %c (for reduce)\n",term[k],a[nn-
1].der[1]);
                                                printf("Press 1. for shift
2. for reduce\n");
                                                int d;

      scanf("%d",&d);while(getchar()!='\n');
                                                if(d==2)
                                                {
                                                      tab[i][k].op = 'R';
                                                      tab[i][k].state_no =
nn;
                                                }
                                    }
                              }
                        }
                  }
            }
            q = q->next;
      }
```

```
}


int main(int argc, char const *argv[])
{
      // Input
      if(argc<2)
      {
            printf("Usage: %s [STARTING SYMBOL]\n",argv[0]);
            exit(0);
      }
      printf("Enter the no of rules\n");
      scanf("%d",&n);
      while(getchar()!='\n');
      a = (struct Rules*)malloc(sizeof(struct Rules)*n);
      for(int i=0;i<n;i++)
      {
            printf("Enter the variable\n");
            scanf("%c",&a[i].var);
            if(variables[a[i].var-'A'] != 1)
            {
                  variables[a[i].var-'A'] = 1;n_var++;
            }
            while(getchar()!='\n');
            printf("Enter the derivation\n");
            scanf("%s",a[i].der);
            for(int j=0;j<strlen(a[i].der);j++)
            {

      if(a[i].der[j]!='@'&&(a[i].der[j]<'A'||a[i].der[j]>'Z')&&terminal
s[a[i].der[j]] != 1)
                  {
                        terminals[a[i].der[j]] = 1;n_term++;
                  }
            }
            while(getchar()!='\n');
      }



      var = (char*)malloc(sizeof(char)*n_var);int ind = 0;
      for(int i=0;i<26;i++)
      {
            if(variables[i])
            var[ind++] = 'A'+i;
      }
      n_term++;
      term = (char*)malloc(sizeof(char)*(n_term));ind = 0;
      for(int i=0;i<255;i++)
      {
            if(terminals[i])
            term[ind++] = (char)i;
```

```
	}

	term[ind++] = '$';


	// # is the starting dummy symbol for S'

	// calculating the nullable
	// for the derivation S -> A, S is nullable if either S gives a
null directly(in some other derivation) or if A is nullable
	int no_change = 0;

	do
	{
		no_change = 0;
		for(int i=0;i<n;i++)
		{
			// Check if it is directly nullable
			if(strlen(a[i].der)==1&&a[i].der[0]=='@')
			{
				if(!nullable[a[i].var-'A'])
				{
					no_change = 1;
					nullable[a[i].var-'A'] = 1;
				}
			}
			// Else check if the RHS is nullable
			else if(is_nullable(a[i].der))
			{
				if(!nullable[a[i].var-'A'])
				{
					no_change = 1;
					nullable[a[i].var-'A'] = 1;
				}
			}
		}
	}while(no_change);

	// calculating the first
	// if the first character of a derivation is a terminal then we
have found our first, else we find first of the first non-nullable
variable which will the first of the lhs also

	do
	{
		no_change = 0;
		for(int i=0;i<n;i++)
		{
			if(a[i].der[0]!='@')
			{
				if(a[i].der[0]>='A'&&a[i].der[0]<='Z')
				{
```

```
                              char sto;
                              for(int j=0;j<strlen(a[i].der);j++)
                              {
                                    sto = a[i].der[j];
                                    a[i].der[j] = '\0';
                                    if(is_nullable(a[i].der))
                                    {
                                          a[i].der[j] = sto;
                                          if(sto>='A'&&sto<='Z')
                                          {
                                                for(int k=0;k<255;k++)
                                                {
                                                      if(first[sto-
'A'][k]&&!first[a[i].var-'A'][k])
                                                      {
                                                            no_change = 1;
                                                            first[a[i].var-
'A'][k] = 1;
                                                      }
                                                }
                                          }
                                          else if(!first[a[i].var-
'A'][sto])
                                          {
                                                no_change = 1;
                                                first[a[i].var-'A'][sto] =
1;
                                                break;
                                          }
                                    }
                                    else
                                    {
                                          a[i].der[j] = sto;
                                          break;
                                    }
                              }
                        }
                        else if(!first[a[i].var-'A'][a[i].der[0]])
                        {
                              no_change = 1;
                              first[a[i].var-'A'][a[i].der[0]] = 1;
                              break;
                        }
                  }
            }
      }while(no_change);


      // finding the follow
      start = argv[1][0];

      follow[start-'A']['$'] = 1; //sentinel
```

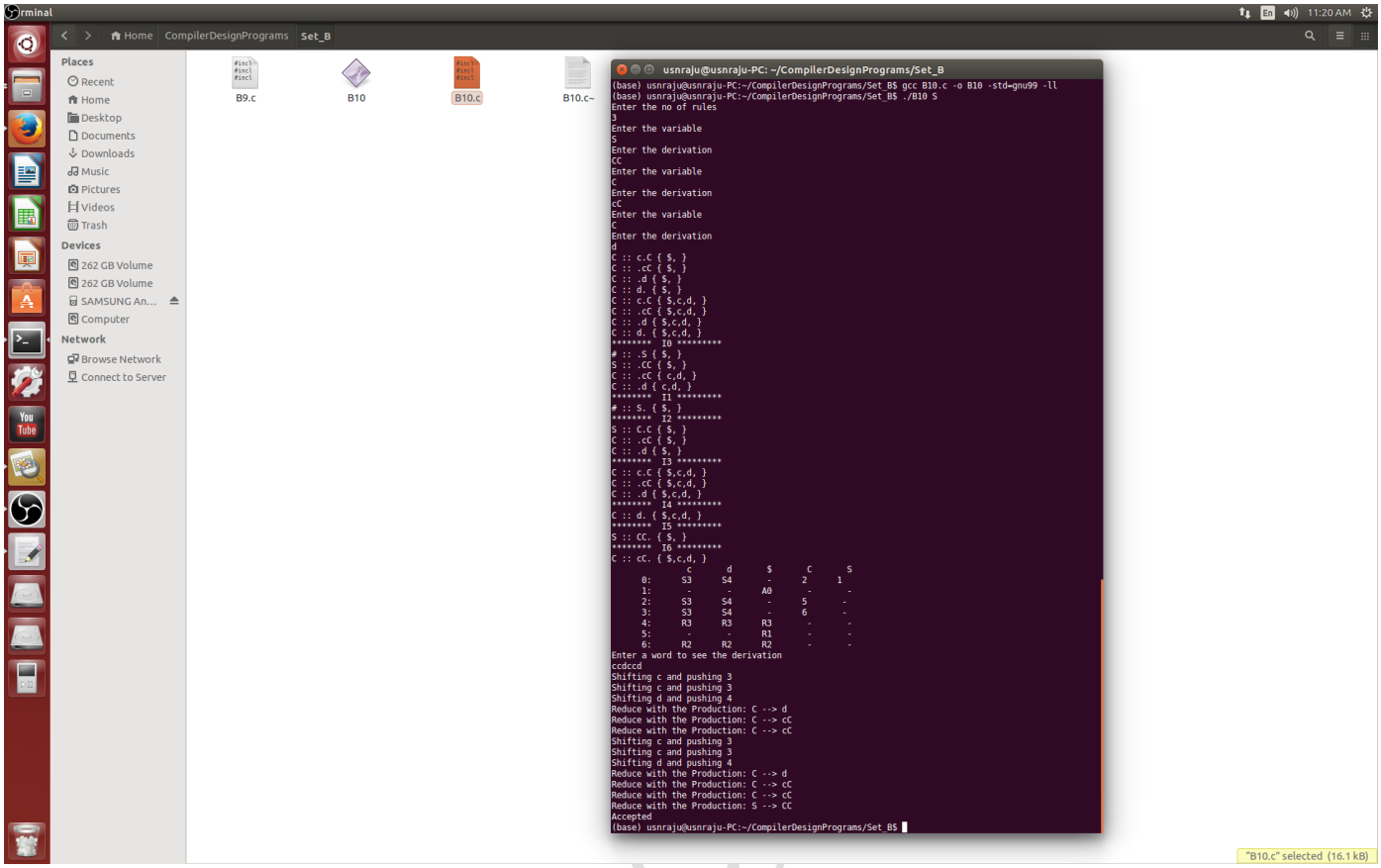
```
	do
	{
		no_change = 0;

		for(int i=0;i<n;i++)
		{
			if(a[i].der[0]!='@')
			{
				for(int j=strlen(a[i].der)-1;j>=0;j--)
				{
					// if the suffix is nullable


	if(a[i].der[j]>='A'&&a[i].der[j]<='Z'&&is_nullable(a[i].der+j+1))
					{
						for(int k=0;k<255;k++)
						{
							if(follow[a[i].var-
'A'][k]&&!follow[a[i].der[j]-'A'][k])
							{
								no_change = 1;
								follow[a[i].der[j]-'A'][k]
= 1;
							}
						}
					}
					if(a[i].der[j]>='A'&&a[i].der[j]<='Z')
					for(int k=j+1;k<strlen(a[i].der);k++)
					{
						char sto = a[i].der[k];
						a[i].der[k] = '\0';

						if(is_nullable(a[i].der+j+1))
						{
							a[i].der[k] = sto;
							if(sto>='A'&&sto<='Z')
							{
								for(int l=0;l<255;l++)
								{
									if(first[sto-
'A'][l]&&!follow[a[i].der[j]-'A'][l])
									{
										no_change = 1;

	follow[a[i].der[j]-'A'][l] = 1;
									}
								}
							}
							else
							{
								if(!follow[a[i].der[j]-
'A'][sto])
```

```
                                                {
                                                      no_change = 1;
                                                      follow[a[i].der[j]-
'A'][sto] = 1;
                                                      break;
                                                }
                                          }
                                    }
                                    else
                                    {
                                          a[i].der[k] = sto;break;
                                    }
                              }
                        }
                  }
            }
      }while(no_change);

      // all prerocessing done!! now the actual part

      // Creating a state diagram from the clr items
      head = (struct list*)malloc(sizeof(struct list));
      tail = head;
      head->data.len = 1;
      //head->data.itm = (struct Item*)malloc(sizeof(struct
Item)*(n+1));

      head->data.itm[0].r.var = '#';

      head->data.itm[0].r.der[0] = start;
      head->data.itm[0].r.der[1] = '\0';

      head->data.itm[0].dotposition = 0;
      head->data.itm[0].f = 0;
      memset(head->data.itm[0].lookahead,0,255);
      head->data.itm[0].lookahead['$'] = 1;



      // Create initial state
      for(int i=0;i<255;i++)
      {
            head->data.transition[i] = -1;
      }
      build_state(&head->data);
      struct list* q;
      q = head;
      for(int i=0;i<q->data.len;i++)
            {
                  if(q->data.itm[i].r.der[0]=='@')
                        q->data.itm[i].r.der[0] = '\0';
            }
```

```
	head->next = NULL;

	// Find out all the states and print them
	tail = head;num++;
	find_out_states(head);
	q = head;int num1 = 0;
	while(q!=NULL)
	{
		printf("********  I%d *********\n",num1);
		print_state(q);
		q = q->next;
		num1++;
	}

	// From the states create the CLR table
	struct Table** tab;
	tab = (struct Table**)malloc(sizeof(struct Table*)*num);
	for(int i=0;i<num;i++)
	{
		tab[i] = (struct Table*)malloc(sizeof(struct
Table)*(n_var+n_term));
		for(int j=0;j<n_var+n_term;j++)
		{
			tab[i][j].state_no = -1;
		}
	}
	for(int i=0;i<n;i++)
		if(a[i].der[0]=='@')
			a[i].der[0] = '\0';
	construct_table(tab,num);

	printf("%8s"," ");
	for(int i=0;i<n_term;i++)
	{
		printf("%8c",term[i]);
	}
	//printf("\n");
	for(int i=0;i<n_var;i++)
		printf("%8c",var[i]);
	printf("\n");
	for(int i=0;i<num;i++)
	{
		printf("%7d:",i);
		for(int j=0;j<n_term+n_var;j++)
		{
			if(tab[i][j].state_no!=-1)
			{
				printf("%7c%d",tab[i][j].op,tab[i][j].state_no);
			}
			else
```

```
                        printf("%8s","-");
                }
                printf("\n");
        }



        char word[100];

        int stack[500],top = -1;
        ind = 0;



        // Parse the word to see if it can be derived from the given LALR
grammer
        printf("Enter a word to see the derivation\n");

        scanf("%s",word);

        strcat(word,"$");


        stack[++top] = 0;

        while(1)
        {
                int ff = find(word[ind]);
                if(tab[stack[top]][ff].state_no==-1)
                {
                        printf("ERROR While parsing!\n");exit(0);
                }
                if(tab[stack[top]][ff].op=='S')
                {
                        printf("Shifting %c and pushing
%d\n",word[ind],tab[stack[top]][ff].state_no);
                        stack[top+1] = term[ff];
                        stack[top+2] =
tab[stack[top]][ff].state_no;top+=2;ind++;
                }
                else if(tab[stack[top]][ff].op=='A')
                {
                        printf("Accepted\n");break;
                }
                else
                {
                        char sto = a[tab[stack[top]][ff].state_no-1].var;
                        printf("Reduce with the Production: %c -->
%s\n",a[tab[stack[top]][ff].state_no-
1].var,a[tab[stack[top]][ff].state_no-1].der);
```

```
                top = top-2*strlen(a[tab[stack[top]][ff].state_no-
    1].der);
                stack[top+1] = sto;
                stack[top+2] =
    tab[stack[top]][find(sto)].state_no;top+=2;
           }
      }



      // for(int i=0;i<n;i++)
      // {
      //   printf("%c :: %s\n",a[i].var,a[i].der);
      // }

      // for(int i=0;i<26;i++)
      // {
      //   if(variables[i]==1)
      //   {
      //     printf("%c: ",(char)(i+'A'));
      //     for(int j=0;j<255;j++)
      //     {
      //       if(first[i][j])
      //       printf("%c, ",(char)(j));
      //     }
      //     printf("\n");
      //   }
      // }


      // for(int i=0;i<26;i++)
      // {
      //   if(variables[i]==1)
      //   {
      //     printf("%c: ",(char)(i+'A'));
      //     for(int j=0;j<255;j++)
      //     {
      //       if(follow[i][j])
      //       printf("%c, ",(char)(j));
      //     }
      //     printf("\n");
      //   }
      // }

      return 0;
    }
```



```
3
S
CC
C
cC
C
d


5
S
AB
A
@
A
a
B
@
B
b

4
E
```

```
E+E
E
E*E
E
(E)
E
i

10
E
TA
E
T
A
+TA
A
@
T
FR
T
F
R
*FR
R
@
F
(E)
F
i

8
E
E+T
E
T
T
T*F
T
F
F
G^F
F
G
G
(E)
G
i

4
S
AB
A
```

```
aAb
A
a
B
d

                                                (       )       *       +
^       i       $       E       F       G       T
                0:      S5      -       -       -       -       S6
-       1       3       4       2
                1:      -       -       -       S7      -       -
A0      -       -       -       -
                2:      -       R2      S8      R2      -       -
R2      -       -       -       -
                3:      -       R4      R4      R4      -       -
R4      -       -       -       -
                4:      -       R6      R6      R6      S9      -
R6      -       -       -       -
                5:      S5      -       -       -       -       S6
-       10      3       4       2
                6:      -       R8      R8      R8      R8      -
R8      -       -       -       -
                7:      S5      -       -       -       -       S6
-       -       3       4       11
                8:      S5      -       -       -       -       S6
-       -       12      4       -
                9:      S5      -       -       -       -       S6
-       -       13      4       -
            10:         -       S14     -       S7      -       -
-       -       -       -       -
            11:         -       R1      S8      R1      -       -
R1      -       -       -       -
            12:         -       R3      R3      R3      -       -
R3      -       -       -       -
            13:         -       R5      R5      R5      -       -
R5      -       -       -       -
            14:         -       R7      R7      R7      R7      -
R7      -       -       -       -


*/
```