```
/*
All Fixed Boundary One-Dimensional Uniform Cellular Automata (CA)
Program. Version 1.0.

Work performed during stay of Arnab Mitra at “Gheorghe Asachi”
Technical University of Iasi, Iasi, Romania under supervision of
Prof. H.-N. Teodorescu.  A One-Dimensional Cellular Automata program
at null boundary condition was developed in India, under supervisions
of Dr. Anirban Kundu. Proposal for the consideration of the several
fixed boundary conditions in the investigation of uniform CA dynamics
is from Prof. H.-N. Teodorescu.

Acknowledgment:
We acknowledge helps from Dr. N. Saharia towards writing this code.

This program will take number of cells as input and will produce
transitions with all 256 CA rules in uniform CA scenario at several
fixed boundary conditions. Initial CA states are fixed to zero. This
version is designed for generating all attractors containing all
possible states upto automata size 20.

The code was used for obtaining results reported in the following
papers. The code can be used under the Commons license, but users using
it for research publications should quote it as the reference:

A. Mitra, H.-N. Teodorescu, Detailed Analysis of Equal Length Cellular
Automata with Fixed Boundaries, Journal of Cellular Automata, 2016,
(Accepted).

*/

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>

void main()
{
     FILE *fp;
     fp=fopen("file name","w");

     int cell,  maxstates, seed, m, n; //n--> number of cells


     int i, s, j, e, y, t, c;

     printf("\nEnter the number of cells in the Cell ( between 1 to
20):");
     scanf("%d", &n);
     if ( n<=20 && n>=1)
          {
                cell=n;
                }
        else
                {
                        printf("\nCheck the number of cells. It is
beyond range.");
                        exit(1);
                  }
                  m=0; // Starting with CA rule 0

        A:        m++;

        fprintf(fp,"\n\n");

        for (; m<=256; ) // Total 256 different elementary CA rules form
CA rule 0 to rule 255
                  {
        printf("M is = %d\n", m);
        fprintf(fp,"\nHomogeneous CA with Rule %d\n", m-1);
        maxstates= (int) pow(2, cell);
        printf("Number of Maximum states = %d\n", maxstates);

        //r--> state value selectoin in CA, STATES--> storage space for
maxstates
        int temp[cell], r[cell+2], STATES[maxstates], x[cell];


/* CA boundary delclaration*/

        r[0]=0; // left fixed boundray 0 or 1
        r[cell+1]=0; // right fixed boundray 0 or 1

        printf("\nInitialization of STATES ....");
        for(i=0;i<=maxstates-1;i++)
        {
                STATES[i]=i;
                printf("%d STATES ....[Done]\n", i);
        }

        //RMT space allocation for ca rules with boundary condition

        int RMT[cell][10] ;


        for(i=0;i<=cell-1;i++)
        {
                x[i]=m-1;
                printf("\nRule is:%d",x[i]);

//decimal to binary conversion for CA rule
                for (j=7;j>=0;j--)
                {
                    if(x[i]%2==0)
                {
                        RMT[i][j]=0;
                        x[i]=(int) x[i]/2;
                        }
                else
                {
                        RMT[i][j]=1;
                        x[i]= (int) x[i]/2;
                        }
                }
   }

        seed=0; // Initial cell value set
        if(seed >=0 && seed <=maxstates-1)
                {
                STATES[0]=seed;
                s=STATES[0];
                }
                else
                {
                        printf("\nNot a valid initial state. Please
enter a value between 0-%d", maxstates-1);
                        exit(1);
                  }
//First state computation
        for (i=cell;i>=1;i--)
            {
                {
                if(s%2==0)
                r[i]=0;
                else
                r[i]=1;

                s=s/2;
                }
            }


   for(j=0;j<maxstates;)
        {
        fprintf(fp,"\nAttractor %d : ", j+1);
         for(e=0;e<=maxstates;e++)
                {
                // Next state computation
                int q=0;
                for(i=1;i<=cell;i++)
                {
                        q=((r[i-1]*4)+(r[i]*2)+(r[i+1]));
                        temp[i-1]=RMT[i-1][7-q];
                }
```

```
                printf("\n");
                for(i=1;i<=cell;i++)
                      {
                            r[i]=temp[i-1];
                      }
     //Binary to decimal conversion of next state
               y=0;
                 t=1;
                for(i=0;i<cell;i++)
                      {
                      y=y+r[cell-i]*t;
                      t=t*2;
                      }
                STATES[e]=y;
                      printf(" %d",STATES[e]);
                      fprintf(fp," %d",STATES[e]);

                for(i=0;i<e;i++)
                {
                      if(STATES[e]==STATES[i])
                {
                      STATES[e]=j;
                      for(i=0;i<e;i++)
                      {
                            if(STATES[e]==STATES[i])
                                  {
                                  if(j==maxstates+1)
                                        goto A;
                                  else{
                                        j++;
                                        i=-1;
                                        STATES[e]=j;
                                      }
                                  }
                      }
                            j++;
                            s=STATES[e];
                            if(j==maxstates+1)
                            {
                            goto A;
                            }

                            for (i=cell;i>0;i--)
                                  {
                                  if(s%2==0)
                                        r[i]=0;
                                  else
                                        r[i]=1;
                                        s=s/2;
                                  }

                                  fprintf(fp,"\nAttractor %d : ", j);
                            printf("\n\n\n");
                            printf(" %d",STATES[e]);
                            fprintf(fp,"%d", STATES[e]);
                                       }
                 }

           }
}

                   }
     fclose(fp);
     getche();
}
```