

```

#include <iostream>
#include <cmath>
#include <map>
#include <cstdlib>
#include <deque>
#include <vector>

using namespace std;

deque<int> new_life;
deque<int> old_life;
deque<int> living;
deque<int> living_candidats;
deque<int> killing_candidats;
vector<int> life_pop;
vector<int> life_exp;
int roww, coll;
int **MM;
int *rr;

void print_matrix(int **M, int row, int col)
{
    for(int i=0;i<row;i++)
    {
        for(int u=0;u<col;u++)
        {
            if(M[i][u]==1)
            {
                cout<<"o"<<" ";
            }
            else
            {
                cout<<"_"<<" ";
            }
        }
        cout<<endl;
    }
    cout<<endl;
}

void print_ar(int *ar, int n)
{
    for(int i=0;i<n;i++)
    {
        cout<<ar[i]<<" ";
    }
    cout<<endl;
}

void print_vec(vector<int> vec)

```

```

{
    for(int i=0;i<vec.size();i++)
    {
        cout<<vec[i]<<" ";
    }
    cout<<endl;
}
void print_deq(deque<int> &mydeq)
{
    for(int i=0;i<mydeq.size();i++)
    {
        cout<<mydeq[i]<<" ";
    }
    //cout<<mydeq.size();
    //cout<<"mydeq";
    cout<<endl;
}
void print_deq2(deque<int> &mydeq)
{
    for(int i=0;i<mydeq.size();i+=2)
    {
        cout<<mydeq[i]<<" ";
        cout<<mydeq[i+1]<<" ";
    }
    //cout<<mydeq.size();
    //cout<<"mydeq";
    cout<<endl;
}
void print_deq_living(deque<int> &mydeq)
{
    for(int i=0;i<mydeq.size();i+=2)
    {
        cout<<mydeq[i]<<" "<<mydeq[i+1]<<" ";
    }
    //cout<<mydeq.size();
    //cout<<"mydeq";
    cout<<endl;
}
int count_life_kolvo(int* ar, int length)
{
    int kolvo=0;
    for(int i=0;i<length;i++)
    {
        if(ar[i]==1)
        {
            kolvo++;
        }
    }
}

```

```

    return kolvo;
}
void coord_recalc(int direc, int sign)
{
    //direc 1 -> W
    //direc -1 -> N

    //sign > 0 -> exp
    //sign <= 0 -> retraction

    if(direc==1)
    {
        for(int i=0;i<living.size();i+=2)
        {
            if(sign>0)
            {
                living[i+1]+=sign;
            }
            else
            {
                living[i+1]+=sign;
            }
        }
    }
    else
    {
        for(int i=0;i<living.size();i+=2)
        {
            if(sign==1)
            {
                living[i]+=sign;
            }
            else
            {
                living[i]+=sign;
            }
        }
    }

    /*int t;
    if(direc==1)
    {
        for(int i=0;i<row;i++)
        {
            if(sign==1)
            {
                for(int u=col-1;u>0;u--)

```

```

        {
            if(M[i][u]==1)
            {
                t=M[i][u];
                M[i][u]=M[i][u+1];
                M[i][u+1]=t;
            }
        }
    }
else
{
    for(int u=0;u<col;u++)
    {
        if(M[i][u]==1)
        {
            t=M[i][u];
            M[i][u]=M[i][u-1];
            M[i][u-1]=t;
        }
    }
}
}
else
{
    for(int i=0;i<row;i++)
    {
        for(int u=col;u>0;u--)
        {
            if(M[i][u]==1)
            {
                if(sign==1)
                {
                }
            }
        }
    }
}
}*/
}
void coord_recalc_candidates(int direc, int sign)
{
    if(direc==1)
    {
        for(int i=0;i<living_candidats.size();i+=2)
        {
            if(sign>0)
            {

```

```

        living_candidats[i+1]+=sign;
    }
    else
    {
        living_candidats[i+1]+=sign;
    }
}

for(int i=0;i<killing_candidats.size();i+=2)
{
    if(sign>0)
    {
        killing_candidats[i+1]+=sign;
    }
    else
    {
        killing_candidats[i+1]+=sign;
    }
}
}
else
{
    for(int i=0;i<living_candidats.size();i+=2)
    {
        if(sign>0)
        {
            living_candidats[i]+=sign;
        }
        else
        {
            living_candidats[i]+=sign;
        }
    }
}

for(int i=0;i<killing_candidats.size();i+=2)
{
    if(sign>0)
    {
        killing_candidats[i]+=sign;
    }
    else
    {
        killing_candidats[i]+=sign;
    }
}
}
}
}

```

```

int** expansion_S(int **M,int **r, int &row, int &col, int step, int exp_red, int fill_val)
{
    //print_matrix(M,row,col);
    if(exp_red>0)
    {
        M=(int**)realloc(M,(row+step)*sizeof(int*));
        //cout<<"test5.0"<<endl;
        //cout<<"*r="<<*r<<endl;
        //cout<<"row="<<row<<" col="<<col<<" step="<<step<<endl;
        (*r)=(int*)realloc((*r),(row*col+col*step)*sizeof(int));
        //cout<<" *r1="<<*r<<endl;
        //cout<<"test5.1"<<endl;
        for(int i=0;i<row+step;i++)
        {
            M[i]=(*r)+i*col;
        }
        //cout<<"test5.2"<<endl;

        for(int i=row;i<row+step;i++)
        {
            for(int u=0;u<col;u++)
            {
                M[i][u]=fill_val;
            }
        }
        //cout<<"test5.3"<<endl;
        row=row+step;
    }
    else if(exp_red<0 && row-step>=0)
    {
        M=(int**)realloc(M,(row-step)*sizeof(int*));
        (*r)=(int*)realloc((*r),(row*col-col*step)*sizeof(int));

        for(int i=0;i<row-step;i++)
        {
            M[i]=(*r)+i*col;
        }
        row=row-step;
    }
    //print_matrix(M,row,col);
    return M;
}

/*void expansion_SS(int step, int exp_red, int fill_val)
{
    //print_matrix(MM,roww,coll);
    if(exp_red>0)
    {

```

```

MM=(int**)realloc(MM,(roww+step)*sizeof(int*));
//cout<<"test5.0"<<endl;
//cout<<"row="<<roww<<" col="<<coll<<" step="<<step<<endl;
rr=(int*)realloc(rr,(roww*coll+coll*step)*sizeof(int));
//cout<<"test5.1"<<endl;
for(int i=0;i<roww+step;i++)
{
    MM[i]=rr+i*coll;
}
//cout<<"test5.2"<<endl;

for(int i=roww;i<roww+step;i++)
{
    for(int u=0;u<coll;u++)
    {
        MM[i][u]=fill_val;
    }
}
//cout<<"test5.3"<<endl;
roww=roww+step;
}
else if(exp_red<0 && roww-step>=0)
{
    MM=(int**)realloc(MM,(roww-step)*sizeof(int*));
    rr=(int*)realloc(rr,(roww*coll-coll*step)*sizeof(int));

    for(int i=0;i<roww-step;i++)
    {
        MM[i]=rr+i*coll;
    }
    roww=roww-step;
}
//print_matrix(MM,roww,coll);
}*/

int** expansion_N(int **M,int **r, int &row, int &col, int step, int exp_red, int fill_val)
{
    int t;
    if(exp_red>0)
    {
        M=(int**)realloc(M,(row+step)*sizeof(int*));
        (*r)=(int*)realloc((*r),(row*col+col*step)*sizeof(int));

        for(int i=0;i<row+step;i++)
        {
            M[i]=(*r)+i*col;
        }
    }
}

```

```

for(int i=row;i<row+step;i++)
{
    for(int u=0;u<col;u++)
    {
        M[i][u]=fill_val;
    }
}
for(int i=(row+step)*col-col*step;i<(row+step)*col;i++)
{
    for(int u=i;u>0;u--)
    {
        t>(*r)[u];
        (*r)[u]=(*r)[u-1];
        (*r)[u-1]=t;
    }
}
row=row+step;
}
else if(exp_red<0 && row-step>=0)
{
    for(int i=col*step;i>0;i--)
    {
        for(int u=i;u<col*row-1;u++)
        {
            t>(*r)[u];
            (*r)[u]=(*r)[u+1];
            (*r)[u+1]=t;
        }
    }
}

M=(int**)realloc(M,(row-step)*sizeof(int*));
(*r)=(int*)realloc((*r),(row*col-col*step)*sizeof(int));

for(int i=0;i<row-step;i++)
{
    M[i]=(*r)+i*col;
}
row=row-step;
}
//print_matrix(M,row,col);
return M;
}

int** expansion_E(int **M,int **r, int &row, int &col, int step, int exp_red, int fill_val)
{
    int t;
    if(exp_red>0)
    {

```



```

(*r)=(int*)realloc((*r),((col+step)*row)*sizeof(int));

for(int i=0;i<row;i++)
{
    M[i]=(*r)+i*(col+step);
}
for(int i=col*row;i<(col+step)*row;i++)
{
    (*r)[i]=fill_val;
}
for(int i=0;i<row;i++)
{
    for(int u=0;u<step;u++)
    {
        for(int j=col*row+u+i*step;j>(col+step)*(i+1)-step+u;j--)
        {
            t=(*r)[j];
            (*r)[j]=(*r)[j-1];
            (*r)[j-1]=t;
        }
    }
}
col=col+step;
}
else if(exp_red<0 && col-step>=0)
{
    for(int i=row-1;i>0;i--)
    {
        for(int u=0;u<step;u++)
        {
            for(int j=i*col-1-u;j<row*col-1;j++)
            {
                t=(*r)[j];
                (*r)[j]=(*r)[j+1];
                (*r)[j+1]=t;
            }
            //print_matrix(M, row, col);
        }
    }
}
(*r)=(int*)realloc((*r),((col-step)*row)*sizeof(int));
for(int i=0;i<row;i++)
{
    M[i]=(*r)+i*(col-step);
}
col=col-step;
}
return M;
}

```

```

int** expansion_W(int **M,int **r, int &row, int &col, int step, int exp_red, int fill_val)
{
    int t;
    if(exp_red>0)
    {
        (*r)=(int*)realloc((*r),((col+step)*row)*sizeof(int));

        for(int i=0;i<row;i++)
        {
            M[i]=(*r)+i*(col+step);
        }
        for(int i=col*row;i<(col+step)*row;i++)
        {
            (*r)[i]=fill_val;
        }
        for(int i=0;i<row;i++)
        {
            for(int u=0;u<step;u++)
            {
                for(int j=col*row+u+i*step;j>(col+step)*(i)+u;j--)
                {
                    t=(*r)[j];
                    (*r)[j]=(*r)[j-1];
                    (*r)[j-1]=t;
                }
            }
        }
        col=col+step;
    }
    else if(exp_red<0 && col-step>=0)
    {
        for(int i=row;i>0;i--)
        {
            for(int u=0;u<step;u++)
            {
                for(int j=i*col-col+u;j<row*col-1;j++)
                {
                    t=(*r)[j];
                    (*r)[j]=(*r)[j+1];
                    (*r)[j+1]=t;
                }
            }
            //print_matrix(M, row, col);
        }
    }
    (*r)=(int*)realloc((*r),((col-step)*row)*sizeof(int));
    for(int i=0;i<row;i++)
    {

```

```

        M[i]=(*r)+i*(col-step);
    }
    col=col-step;
}
return M;
}

```

```

void fill_matrix(int** M, int row, int col, int* ar)
{
    int j=0,flag;
    int first,second;
    for(int i=0;i<row;i++)
    {
        M[i][0]=0;
        M[0][i]=0;
        M[i][row-1]=0;
        M[row-1][i]=0;
    }
    for(int i=1;i<row-1;i++)
    {
        for(int u=1;u<col-1;u++)
        {
            M[i][u]=ar[j];
            if(ar[j]==1)
            {
                //living.push_back(i);
                //living.push_back(u);
                for(int x=i-1;x<=i+1;x++)
                {
                    for(int y=u-1;y<=u+1;y++)
                    {
                        // need to push_back unique x and y
                        flag=0;
                        //cout<<"x="<<x<<" y="<<y<<endl;
                        for(int k=0;k<living.size();k+=2)
                        {
                            first=living[k];
                            second=living[k+1];
                            if(first==x && second==y)
                            {
                                flag=1;
                                break;
                            }
                        }
                    }
                }
                if(flag==0)
                {
                    living.push_back(x);
                    living.push_back(y);
                }
            }
        }
    }
}

```

```

        //print_deq(living);
    }
}
}
}
j++;
}
}
//print_deq(living);
}

```

```

void life(int &life_kolvo)
{
    int first,second,alive_sosed,flag;
    int first_new,second_new;
    //deque<int> new_life;
    //deque<int> old_life;
    //print_deq_living(living);
    //cout<<"test1"<<endl;
    cout<<endl<<endl;
    for(int k=0;k<living.size();k+=2)
    {
        first=living[k];
        second=living[k+1];
        alive_sosed=0;
        for(int i=first-1;i<=first+1;i++)
        {
            for(int u=second-1;u<=second+1;u++)
            {
                //i==first && u==second
                if(i!=first || u!=second)
                {
                    //M[i][u] does not exist = dead
                    if(i<0 || i>=roww || u<0 || u>=coll)
                    {

                    }

                    //M[i][u] exists but dead
                    else if(MM[i][u]==0)
                    {

                    }

                    //M[i][u] exists and alive
                    else if(MM[i][u]==1)
                    {
                        alive_sosed++;
                    }
                }
            }
        }
    }
}

```

```

    }
}

//cout<<"test2 "<<k<<endl;
//creation of life
//cout<<"cell = "<<first<<" "<<second<<" MM="<<MM[first][second]<<" sosed =
"<<alive_sosed<<endl;
//cout<<"test2 "<<k<<endl;
if(alive_sosed==3 && MM[first][second]==0)
{
    new_life.push_back(first);
    new_life.push_back(second);
}
else if(MM[first][second]==1 && (alive_sosed>3 || alive_sosed<2))
{
    old_life.push_back(first);
    old_life.push_back(second);
}
}
//cout<<"test3"<<endl;
//correcting living
//cout<<"living"<<endl;
//print_deq(living);

////cout<<"new_life"<<endl;
////print_deq2(new_life);

////cout<<"living"<<endl;
////print_deq2(living);

for(int k=0;k<new_life.size();k+=2)
{
    first=new_life[k];
    second=new_life[k+1];
    MM[first][second]=1;

    for(int x=first-1;x<=first+1;x++)
    {
        for(int y=second-1;y<=second+1;y++)
        {
            flag=0;
            for(int k=0;k<living.size();k+=2)
            {
                first_new=living[k];
                second_new=living[k+1];
                if(first_new==x && second_new==y)
                {
                    flag=1;
                }
            }
        }
    }
}

```

```

        break;
    }
}

if(flag==0)
{
    flag=0;
    for(int k=0;k<living_candidats.size();k+=2)
    {
        first_new=living_candidats[k];
        second_new=living_candidats[k+1];
        if(first_new==x && second_new==y)
        {
            flag=1;
            break;
        }
    }

    if(flag==0)
    {
        living_candidats.push_back(x);
        living_candidats.push_back(y);
    }
}
}
}
}
}

```

```

///cout<<"living_candidats medium"<<endl;
///print_deq2(living_candidats);

```

```

for(int k=0;k<old_life.size();k+=2)
{
    first=old_life[k];
    second=old_life[k+1];
    MM[first][second]=0;
}

```

```

//EXPANSION

```

```

int max_x,max_y,min_x,min_y;
int flag1=0;
for(int i=0;i<roww && flag1==0;i++)
{
    for(int u=0;u<coll && flag1==0;u++)
    {
        if(MM[i][u]==1 && flag1==0)
        {
            min_x=i;

```

```

        flag1=1;
    }
}
flag1=0;
for(int i=roww-1;i>=0 && flag1==0;i--)
{
    for(int u=coll-1;u>=0 && flag1==0;u--)
    {
        if(MM[i][u]==1 && flag1==0)
        {
            max_x=i;
            flag1=1;
        }
    }
}
flag1=0;
for(int i=coll-1;i>=0 && flag1==0;i--)
{
    for(int u=roww-1;u>=0 && flag1==0;u--)
    {
        if(MM[u][i]==1 && flag1==0)
        {
            max_y=i;
            flag1=1;
        }
    }
}
flag1=0;
for(int i=0;i<coll && flag1==0;i++)
{
    for(int u=0;u<roww && flag1==0;u++)
    {
        if(MM[u][i]==1 && flag1==0)
        {
            min_y=i;
            flag1=1;
        }
    }
}

////cout<<"print_matrix before EXPAND"<<endl;
////print_matrix(MM,roww,coll);

//-----
//cout<<"-----"<<endl;
//print_deq_living(living);
//cout<<"-----"<<endl;

```

```

        //cout<<"max_x="<<max_x<<" max_y="<<max_y<<" min_x="<<min_x<<"
min_y="<<min_y<<" roww="<<roww<<" coll="<<coll<<endl;
        if(flag1==1)
        {
            //cout<<"()()()()()coll="<<coll<<" roww="<<roww<<" max_x="<<max_x<<"
min_x="<<min_x<<" max_y="<<max_y<<" min_y="<<min_y<<endl;
            //expand M to new life
            if(max_y==coll-1)
            {
                MM=expansion_E(MM,&rr,roww,coll,1,1,0);
                life_exp.push_back(1);
            }
            else
            {
                life_exp.push_back(0);
            }
            //cout<<"print_matrix after EXPAND1"<<endl;
            //print_matrix(MM,roww,coll);

            if(min_y==0)
            {
                MM=expansion_W(MM,&rr,roww,coll,1,1,0);
                coord_recalc(1,1);
                coord_recalc_candidates(1,1);
                max_y++;
                min_y++;
                life_exp.push_back(1);
            }
            else
            {
                life_exp.push_back(0);
            }
            //cout<<"print_matrix after EXPAND2"<<endl;
            //print_matrix(MM,roww,coll);

            if(max_x==roww-1)
            {
                //cout<<"-----coll="<<coll<<" roww="<<roww<<"
max_x="<<max_x<<endl;
                MM=expansion_S(MM,&rr,roww,coll,1,1,0);
                life_exp.push_back(1);
            }
            else
            {
                life_exp.push_back(0);
            }
            //cout<<"print_matrix after EXPAND3"<<endl;

```



```

//print_matrix(MM,roww,coll);

if(min_x==0)
{
    //cout<<"++++++++++++coll="<<coll<<" roww="<<roww<<"
max_x="<<max_x<<endl;
    MM=expansion_N(MM,&rr,roww,coll,1,1,0);
    coord_recalc(-1,1);
    coord_recalc_candidates(-1,1);
    max_x++;
    min_x++;
    life_exp.push_back(1);
}
else
{
    life_exp.push_back(0);
}
//cout<<"print_matrix after EXPAND"<<endl;
//print_matrix(MM,roww,coll);
//-----
//compression of M with empty columns and rows on edges
if(max_y<=coll-3)
{
    //cout<<"test7"<<endl;
    MM=expansion_E(MM,&rr,roww,coll,coll-3-max_y+1,-1,0);
    life_exp.push_back((coll-3-max_y+1)*(-1));
}
else
{
    life_exp.push_back(0);
}
if(min_y>=2)
{
    MM=expansion_W(MM,&rr,roww,coll,min_y-2+1,-1,0);
    //coord_recalc(1,-1);
    //coord_recalc_candidates(1,-1);
    coord_recalc(1,-(min_y-2+1));
    coord_recalc_candidates(1,-(min_y-2+1));
    max_y--;
    min_y--;
    life_exp.push_back((min_y-2+1)*(-1));
}
else
{
    life_exp.push_back(0);
}
if(max_x<=roww-3)
{

```

```

        //cout<<"coll="<<coll<<" roww="<<roww<<" max_x="<<max_x<<endl;
        MM=expansion_S(MM,&rr,roww,coll,roww-3-max_x+1,-1,0);
        life_exp.push_back((roww-3-max_x+1)*(-1));
    }
    else
    {
        life_exp.push_back(0);
    }
    if(min_x>=2)
    {
        MM=expansion_N(MM,&rr,roww,coll,min_x-2+1,-1,0);
        //coord_recalc(-1,-1);
        //coord_recalc_candidates(-1,-1);
        coord_recalc(-1,-(min_x-2+1));
        coord_recalc_candidates(-1,-(min_x-2+1));
        max_x--;
        min_x--;
        life_exp.push_back((min_x-2+1)*(-1));
    }
    else
    {
        life_exp.push_back(0);
    }
}

///cout<<"print_matrix after EXPAND"<<endl;
///print_matrix(MM,roww,coll);

//living_candidats should be here
///cout<<"living last1"<<endl;
///print_deq2(living);

///cout<<"living_candidats last"<<endl;
///print_deq2(living_candidats);

for(int i=0;i<living_candidats.size();i+=2)
{
    living.push_back(living_candidats[i]);
    living.push_back(living_candidats[i+1]);
}

///cout<<"living last2"<<endl;
///print_deq2(living);

//OUT RANGE cleaning - NESSESARY
//because "main array" cutting on the top, but "living array" can cut only here

```



```

        //M[i][u] exists and alive
        else if(MM[i][u]==1)
        {
            alive_sosed++;
        }
    }
}

if(MM[first][second]==0 && alive_sosed==0)
{
    //cout<<"ki_ca"<<endl;
    //print_deq2(killing_candidats);
    killing_candidats.push_back(k);
}
}

///cout<<"killing_candidats"<<endl;
///print_deq2(killing_candidats);

for(int i=killing_candidats.size()-1;i>=0;i--)
{
    //living.erase(living.begin()+killing_candidats[i]);
    //living.erase(living.begin()+killing_candidats[i]);
}

//cout<<"living_after"<<endl;
//print_deq2(living);

//cout<<"++++++++++"<<endl;
//print_deq_living(living);

//OUT RANGE cleaning - NESSESARY
//because "main array" cutting on the top, but "living array" can cut only here
/*
    cout<<"-----OUT RANGE cleaning---roww="<<roww<<"
coll="<<coll<<"-----"<<endl;
    for(int i=0;i<living.size();i+=2)
    {
        first=living[i];
        second=living[i+1];
        if(first<0 || second<0 || first>=roww || second>=coll)
        {
            cout<<"-----OUT RANGE cleaning---first="<<first<<"
second="<<second<<"-----"<<endl;

            living.erase(living.begin()+i);
            living.erase(living.begin()+i);

```

```

        i-=2;
    }
}
*/

//cout<<"++++++"<<endl;
//print_deq_living(living);

///cout<<"living_recalc"<<endl;
///print_deq2(living);

cout<<"new="<<new_life.size()/2<<" old="<<old_life.size()/2<<"
life_kolvo="<<life_kolvo<<endl;
//print_deq(old_life);
life_kolvo+=new_life.size()/2-old_life.size()/2;
}
void create_memory(int n)
{
    free(MM);
    free(rr);
    roww=sqrt(n)+2;
    coll=sqrt(n)+2;
    MM=(int**)calloc(roww,sizeof(int*));
    rr=(int*)calloc(roww*coll,sizeof(int));
    //MM=new int*[roww];
    //rr=new int[roww*coll];
    //cout<<"rr="<<rr<<endl;
    for(int i=0;i<roww;i++)
    {
        MM[i]=rr+i*coll;
    }
    for(int i=0;i<roww;i++)
    {
        for(int u=0;u<coll;u++)
        {
            MM[i][u]=0;
        }
    }
}
int life_proverka()
{
    int stop=0;
    vector<int> simptom_vec;
    simptom_vec.insert(simptom_vec.end(), {0,0});
    // 0- in progress
    // 1- found
    // -1- not found

```

```

//10 consecutive
    int counter=1;
//period
    int temp=life_pop[life_pop.size()-1];
    int flag;
    int flag_dif=-1;
    vector<int> hvost;

for(int i=life_pop.size()-1;i>=0;i--)
{
    //10 consecutive
    if(counter==10)
    {
        symptom_vec[0]=1;
    }
    if(life_pop[i]==life_pop[i-1] && symptom_vec[0]==0)
    {
        counter++;
    }
    else if(symptom_vec[0]!=1)
    {
        symptom_vec[0]=-1;
    }
//-----
//period
    if(i!=life_pop.size()-1 && life_pop[i]==temp && symptom_vec[1]==0)
    {
        //cout<<"life_pop"<<endl;
        //print_vec(life_pop);
        //cout<<"hvost"<<endl;
        //print_vec(hvost);
        //cout<<"life_pop[i]= "<<life_pop[i-(life_pop.size()-2-i)]<<endl;
        flag=0;
        //cout<<"----- hvost[...]"<<endl;
        for(int u=i-(life_pop.size()-2-i);u<i;u++)
        {
            //cout<<hvost[hvost.size()-(u-(i-(life_pop.size()-2-i)))-1]<<" ";
            if(life_pop[u]!=hvost[hvost.size()-(u-(i-(life_pop.size()-2-i)))-1])
            {
                hvost.push_back(life_pop[i]);
                if(hvost.size()>life_pop.size()/2)
                {
                    symptom_vec[1]=-1;
                }
            }
            flag=1;
            break;
        }
    }
}

```

```

        //cout<<endl<<"-----"<<endl;
        if(flag==0 && flag_dif==-2)
        {
            simptom_vec[1]=1;
        }
    }
    else
    {
        if(flag_dif==-1)
        {
            flag_dif=life_pop[i];
        }
        else if(flag_dif!=life_pop[i])
        {
            flag_dif=-2;
        }
        hvost.push_back(life_pop[i]);
    }
    //-----
    //big
}
for(int i=0;i<simptom_vec.size();i++)
{
    if(simptom_vec[i]==1)
    {
        stop=1;
        break;
    }
}
return stop;
}

void ar_create(int n)
{
    create_memory(n);

    //-----
    int life_kolvo;
    int* ar=(int*)calloc(n,sizeof(int));
    int ar_temp[9]={0, 0, 1, 0, 1, 0, 1, 1, 1};
    //int* ar=new int(n);
    int flag=0;
    for(int i=0;i<n;i++)
    {
        ar[i]=0;
    }
    int counter=0;
}

```

```

int w;
while(flag==0)
{
    flag=1;
    print_ar(ar,n);
    living.clear();

    cout<<"-----"<<endl;
    cout<<"----"<<counter<<"----"<<endl;
    //if(counter>0 && counter<5)
    //{
    fill_matrix(MM,roww,coll,ar);
    print_matrix(MM,roww,coll);
    life_kolvo=count_life_kolvo(ar,n);
    w=0;
    int stop;
    while(w<30)
    {
        life(life_kolvo);
        cout<<"phase"<<endl;
        print_matrix(MM,roww,coll);
        life_pop.push_back(life_kolvo);
        new_life.clear();
        old_life.clear();
        living_candidats.clear();
        killing_candidats.clear();
        cout<<"w="<<w<<" life_kolvo="<<life_kolvo<<endl;
        if(life_kolvo<=0)
        {
            break;
        }
        stop=life_proverka();
        if(stop==1)
        {
            break;
        }
        w++;
    }
    print_vec(life_exp);
    print_vec(life_pop);
    cout<<"stop="<<stop<<" length="<<life_pop.size()<<endl;
    cout<<"final"<<endl;
    print_matrix(MM,roww,coll);
    life_exp.clear();
    life_pop.clear();
    create_memory(n);
    //}
    for(int i=n-1;i>=0;i--)

```



```

    {
        if(ar[i]==0)
        {
            ar[i]=1;
            for(int u=i+1;u<n;u++)
            {
                ar[u]=0;
            }
            flag=0;
            break;
        }
    }

    counter++;
}
}

```

```

void experiment_expen()

```

```

{
    /*row=3;
    col=5;
    MM=new int*[row];
    rr=new int[row*col];
    for(int i=0;i<row;i++)
    {
        MM[i]=rr+i*col;
    }
    for(int i=0;i<row;i++)
    {
        for(int u=0;u<col;u++)
        {
            MM[i][u]=0;
        }
    }
    */

    //-----

    /*M=expansion_S(M,&r,row,col,6,1,4);
    print_matrix(M,row,col);
    M=expansion_S(M,&r,row,col,2,1,5);
    print_matrix(M,row,col);
    M=expansion_S(M,&r,row,col,2,1,9);
    print_matrix(M,row,col);*/

    //M=expansion_N(M,&r,row,col,4,1,7);
    //print_matrix(M,row,col);
    //M=expansion_N(M,&r,row,col,2,-1,7);
    //print_matrix(M,row,col);

```

```

/*M=expansion_E(M,&r,row,col,3,1,1);
print_matrix(M,row,col);
M=expansion_E(M,&r,row,col,2,1,2);
print_matrix(M,row,col);
M=expansion_E(M,&r,row,col,2,1,3);
print_matrix(M,row,col);
M=expansion_E(M,&r,row,col,3,-1,1);
print_matrix(M,row,col);*/

```

```

/*MM=expansion_W(MM,&rr,row,col,6,1,6);
print_matrix(MM,row,col);

```

```

MM=expansion_W(MM,&rr,row,col,3,-1,4);
print_matrix(MM,row,col);*/

```

```

//-----

```

```

/*
//life_pop.push_back();
//life_pop.insert(life_pop.end(), {19,19,49,59,29,39,49,59,59,29,39,49,59});
//life_pop.insert(life_pop.end(), {19,19,49,59,29,39,59,29,39,49,59});
//life_pop.insert(life_pop.end(), {19,19,49,59,29,39,49,59,59,29,39,49,59});
//life_pop.insert(life_pop.end(), {19,0,0,0,0,0,0,0,0,1,1});
print_vec(life_pop);
//{1,1,4,5,2,3,4,5,5,2,3,4,5};
int stop=life_proverka();
cout<<endl<<stop<<endl;
*/

```

```

//-----

```

```

int n=9;

```

```

create_memory(n);
int life_kolvo;
int ar_temp[9]={1, 1, 1, 1, 1, 1, 1, 0, 1};
//int* ar=new int(n);
int counter=0;
int w;

```

```

print_ar(ar_temp,n);
living.clear();
fill_matrix(MM,roww,coll,ar_temp);
print_matrix(MM,roww,coll);
life_kolvo=count_life_kolvo(ar_temp,n);
w=0;

```

```

int stop;
while(w<30)
{
    life(life_kolvo);
    cout<<"phase"<<endl;
    print_matrix(MM,roww,coll);
    life_pop.push_back(life_kolvo);
    new_life.clear();
    old_life.clear();
    living_candidats.clear();
    killing_candidats.clear();
    cout<<"w="<<w<<" life_kolvo="<<life_kolvo<<endl;
    if(life_kolvo<=0)
    {
        break;
    }
    stop=life_proverka();
    if(stop==1)
    {
        break;
    }
    w++;
}
print_vec(life_exp);
print_vec(life_pop);
cout<<"stop="<<stop<<" length="<<life_pop.size()<<endl;
cout<<"final"<<endl;
print_matrix(MM,roww,coll);
life_exp.clear();
life_pop.clear();
}
//0 0 1 0 1 0 1 1 1
int main()
{
    //life();
    ar_create(9);
    //experiment_expen();
    return 0;
}

```