

main

```
#include <iostream>
#include <cmath>
#include <map>
#include <cstdlib>
#include <ctime>
#include <deque>
#include <vector>
#include <string>
#include <fstream>
#include <map>
#include <string>
#include "/usr/include/mariadb/mysql.h"
#include "sha2.h"

#define EXPERIMENT_SIZE 9

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;
vector<string> life_hash;
vector<string> bank_hash;
vector<string> spaceship_bank_hash;
int life_pop_max;
int roww,coll;
int roww_red,coll_red;
int **MM;
int *rr;
int **M_reduction;
int *r_reduction;
int print_counter = 1;
int enum_counter; //enumeration counter of games

//wait till period has not arrived
int WAIT = 1000;

//hash constant (number of periods of game) how many times a period has repeated
int HASH = 50;

//flight constant (how many times a period has repeated for a spaceship)
int FLIGHT=10;

//initial hash sector !!!!!!!!!(SECTOR >= SUSPICIOUS_SECTOR)!!!!!!!!!!!!!! depreciated
int SECTOR = 40;

//number of initial steps of game life that go through bank_proverka
int SUSPICIOUS_SECTOR = 10;
```

```
//determines number for vicinity in search for spaceships
int VICINITY=15;
```

```
struct connection_details
{
    const char *server, *user, *password, *database;
};
```

```
MYSQL* mysql_connection_setup(struct connection_details mysql_details){
    MYSQL *connection = mysql_init(NULL); // mysql instance
```

```
    //connect database
    if(!mysql_real_connect(connection, mysql_details.server, mysql_details.user,
mysql_details.password, mysql_details.database, 0, NULL, 0)){
        std::cout << "Connection Error: " << mysql_error(connection) << std::endl;
        exit(1);
    }
```

```
    return connection;
}
```

```
// mysql_res = mysql result
```

```
MYSQL_RES* mysql_perform_query(MYSQL *connection, const char *sql_query){
    //send query to db
    if(mysql_query(connection, sql_query)){
        std::cout << "MySQL Query Error: " << mysql_error(connection) << std::endl;
        exit(1);
    }
```

```
    return mysql_use_result(connection);
}
```

```
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_matrix_test(int **M, int row, int col)
```

```

{
    for(int i=0;i<row;i++)
    {
        for(int u=0;u<col;u++)
        {
            //cout<<M[i][u]<<" ";
            printf("%5d ",M[i][u]);

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

void print_map_string_int(map<string,int> mymap)
{
    map<string,int>::iterator it;
    for(auto it = mymap.cbegin(); it != mymap.cend(); ++it)
    {
        cout<<it->first<<" "<<it->second<<endl;
    }
    cout<<endl;
}

void print_vec(vector<int> vec)
{
    for(int i=0;i<vec.size();i++)
    {
        cout<<vec[i]<<" ";
    }
    cout<<endl;
}

void print_vec_string(vector<string> vec)
{
    for(int i=0;i<vec.size();i++)
    {
        cout<<vec[i]<<endl;
    }
}

```

```

    }
    cout<<endl;
}

void print_vec_4(vector<int> vec)
{
    for(int i=0;i<vec.size();i++)
    {
        cout<<vec[i]<<" ";
        if(i%4==3)
        {
            cout<<" ";
        }
    }
    cout<<endl;
}

void print_deq(deque<int> &mydeq)
{
    for(int i=0;i<mydeq.size();i++)
    {
        cout<<mydeq[i]<<" ";
    }
    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<<endl;
}

void print_deq_living(deque<int> &mydeq)
{
    for(int i=0;i<mydeq.size();i+=2)
    {
        cout<<mydeq[i]<<" "<<mydeq[i+1]<<" ";
    }
    cout<<endl;
}

void print_init_matrix(int* ar,int n)
{
    int row=sqrt(n);
    int col=sqrt(n);
    for(int j=0;j<n;j++)
    {
        if(ar[j]==1)
        {
            cout<<"o ";
        }
    }
}

```

```

        else
        {
            cout<<"_ ";
        }
        if((j+1)%row==0)
        {
            cout<<endl;
        }
    }
    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;
}

```

```

int count_life_kolvo_MM()
{
    int kolvo=0;
    for(int i=0;i<roww;i++)
    {
        for(int u=0;u<coll;u++)
        {
            if(MM[i][u]==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;
        }
    }
}
}

```

```

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;
        }
    }
}

void copy_matrix()
{
    free(M_reduction);
    free(r_reduction);

    roww_red=roww;
    coll_red=coll;

    M_reduction=(int**)calloc((roww_red),sizeof(int*));
    r_reduction=(int*)calloc(((roww_red)*(coll_red)),sizeof(int));

    for(int i=0;i<roww_red;i++)
    {
        M_reduction[i]=r_reduction+i*coll_red;
    }
    for(int i=0;i<roww_red;i++)
    {
        for(int u=0;u<coll_red;u++)
        {
            M_reduction[i][u]=MM[i][u];
        }
    }
}

int** expansion_S(int **M,int **r, int &row, int &col, int step, int exp_red, int fill_val)
{
    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;
        }
    }
    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;
}
return M;
}

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 i=col*step-1;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;
}
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;
                }
            }
        }
        (*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;j<row*col-1;j++)
            {
                t=(*r)[j];
                (*r)[j]=(*r)[j+1];
                (*r)[j+1]=t;
            }
        }
    }
    (*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 pair_compare(int first_x, int first_y, int second_x, int second_y)
{
    if(first_x<second_x)
    {
        return -1;
    }
    else if(first_x==second_x && first_y<second_y)
    {
        return -1;
    }
    else if(first_x==second_x && first_y==second_y)
    {
        return 0;
    }
    else
    {
        return 1;
    }
}

void insert_binary_search(int x, int y, deque<int> &liv)
{
    liv.push_back(x);
    liv.push_back(y);
    int temp;
    for(int i=liv.size()-1;i>2;i-=2)
    {
        if(pair_compare(liv[i-1],liv[i],liv[i-3],liv[i-2])==-1)
        {
            temp=liv[i-1];
            liv[i-1]=liv[i-3];
            liv[i-3]=temp;
        }
    }
}

```

```

        temp=liv[i];
        liv[i]=liv[i-2];
        liv[i-2]=temp;
    }
    else
    {
        break;
    }
}
}
int select_binary_search(int x, int y, deque<int> &liv)
{
    int middle;
    int start=0,finish=liv.size()-2;
    int result=0;
    //[(2+8)-((2+8)/2)%2]/2
    if(liv.size(>0)
    {
        while(finish-start>2)
        {
            middle=((start+finish)-((start+finish)/2)%2)/2;
            if(pair_compare(x,y,liv[middle],liv[middle+1])==-1)
            {
                finish=middle;
            }
            else if(pair_compare(x,y,liv[middle],liv[middle+1])==1)
            {
                start=middle;
            }
            else if(pair_compare(x,y,liv[middle],liv[middle+1])==0)
            {
                result=1;
                break;
            }
        }
        if(result==0 && (pair_compare(x,y,liv[start],liv[start+1])==0 ||
pair_compare(x,y,liv[finish],liv[finish+1])==0))
        {
            result=1;
        }
    }

    return result;
}

```

```

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)
            {
                for(int x=i-1;x<=i+1;x++)
                {
                    for(int y=u-1;y<=u+1;y++)
                    {
                        //BINARY SELECT
                        flag=0;
                        if(select_binary_search(x,y,living)==1)
                        {
                            flag=1;
                        }
                        //BINARY INSERT
                        if(flag==0)
                        {
                            insert_binary_search(x,y,living);
                        }
                    }
                }
            }
            j++;
        }
    }
}

```

```

void file_print(int* ar, int n, vector<int> simptom_vec)
{
    int row=sqrt(n);
    int col=sqrt(n);
    FILE* fp=fopen("Interesting_combinations.txt","a");
    fprintf(fp,"print_counter= %d\n",print_counter);
    print_counter++;
    for(int i=0;i<n;i++)
    {
        fprintf(fp,"%d ",ar[i]);
    }
    fprintf(fp,"\n");
    for(int i=0;i<simptom_vec.size();i++)
    {
        fprintf(fp,"%d ",simptom_vec[i]);
    }
    fprintf(fp,"\n");
    fprintf(fp,"max life= %d",life_pop_max);
    fprintf(fp,"\n");
    fprintf(fp,"life_pop\n");
    for(int i=0;i<life_pop.size();i++)
    {

```

```

        fprintf(fp,"%d ",life_pop[i]);
    }
    fprintf(fp,"\n");
    fprintf(fp,"life_exp\n");
    for(int i=0;i<life_exp.size();i++)
    {
        fprintf(fp,"%d ",life_exp[i]);
    }
    fprintf(fp,"\n");
    for(int j=0;j<n;j++)
    {
        if(ar[j]==1)
        {
            fprintf(fp,"o ");
        }
        else
        {
            fprintf(fp,"_ ");
        }
        if((j+1)%row==0)
        {
            fprintf(fp,"\n");
        }
    }
    fprintf(fp,"\n");
    fclose(fp);
}

void file_print_hash(int* ar, int n, int game_counter,int test)
{
    int row=sqrt(n);
    int col=sqrt(n);
    FILE* fp=fopen("Interesting_combinations.txt","a");
    //fprintf(fp,"file_print_hash\n");
    fprintf(fp,"print_counter= %d\n",print_counter);
    print_counter++;
    fprintf(fp,"game_counter= %d\n",game_counter);

    // if(test!=1)
    // {
    //     for(int i=0;i<n;i++)
    //     {
    //         fprintf(fp,"%d ",ar[i]);
    //     }
    // }
    // fprintf(fp,"\n");
    fprintf(fp,"max life= %d",life_pop_max);
    fprintf(fp,"\n");
    // fprintf(fp,"life_pop\n");
    // for(int i=0;i<life_pop.size();i++)
    // {
    //     fprintf(fp,"%d ",life_pop[i]);
    // }
    // fprintf(fp,"\n");
}

```

```

//fprintf(fp,"life_exp\n");
//for(int i=0;i<life_exp.size();i++)
//{
//  fprintf(fp,"%d ",life_exp[i]);
//}
//fprintf(fp,"\n");
if(test!=1)
{
  for(int j=0;j<n;j++)
  {
    if(ar[j]==1)
    {
      fprintf(fp,"o ");
    }
    else
    {
      fprintf(fp,"_ ");
    }
    if((j+1)%row==0)
    {
      fprintf(fp,"\n");
    }
  }
  fprintf(fp,"\n");
}
fclose(fp);
}

void file_print_vec(vector<int> index_mass,char mystr[])
{
  FILE* fp=fopen("Interesting_combinations.txt","a");
  fprintf(fp,"%s\n",mystr);
  for(int i=0;i<index_mass.size();i++)
  {
    fprintf(fp,"%d ",index_mass[i]);
  }
  fprintf(fp,"\n ----- \n");
  fclose(fp);
}

void file_print_num(int num,char mystr[])
{
  FILE* fp=fopen("Interesting_combinations.txt","a");
  fprintf(fp,"%s= %d\n",mystr,num);
  fclose(fp);
}

void file_print_phase(int counter)
{
  FILE* fp=fopen("Interesting_combination_phase.txt","a");
  fprintf(fp,"%d phase \n",counter);
  for(int i=0;i<roww;i++)

```

```

{
  for(int u=0;u<coll;u++)
  {
    if(MM[i][u]==1)
    {
      fprintf(fp,"o ");
    }
    else
    {
      fprintf(fp,"_ ");
    }
  }
  fprintf(fp,"\n");
}
fprintf(fp,"\n");
fclose(fp);
}

```

string create_hash_from_array()

```

{
  string str="";
  for(int i=0;i<roww;i++)
  {
    for(int u=0;u<coll;u++)
    {
      if(MM[i][u]==1)
      {
        str+="1";
      }
      else
      {
        str+="0";
      }
    }
    str+="2";
  }
  string output = sha512(str);
  return output;
}

```

string create_hash_from_array_reduction()

```

{
  string str="";
  for(int i=0;i<roww_red;i++)
  {
    for(int u=0;u<coll_red;u++)
    {
      if(M_reduction[i][u]==1)
      {
        str+="1";
      }
      else
      {
        str+="0";
      }
    }
  }
}

```



```

    }
    str+="2";
}
string output = sha512(str);
return output;
}

```

```
void rotate_counterclockwise()
```

```

{
    int temp;
    for(int i=0; i<roww/2; i++)
    {
        for(int j=i; j<roww-1-i; j++)
        {
            temp=MM[i][j];
            MM[i][j]=MM[j][roww-1-i];
            MM[j][roww-1-i]=MM[roww-1-i][roww-1-j];
            MM[roww-1-i][roww-1-j]=MM[roww-1-j][i];
            MM[roww-1-j][i]=temp;
        }
    }
}

```

```
string virtual_rotateMN_clockwise(int case_num)
```

```

{
    string str="";
    if(case_num==0)
    {
        for(int i=0; i<roww; i++)
        {
            for(int u=0; u<coll; u++)
            {
                if(MM[i][u]==1)
                {
                    str+="1";
                    //cout<<"o ";
                }
                else
                {
                    str+="0";
                    //cout<<"_ ";
                }
            }
            str+="2";
            //cout<<endl;
        }
        //cout<<endl;
    }
    else if(case_num==1)
    {
        for(int i=coll-1; i>=0; i--)
        {
            for(int u=0; u<roww; u++)
            {
                if(MM[u][i]==1)

```

```

        {
            str+="1";
            //cout<<"o ";
        }
        else
        {
            str+="0";
            //cout<<"_ ";
        }
    }
    str+="2";
    //cout<<endl;
}
//cout<<endl;
}
else if(case_num==2)
{
    for(int i=roww-1;i>=0;i--)
    {
        for(int u=coll-1;u>=0;u--)
        {
            if(MM[i][u]==1)
            {
                str+="1";
                //cout<<"o ";
            }
            else
            {
                str+="0";
                //cout<<"_ ";
            }
        }
        str+="2";
        //cout<<endl;
    }
    //cout<<endl;
}
else if(case_num==3)
{
    for(int i=0;i<coll;i++)
    {
        for(int u=roww-1;u>=0;u--)
        {
            if(MM[u][i]==1)
            {
                str+="1";
                //cout<<"o ";
            }
            else
            {
                str+="0";
                //cout<<"_ ";
            }
        }
    }
    str+="2";
}

```

```

        //cout<<endl;
    }
    //cout<<endl;
}

string output = sha512(str);
return output;
}

string virtual_rotateMN_clockwise_red(int case_num)
{
    string str="";
    cout<<"---virtual_rotateMN_clockwise_red---"<<endl;
    if(case_num==0)
    {
        for(int i=0;i<roww_red;i++)
        {
            for(int u=0;u<coll_red;u++)
            {
                if(M_reduction[i][u]==1)
                {
                    str+="1";
                    cout<<"o ";
                }
                else
                {
                    str+="0";
                    cout<<"_ ";
                }
            }
            str+="2";
            cout<<endl;
        }
        cout<<endl;
    }
    else if(case_num==1)
    {
        for(int i=coll_red-1;i>=0;i--)
        {
            for(int u=0;u<roww_red;u++)
            {
                if(M_reduction[u][i]==1)
                {
                    str+="1";
                    cout<<"o ";
                }
                else
                {
                    str+="0";
                    cout<<"_ ";
                }
            }
            str+="2";
        }
    }
}

```

```

        cout<<endl;
    }
    cout<<endl;
}
else if(case_num==2)
{
    for(int i=roww_red-1;i>=0;i--)
    {
        for(int u=coll_red-1;u>=0;u--)
        {
            if(M_reduction[i][u]==1)
            {
                str+="1";
                cout<<"o ";
            }
            else
            {
                str+="0";
                cout<<"_ ";
            }
        }
        str+="2";
        cout<<endl;
    }
    cout<<endl;
}
else if(case_num==3)
{
    for(int i=0;i<coll_red;i++)
    {
        for(int u=roww_red-1;u>=0;u--)
        {
            if(M_reduction[u][i]==1)
            {
                str+="1";
                cout<<"o ";
            }
            else
            {
                str+="0";
                cout<<"_ ";
            }
        }
        str+="2";
        cout<<endl;
    }
    cout<<endl;
}
cout<<"---vrcr finish---"<<endl;

string output = sha512(str);
return output;
}

```

```

void axial_symmetry()
{
    int temp;
    for(int i=0;i<roww;i++)
    {
        for(int j=0;j<coll/2;j++)
        {
            temp=MM[i][j];
            MM[i][j]=MM[i][coll-1-j];
            MM[i][coll-1-j]=temp;
        }
    }
}

```

```

void axial_symmetry_red()
{
    int temp;
    for(int i=0;i<roww_red;i++)
    {
        for(int j=0;j<coll_red/2;j++)
        {
            temp=M_reduction[i][j];
            M_reduction[i][j]=M_reduction[i][coll_red-1-j];
            M_reduction[i][coll_red-1-j]=temp;
        }
    }
}

```

```

void life(int &life_kolvo)
{
    //cout<<"life0"<<endl;
    int first,second,alive_sosed,flag;
    int first_new,second_new;
    //cout<<"life0.5"<<endl;
    //print_deq2(living);
    for(int k=0;k<living.size();k+=2)
    {
        first=living[k];
        second=living[k+1];
        alive_sosed=0;
        //cout<<"first="<<first<<" second="<<second<<endl;
        //cout<<"k="<<k<<endl;
        for(int i=first-1;i<=first+1;i++)
        {
            for(int u=second-1;u<=second+1;u++)
            {
                //cout<<"i="<<i<<" u="<<u<<endl;
                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++;
            }
        }
    }
}
if(alive_sosed==3 && MM[first][second]==0)
{
    //cout<<"k="<<k<<" first="<<first<<" second="<<second<<endl;
    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<<"life1"<<endl;
//print_deq2(new_life);
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++)
        {
            //BINARY SELECT
            flag=0;
            if(select_binary_search(x,y,living)==1)
            {
                flag=1;
            }
            if(flag==0)
            {
                //BINARY SELECT
                flag=0;
                if(select_binary_search(x,y,living_candidats)==1)
                {
                    flag=1;
                }
                //BINARY INSERT
                if(flag==0)
                {
                    insert_binary_search(x,y,living_candidats);
                }
            }
        }
    }
}

```

```

    }
    }
}
//cout<<"life2"<<endl;

for(int k=0;k<old_life.size();k+=2)
{
    first=old_life[k];
    second=old_life[k+1];
    MM[first][second]=0;
}
//cout<<"life3"<<endl;

//EXPANSION
int max_x,max_y,min_x,min_y;
int flag1=0, flag_expand_compress=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;
            flag_expand_compress=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;
            flag_expand_compress=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;
            flag_expand_compress=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;
            flag_expand_compress=1;
        }
    }
}
//cout<<"life4"<<endl;

int flag_mass_expand_compress[4]={0,0,0,0};
if(flag_expand_compress==1)
{
    //East
    //Expansion
    if(max_y==coll-1)
    {
        MM=expansion_E(MM,&rr,roww,coll,1,1,0);
        life_exp.push_back(1);
        flag_mass_expand_compress[0]=1;
    }
    //Compression
    else if(max_y<=coll-3)
    {
        life_exp.push_back((coll-3-max_y+1)*(-1));
        MM=expansion_E(MM,&rr,roww,coll,coll-3-max_y+1,-1,0);
        flag_mass_expand_compress[0]=1;
    }
    else if(flag_mass_expand_compress[0]==0)
    {
        life_exp.push_back(0);
    }
}

//West
//Expansion
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);
    flag_mass_expand_compress[1]=1;
}
//Compression
else if(min_y>=2)
{
    life_exp.push_back((min_y-2+1)*(-1));
    MM=expansion_W(MM,&rr,roww,coll,min_y-2+1,-1,0);
}
}

```



```

        coord_recalc(1,-(min_y-2+1));
        coord_recalc_candidates(1,-(min_y-2+1));

        max_y-=(min_y-2+1);
        min_y-=(min_y-2+1);
        flag_mass_expand_compress[1]=1;
    }
    else if(flag_mass_expand_compress[1]==0)
    {
        life_exp.push_back(0);
    }

//South
//Expansion
if(max_x==roww-1)
{
    MM=expansion_S(MM,&rr,roww,coll,1,1,0);
    life_exp.push_back(1);
    flag_mass_expand_compress[2]=1;
}
//Compression
else if(max_x<=roww-3)
{
    life_exp.push_back((roww-3-max_x+1)*(-1));
    MM=expansion_S(MM,&rr,roww,coll,roww-3-max_x+1,-1,0);
    flag_mass_expand_compress[2]=1;
}
else if(flag_mass_expand_compress[2]==0)
{
    life_exp.push_back(0);
}

//North
//Expansion
if(min_x==0)
{
    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);
    flag_mass_expand_compress[3]=1;
}
//Compression
else if(min_x>=2)
{
    life_exp.push_back((min_x-2+1)*(-1));
    MM=expansion_N(MM,&rr,roww,coll,min_x-2+1,-1,0);

    coord_recalc(-1,-(min_x-2+1));
    coord_recalc_candidates(-1,-(min_x-2+1));

    max_x-=(min_x-2+1);
}

```

```

        min_x=(min_x-2+1);
        flag_mass_expand_compress[3]=1;
    }
    else if(flag_mass_expand_compress[3]==0)
    {
        life_exp.push_back(0);
    }
}
//cout<<"life5"<<endl;

//BINARY INSERT
for(int i=0;i<living_candidats.size();i+=2)
{
    insert_binary_search(living_candidats[i],living_candidats[i+1],living);
}
//cout<<"life6"<<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)
    {
        living.erase(living.begin()+i);
        living.erase(living.begin()+i);
        i-=2;
    }
}
//cout<<"life7"<<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<roww;i++)
    {
        for(int u=second-1;u<=second+1 && u<coll;u++)
        {
            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++;
                }
            }
        }
    }
}

```

```

    }
    }
}

if(MM[first][second]==0 && alive_sosed==0)
{
    killing_candidats.push_back(k);
}
}
//cout<<"life8"<<endl;

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<<"life9"<<endl;
life_kolvo+=new_life.size()/2-old_life.size()/2;
}

```

```

void create_memory(int n)
{
    free(MM);
    free(rr);
    life_pop_max=0;
    roww=sqrt(n)+2;
    coll=sqrt(n)+2;
    MM=(int**)calloc(roww,sizeof(int*));
    rr=(int*)calloc(roww*coll,sizeof(int));
    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 dimension_collapse(int n)
{
    int W=0,E=0,S=0,N=0;
    for(int i=1;i<n-1;i++)
    {
        if(MM[i][1]==1)
        {
            W=1;
            break;
        }
    }
    for(int i=1;i<n-1;i++)

```

```

    {
        if(MM[i][n-2]==1)
        {
            E=1;
            break;
        }
    }
    if(E==1 && W==1)
    {
        return 1;
    }
    for(int i=1;i<n-1;i++)
    {
        if(MM[1][i]==1)
        {
            N=1;
            break;
        }
    }
    for(int i=1;i<n-1;i++)
    {
        if(MM[n-2][i]==1)
        {
            S=1;
            break;
        }
    }
    if(S==1 && N==1)
    {
        return 1;
    }
    return 0;
}

int serch_in_bank(string current_hash)
{
    for(int i=0;i<bank_hash.size();i++)
    {
        if(bank_hash[i]==current_hash)
        {
            return 1;
        }
    }
    return 0;
}

int bank_proverka()
{
    string temp_hash;
    if(bank_hash.size()!=0)
    {
        for(int i=0;i<4;i++)
        {
            temp_hash=virtual_rotateMN_clockwise(i);

```

```

        if(serch_in_bank(temp_hash))
        {
            return 0;
        }
    }
    axial_symmetry();
    for(int i=0;i<4;i++)
    {
        temp_hash=virtual_rotateMN_clockwise(i);
        if(serch_in_bank(temp_hash))
        {
            return 0;
        }
    }
    axial_symmetry();
    return 1;
}
return 1;
}

int truncate_db_unique()
{
    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

    struct connection_details mysqlID;

    mysqlID.server = "localhost"; // where the mysql database is
    mysqlID.user = "phpmyadmin"; // user
    mysqlID.password = "dragon3000"; // the password for the database
    mysqlID.database = "LIFE_GAME"; // the databse
    //cout<<"truncate_db_unique ";
    string my_query="TRUNCATE TABLE `LIFE_GAME`.`UNIQUE_CASES`";
    con = mysql_connection_setup(mysqlID);
    res = mysql_perform_query(con, my_query.c_str());
    mysql_free_result(res);
    mysql_close(con);

    my_query="TRUNCATE TABLE `LIFE_GAME`.`LOG_GAMES`";
    con = mysql_connection_setup(mysqlID);
    res = mysql_perform_query(con, my_query.c_str());
    mysql_free_result(res);
    mysql_close(con);

    return 1;
}

int insert_db_log(int time_laps)
{
    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

```

```

struct connection_details mysqlID;

mysqlID.server = "localhost"; // where the mysql database is
mysqlID.user = "phpmyadmin"; // user
mysqlID.password = "dragon3000"; // the password for the database
mysqlID.database = "LIFE_GAME"; // the databse

string my_query="";
my_query+="insert into `LOG_GAMES` ";
my_query+="(`LOG_GAMES_ID`,`ENUMERATION_NUM`,`GAME_SIZE`,`TIME_LAPS`,`G_WAIT`,`G_HASH`,`G_FLIGHT`,`G_SUSPICIOUS_SECTOR`,`G_VICINITY`,`INSERT_DATE`) ";
my_query+="VALUES (NULL, ";
my_query+=to_string(enum_counter);
my_query+="", ";
my_query+=to_string(EXPERIMENT_SIZE);
my_query+="", ";
my_query+=to_string(time_laps);
my_query+="", ";
my_query+=to_string(WAIT);
my_query+="", ";
my_query+=to_string(HASH);
my_query+="", ";
my_query+=to_string(FLIGHT);
my_query+="", ";
my_query+=to_string(SUSPICIOUS_SECTOR);
my_query+="", ";
my_query+=to_string(VICINITY);
my_query+="", NOW()); ";

con = mysql_connection_setup(mysqlID);
res = mysql_perform_query(con, my_query.c_str());
mysql_free_result(res);
mysql_close(con);
return 1;
}

int insert_hash_db_interesting(string hash, int* ar, int counter_kill)
{
string interesting_ar="";
for(int i=0;i<EXPERIMENT_SIZE;i++)
{
interesting_ar+=to_string(ar[i]);
}
MYSQL *con; // the connection
MYSQL_RES *res; // the results
MYSQL_ROW row; // the results rows (array)

struct connection_details mysqlID;

mysqlID.server = "localhost"; // where the mysql database is
mysqlID.user = "phpmyadmin"; // user
mysqlID.password = "dragon3000"; // the password for the database
mysqlID.database = "LIFE_GAME"; // the databse

```

```

    string my_query="";
    my_query+="insert into `INTERESTING_CASES` ";
    my_query+="(`INTERESTING_CASES_ID`, `HASH`,
`INTERESTING_AR`, `ENUMERATION_NUM`, `GAME_SIZE`, `DEAD_SPACESHIPS`,
`G_WAIT`, `G_HASH`, `G_FLIGHT`, `G_SUSPICIOUS_SECTOR`, `G_VICINITY`,
`INSERT_DATE`) ";
    my_query+="VALUES (NULL, ";
    my_query+=hash;
    my_query+="", ";
    my_query+=interesting_ar;
    my_query+="", ";
    my_query+=to_string(enum_counter);
    my_query+="", ";
    my_query+=to_string(EXPERIMENT_SIZE);
    my_query+="", ";
    my_query+=to_string(counter_kill);
    my_query+="", ";
    my_query+=to_string(WAIT);
    my_query+="", ";
    my_query+=to_string(HASH);
    my_query+="", ";
    my_query+=to_string(FLIGHT);
    my_query+="", ";
    my_query+=to_string(SUSPICIOUS_SECTOR);
    my_query+="", ";
    my_query+=to_string(VICINITY);
    my_query+="", NOW()); ";

    con = mysql_connection_setup(mysqlID);
    /*insert into `INTERESTING_CASES` (`INTERESTING_CASES_ID`, `HASH`,
`INTERESTING_AR`, `GAME_SIZE`) VALUES (NULL, "+hash+",
"+interesting_ar+", "+EXPERIMENT_SIZE+"); */
    res = mysql_perform_query(con, my_query.c_str());
    mysql_free_result(res);
    mysql_close(con);
    return 1;
}

int insert_hash_db_all(string hash)
{
    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

    struct connection_details mysqlID;

    mysqlID.server = "localhost"; // where the mysql database is
    mysqlID.user = "phpmyadmin"; // user
    mysqlID.password = "dragon3000"; // the password for the database
    mysqlID.database = "LIFE_GAME"; // the databse

    string my_query="";
    my_query+="insert IGNORE into `UNIQUE_CASES` ";
    my_query+="(`UNIQUE_CASES_ID`, `HASH`, `GAME_SIZE`) ";

```

```

my_query+="VALUES (NULL, ";
my_query+=hash;
my_query+="", ";
my_query+=to_string(EXPERIMENT_SIZE);
my_query+=""); ";

con = mysql_connection_setup(mysqlID);
//insert into `UNIQUE_CASES` (`UNIQUE_CASES_ID`, `HASH`, `GAME_SIZE`)
VALUES (NULL, ""+hash+"", ""+EXPERIMENT_SIZE+"") "
//cout<<my_query<<endl;
res = mysql_perform_query(con, my_query.c_str());
//cout<<"here"<<endl;
mysql_free_result(res);
mysql_close(con);
return 1;
}

int select_hash_db_all(string hash)
{
int flag_query=0;

MYSQL *con; // the connection
MYSQL_RES *res; // the results
MYSQL_ROW row; // the results rows (array)

struct connection_details mysqlID;

mysqlID.server = "localhost"; // where the mysql database is
mysqlID.user = "phpmyadmin"; // user
mysqlID.password = "dragon3000"; // the password for the database
mysqlID.database = "LIFE_GAME"; // the database
//select `UNIQUE_CASES_ID` from `UNIQUE_CASES` WHERE
`HASH`="" +temp_hash+"";
string my_query="";
my_query+="select `UNIQUE_CASES_ID` ";
my_query+="from UNIQUE_CASES ";
my_query+="WHERE `HASH`=";
my_query+=hash;
my_query+="";";

con = mysql_connection_setup(mysqlID);
res = mysql_perform_query(con, my_query.c_str());
while ((row = mysql_fetch_row(res)) != NULL)
{
flag_query=1;
break;
}
mysql_free_result(res);
mysql_close(con);
return flag_query;
}

int bank_proverka_with_db()
{

```



```

string temp_hash;
int flag_query=0;
int result;

for(int i=0;i<4;i++)
{
    temp_hash=virtual_rotateMN_clockwise(i);
    if(i==0)
    {
        flag_query=select_hash_db_all(temp_hash);
        if(flag_query==1)
        {
            return 0;
        }
    }
    result=insert_hash_db_all(temp_hash);
}
axial_symmetry();
for(int i=0;i<4;i++)
{
    temp_hash=virtual_rotateMN_clockwise(i);
    result=insert_hash_db_all(temp_hash);
}
axial_symmetry();
return 1;
}

// int bank_insert_with_db()
// {
//     string temp_hash;
//     int result;

//     for(int i=0;i<4;i++)
//     {
//         temp_hash=virtual_rotateMN_clockwise(i);
//         result=insert_hash_db_all(temp_hash);
//     }
//     axial_symmetry();
//     for(int i=0;i<4;i++)
//     {
//         temp_hash=virtual_rotateMN_clockwise(i);
//         result=insert_hash_db_all(temp_hash);
//     }
//     axial_symmetry();
//     return result;
// }

int insert_hash_db_bank(string hash)
{
    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

```

```

struct connection_details mysqlID;

mysqlID.server = "localhost"; // where the mysql database is
mysqlID.user = "phpmyadmin"; // user
mysqlID.password = "dragon3000"; // the password for the database
mysqlID.database = "LIFE_GAME"; // the database

string my_query="";
my_query+="insert IGNORE into `SPACESHIP_HASH_BANK` ";
my_query+="(`SPACESHIP_HASH_BANK_ID`, `HASH`) ";
my_query+="VALUES (NULL, ";
my_query+=hash;
my_query+=")";

con = mysql_connection_setup(mysqlID);
//"insert into `SPACESHIP_HASH_BANK` (`SPACESHIP_HASH_BANK_ID`, `HASH`)
VALUES (NULL, "+hash+" ) "
//cout<<my_query<<endl;
res = mysql_perform_query(con, my_query.c_str());
//cout<<"here"<<endl;
mysql_free_result(res);
mysql_close(con);
return 1;
}
void spaceship_bank_fill_db()
{
string temp_hash;
for(int i=0;i<4;i++)
{
temp_hash=virtual_rotateMN_clockwise_red(i);
insert_hash_db_bank(temp_hash);
}
axial_symmetry_red();
for(int i=0;i<4;i++)
{
temp_hash=virtual_rotateMN_clockwise_red(i);
insert_hash_db_bank(temp_hash);
}
axial_symmetry_red();
}
void spaceship_bank_fill()
{
string temp_hash;
fstream fp;
fp.open("spaceship_hash_bank.txt", ios::app);
for(int i=0;i<4;i++)
{
temp_hash=virtual_rotateMN_clockwise_red(i);
fp<<temp_hash<<endl;
}
axial_symmetry_red();
for(int i=0;i<4;i++)
{
temp_hash=virtual_rotateMN_clockwise_red(i);
fp<<temp_hash<<endl;
}
}

```

```

    }
    axial_symmetry_red();
    fp.close();
}

void glider_directions_bank_fill()
{
    string temp_hash;
    fstream fp[4];

    //fp[0].open("glider_S-W.txt", ios::app);
    //fp[1].open("glider_N-W.txt", ios::app);
    //fp[2].open("glider_N-E.txt", ios::app);
    //fp[3].open("glider_S-E.txt", ios::app);

    fp[0].open("glider_S-W.txt", ios::app);
    fp[1].open("glider_S-E.txt", ios::app);
    fp[2].open("glider_N-E.txt", ios::app);
    fp[3].open("glider_N-W.txt", ios::app);

    /*fp[0].open("glider_S-E.txt", ios::app);
    fp[1].open("glider_S-W.txt", ios::app);
    fp[2].open("glider_N-W.txt", ios::app);
    fp[3].open("glider_N-E.txt", ios::app);*/
    for(int k=0;k<4;k++)
    {
        temp_hash=virtual_rotateMN_clockwise_red(k);
        fp[k]<<temp_hash<<endl;
    }
    for(int i=0;i<4;i++)
    {
        fp[i].close();
    }
}

int insert_hash_db_directions(string hash, int direct)
{
    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

    struct connection_details mysqlID;

    mysqlID.server = "localhost"; // where the mysql database is
    mysqlID.user = "phpmyadmin"; // user
    mysqlID.password = "dragon3000"; // the password for the database
    mysqlID.database = "LIFE_GAME"; // the databse

    string my_query="";
    my_query+="insert IGNORE into `GLIDER_DIRECTIONS` ";
    my_query+="(`GLIDER_DIRECTIONS_ID`, `HASH`, `DIRECTION`) ";
    my_query+="VALUES (NULL, ";
    my_query+=hash;
    my_query+="", ";
    my_query+=to_string(direct);

```

```

my_query+=""); ";

con = mysql_connection_setup(mysqlID);
//"insert into `GLIDER DIRECTIONS` (`GLIDER DIRECTIONS_ID`, `HASH`,
`DIRECTION`) VALUES (NULL, ""+hash+", ""+direct+"") "
//cout<<my_query<<endl;
res = mysql_perform_query(con, my_query.c_str());
//cout<<"here"<<endl;
mysql_free_result(res);
mysql_close(con);
return 1;
}
void glider_directions_bank_fill_db()
{
string temp_hash;
// 0-glider_S-W
// 1-glider_S-E
// 2-glider_N-E
// 3-glider_N-W

//k
// 0-glider_N-E
// 1-glider_N-W
// 2-glider_S-W
// 3-glider_S-E

// (4-k)%4 - sw nw ne se
// 3-(4-k)%4 - se ne nw sw
// (-k)%4+3 - nw ne se sw

//((-k)%4+3-1+4)%4
//N-E -> 0
//S-E -> 1
//S-W -> 2
//N-W -> 3
for(int k=0;k<4;k++)
{
temp_hash=virtual_rotateMN_clockwise_red(k);
insert_hash_db_directions(temp_hash,((-k)%4+3-1+4)%4);
}
}

void file_to_vector_spaceship()
{
spaceship_bank_hash.clear();
fstream fp;
string where;
fp.open("spaceship_hash_bank.txt");
while(fp>>where)
{
spaceship_bank_hash.push_back(where);
}
fp.close();
}

```

```

void select_db_to_vector_spaceship()
{
    spaceship_bank_hash.clear();

    MYSQL *con; // the connection
    MYSQL_RES *res; // the results
    MYSQL_ROW row; // the results rows (array)

    struct connection_details mysqlD;

    mysqlD.server = "localhost"; // where the mysql database is
    mysqlD.user = "phpmyadmin"; // user
    mysqlD.password = "dragon3000"; // the password for the database
    mysqlD.database = "LIFE_GAME"; // the database
    //"select `HASH` from `SPACESHIP_HASH_BANK` WHERE 1;"
    string my_query="";
    my_query+="select `HASH` ";
    my_query+="from SPACESHIP_HASH_BANK ";
    my_query+="WHERE 1;";

    con = mysql_connection_setup(mysqlD);
    res = mysql_perform_query(con, my_query.c_str());
    string key_for_map;
    int value_for_map;
    //cout<<"HERE1"<<endl;
    while ((row = mysql_fetch_row(res)) != NULL)
    {

        spaceship_bank_hash.push_back(row[0]);

    }
    //cout<<"HERE2"<<endl;
    mysql_free_result(res);
    mysql_close(con);
}

```

```

int compare_mass_flight(int i, int j, vector<int> index_mass)
{
    //E W S N
    int result;
    int k, flag;
    if(i<0 || j<0)
    {
        result=-1;
    }
    else
    {
        result=1;
        if(result ==1 && index_mass[3]==1)
        {
            if(life_exp[i]!=life_exp[j])
            {
                result=0;
            }
        }
    }
}

```

```

    if(result ==1 && index_mass[2]==1)
    {
        if(life_exp[i-1]!=life_exp[j-1])
        {
            result=0;
        }
    }
    if(result ==1 && index_mass[1]==1)
    {
        if(life_exp[i-2]!=life_exp[j-2])
        {
            result=0;
        }
    }
    if(result ==1 && index_mass[0]==1)
    {
        if(life_exp[i-3]!=life_exp[j-3])
        {
            result=0;
        }
    }
    //preventing case with only 0
}

return result;
}
int compare_mass_bloc(int position, int flag_dif_mass[], vector<int> index_mass)
{
    for(int j=0;j<4;j++)
    {
        if(index_mass[j]==1)
        {
            if(life_exp[position+j]!=flag_dif_mass[j])
            {
                return 0;
            }
        }
    }
    return 1;
}
void appropriate(vector<int> index_mass, int flag_dif_mass[], int position)
{
    for(int i=0;i<4;i++)
    {
        if(index_mass[i]==0)
        {
            flag_dif_mass[i]=0;
        }
        else
        {
            flag_dif_mass[i]=life_exp[position+i];
        }
    }
}
int flight_step(int flag_syptom, vector<int> index_mass)

```

```

{
  int flag_dif_mass[4]={0,0,0,0};
  int flag_dif=-1;
  int last_element=life_exp.size()-1;
  int flag;
  int number_of_periods=FLIGHT;
  int nop;
  for(int i=life_exp.size()-1;i>0;i-=4)
  {
    if(i!=life_exp.size()-1 && compare_mass_flight(i,last_element,index_mass)==1 &&
flag_symptom==0)
    {
      if((life_exp.size()-1-i)*4>life_exp.size()/2)
      {
        flag_symptom=-1;
      }
      else
      {
        flag=0;
        nop=1;
        while(nop<=number_of_periods)
        {
          for(int u=i-(life_exp.size()-1-i)*nop+1+3;u<i-(life_exp.size()-1-i)*(nop-1);u+=4)
          {
            if(compare_mass_flight(u,u+life_exp.size()-1-i,index_mass)<=0)
            {
              flag=1;
              break;
            }
          }
          nop++;
        }
        int counter;
        if(flag==0 && flag_dif==-2)
        {
          for(int g=0;g<4;g++)
          {
            if(index_mass[g]==1)
            {
              counter=0;
              for(int h=i+1+g;h<life_exp.size();h+=4)
              {
                counter+=life_exp[h];
              }
              if(counter>0)
              {
                flag_symptom=1;
                break;
              }
            }
            else
            {
              flag_symptom=-1;
              break;
            }
          }
          break;
        }
      }
    }
  }
}

```



```

        if(flag==0)
        {
            period_length=life_hash.size()-1-i;
            simptom=1;
        }
    }
}
if(life_pop[i]>life_pop_max)
{
    life_pop_max=life_pop[i];
}
}
return simptom;
}

```

```

int import_matrix_size(char* s)
{
    int symba,row=0,col=0,n;
    int flag=0;
    int counter=0;
    FILE* fp=fopen(s,"r");
    while(fscanf(fp,"%c",&symba)!=-1)
    {
        //not space
        if(symba!=32)
        {
            if(flag==0 && (symba==95 || symba==111))
            {
                counter++;
            }
            //if \n
            else if(symba==10)
            {
                if(flag==0)
                {
                    col=counter;
                    flag=1;
                }
                row++;
            }
        }
    }
}
fclose(fp);
if(col>row)
{
    n=col;
}
else
{
    n=row;
}
return n*n;
}
int import_matrix_size_string(string s)
{

```

```

int row=0,col=0,n;
char symba;
int flag=0;
int counter=0;
fstream fp;
fp.open(s);
while(fp.get(symba))
{
    //not space
    if(symba!=32)
    {

        if(flag==0 && (symba==95 || symba==111))
        {
            counter++;
        }
        //if \n
        else if(symba==10)
        {
            if(flag==0)
            {
                col=counter;
                flag=1;
            }
            row++;
        }
    }
}
fp.close();
if(col>row)
{
    n=col;
}
else
{
    n=row;
}
return n*n;
}
void import_matrix(char* s)
{
    int symba;
    int row=0,col=0;
    int flag;
    int cuonter_symba=0;
    FILE* fp=fopen(s,"r");

    while(fscanf(fp,"%c",&symba)!=-1)
    {
        //not space
        if(symba!=32)
        {
            cuonter_symba++;
            //if _
            if(symba==95)

```

```

    {
        MM[row+1][col+1]=0;
        col++;
    }
    //if o
    else if(symba==111)
    {
        MM[row+1][col+1]=1;

        for(int x=(row+1)-1;x<=(row+1)+1;x++)
        {
            for(int y=(col+1)-1;y<=(col+1)+1;y++)
            {
                //BINARY SELECT
                flag=0;
                if(select_binary_search(x,y,living)==1)
                {
                    flag=1;
                }

                //BINARY INSERT
                if(flag==0)
                {
                    insert_binary_search(x,y,living);
                }
            }
        }
        col++;

    }
    //if \n
    else if(symba==10)
    {
        row++;
        col=0;
    }
    else
    {
        printf("MURDER => %d \n",symba);
    }
}
}
fclose(fp);
//cout<<"finish"<<endl;
}

```

```

void import_matrix_string(string s)
{
    char symba;
    int row=0,col=0;
    int flag;
    int cuonter_symba=0;
    fstream fp;
    fp.open(s);
}

```

```

while(fp.get(symba))
{
    //not space
    if(symba!=32)
    {
        cuonter_symba++;
        //if _
        if(symba==95)
        {
            MM[row+1][col+1]=0;
            col++;
        }
        //if o
        else if(symba==111)
        {
            MM[row+1][col+1]=1;

            for(int x=(row+1)-1;x<=(row+1)+1;x++)
            {
                for(int y=(col+1)-1;y<=(col+1)+1;y++)
                {
                    //BINARY SELECT
                    flag=0;
                    if(select_binary_search(x,y,living)==1)
                    {
                        flag=1;
                    }

                    //BINARY INSERT
                    if(flag==0)
                    {
                        insert_binary_search(x,y,living);
                    }
                }
            }
            col++;
        }

        //if \n
        else if(symba==10)
        {
            row++;
            col=0;
        }
        else
        {
            printf("MURDER => %d \n",symba);
        }
    }
}
fp.close();
}

```

```

void reduction()
{
    int max_x,max_y,min_x,min_y;
    int flag1=0, flag_expand_compress=0;
    for(int i=0;i<roww_red && flag1==0;i++)
    {
        for(int u=0;u<coll_red && flag1==0;u++)
        {
            if(M_reduction[i][u]==1 && flag1==0)
            {
                min_x=i;
                flag1=1;
                flag_expand_compress=1;
            }
        }
    }
    flag1=0;
    for(int i=roww_red-1;i>=0 && flag1==0;i--)
    {
        for(int u=coll_red-1;u>=0 && flag1==0;u--)
        {
            if(M_reduction[i][u]==1 && flag1==0)
            {
                max_x=i;
                flag1=1;
                flag_expand_compress=1;
            }
        }
    }
    flag1=0;
    for(int i=coll_red-1;i>=0 && flag1==0;i--)
    {
        for(int u=roww_red-1;u>=0 && flag1==0;u--)
        {
            if(M_reduction[u][i]==1 && flag1==0)
            {
                max_y=i;
                flag1=1;
                flag_expand_compress=1;
            }
        }
    }
    flag1=0;
    for(int i=0;i<coll_red && flag1==0;i++)
    {
        for(int u=0;u<roww_red && flag1==0;u++)
        {
            if(M_reduction[u][i]==1 && flag1==0)
            {
                min_y=i;
                flag1=1;
                flag_expand_compress=1;
            }
        }
    }
}

```

```

min_y++;
min_x++;
max_y--;
max_x--;
int flag_mass_expand_compress[4]={0,0,0,0};
if(flag_expand_compress==1)
{
    //East
    //Expansion
    if(max_y==coll_red-1)
    {
        M_reduction=expansion_E(M_reduction,&r_reduction,roww_red,coll_red,1,1,0);
        life_exp.push_back(1);
        flag_mass_expand_compress[0]=1;
    }
    //Compression
    else if(max_y<=coll_red-3)
    {
        life_exp.push_back((coll_red-3-max_y+1)*(-1));
    }
}
M_reduction=expansion_E(M_reduction,&r_reduction,roww_red,coll_red,coll_red-3-max_y+
1,-1,0);
    flag_mass_expand_compress[0]=1;
}
/*else if(flag_mass_expand_compress[0]==0)
{
    life_exp.push_back(0);
}*/

//West
//Expansion
if(min_y==0)
{
    M_reduction=expansion_W(M_reduction,&r_reduction,roww_red,coll_red,1,1,0);
    max_y++;
    min_y++;
    life_exp.push_back(1);
    flag_mass_expand_compress[1]=1;
}
//Compression
else if(min_y>=2)
{
    life_exp.push_back((min_y-2+1)*(-1));
}
}
M_reduction=expansion_W(M_reduction,&r_reduction,roww_red,coll_red,min_y-2+1,-1,0);
    max_y--=(min_y-2+1);
    min_y--=(min_y-2+1);
    flag_mass_expand_compress[1]=1;
}
/*else if(flag_mass_expand_compress[1]==0)
{
    life_exp.push_back(0);
}*/

```

```

//South
//Expansion
if(max_x==roww_red-1)
{
    M_reduction=expansion_S(M_reduction,&r_reduction,roww_red,coll_red,1,1,0);
    life_exp.push_back(1);
    flag_mass_expand_compress[2]=1;
}
//Compression
else if(max_x<=roww_red-3)
{
    life_exp.push_back((roww_red-3-max_x+1)*(-1));

M_reduction=expansion_S(M_reduction,&r_reduction,roww_red,coll_red,roww_red-3-max_x
+1,-1,0);
    flag_mass_expand_compress[2]=1;
}
/*else if(flag_mass_expand_compress[2]==0)
{
    life_exp.push_back(0);
}*/

//North
//Expansion
if(min_x==0)
{
    M_reduction=expansion_N(M_reduction,&r_reduction,roww_red,coll_red,1,1,0);
    max_x++;
    min_x++;
    life_exp.push_back(1);
    flag_mass_expand_compress[3]=1;
}
//Compression
else if(min_x>=2)
{
    life_exp.push_back((min_x-2+1)*(-1));

M_reduction=expansion_N(M_reduction,&r_reduction,roww_red,coll_red,min_x-2+1,-1,0);
    max_x--=(min_x-2+1);
    min_x--=(min_x-2+1);
    flag_mass_expand_compress[3]=1;
}
/*else if(flag_mass_expand_compress[3]==0)
{
    life_exp.push_back(0);
}*/
}
}

void catch_vicinity(int x, int y)
{
    int min_x=-1,max_x=-1,min_y=-1,max_y=-1;
    int coat_min_x,coat_max_x,coat_min_y,coat_max_y;
    if(x-VICINITY>0)
    {

```

```

    coat_min_x=x-VICINITY;
}
else
{
    coat_min_x=0;
}
if(y-VICINITY>0)
{
    coat_min_y=y-VICINITY;
}
else
{
    coat_min_y=0;
}
if(x+VICINITY<roww)
{
    coat_max_x=x+VICINITY;
}
else
{
    coat_max_x=roww;
}
if(y+VICINITY<coll)
{
    coat_max_y=y+VICINITY;
}
else
{
    coat_max_y=coll;
}

free(M_reduction);
free(r_reduction);

roww_red=coat_max_x-coat_min_x;
coll_red=coat_max_y-coat_min_y;

M_reduction=(int**)calloc((roww_red),sizeof(int*));
r_reduction=(int*)calloc(((roww_red)*(coll_red)),sizeof(int));

for(int i=0;i<roww_red;i++)
{
    M_reduction[i]=r_reduction+i*coll_red;
}

for(int i=coat_min_x;i<coat_max_x;i++)
{
    for(int u=coat_min_y;u<coat_max_y;u++)
    {
        M_reduction[i-coat_min_x][u-coat_min_y]=MM[i][u];
    }
}
reduction();
string hash_for_test=create_hash_from_array_reduction();
file_to_vector_spaceship();

```



```

int flag_kill_spaceship=0;
for(int i=0;i<spaceship_bank_hash.size();i++)
{
    if(hash_for_test==spaceship_bank_hash[i])
    {
        flag_kill_spaceship=1;
        break;
    }
}
if(flag_kill_spaceship==1)
{
    //cout<<"Kill"<<endl;
    for(int i=coat_min_x;i<coat_max_x;i++)
    {
        for(int u=coat_min_y;u<coat_max_y;u++)
        {
            if(MM[i][u]==1)
            {
                MM[i][u]=0;
            }
        }
    }
}
}

```

```

void catch_vicinity_exp(int x, int y, vector<int> &directions, vector<int> &coats)
{
    //directions
    //N-E -> 0
    //S-E -> 1
    //S-W -> 2
    //N-W -> 3

    //coats
    //coat_min_x -> 0
    //coat_max_x -> 1
    //coat_min_y -> 2
    //coat_max_y -> 3
    int min_x=-1,max_x=-1,min_y=-1,max_y=-1;
    int coat_min_x,coat_max_x,coat_min_y,coat_max_y;

    if(x-VICINITY>0)
    {
        coat_min_x=x-VICINITY;
    }
    else
    {
        coat_min_x=0;
    }
    if(y-VICINITY>0)
    {
        coat_min_y=y-VICINITY;
    }
    else
    {

```

```

    coat_min_y=0;
}
if(x+VICINITY<roww)
{
    coat_max_x=x+VICINITY;
}
else
{
    coat_max_x=roww;
}
if(y+VICINITY<coll)
{
    coat_max_y=y+VICINITY;
}
else
{
    coat_max_y=coll;
}

free(M_reduction);
free(r_reduction);

roww_red=coat_max_x-coat_min_x;
coll_red=coat_max_y-coat_min_y;

M_reduction=(int**)calloc((roww_red),sizeof(int*));
r_reduction=(int*)calloc(((roww_red)*(coll_red)),sizeof(int));

for(int i=0;i<roww_red;i++)
{
    M_reduction[i]=r_reduction+i*coll_red;
}

for(int i=coat_min_x;i<coat_max_x;i++)
{
    for(int u=coat_min_y;u<coat_max_y;u++)
    {
        M_reduction[i-coat_min_x][u-coat_min_y]=MM[i][u];
    }
}
//print_matrix(MM,roww,coll);
//print_matrix(M_reduction,roww_red,coll_red);
reduction();
//print_matrix(M_reduction,roww_red,coll_red);
string hash_for_test=create_hash_from_array_reduction();
file_to_vector_spaceship();
//select_db_to_vector_spaceship();
int flag_kill_spaceship=0;
fstream fp;
string where;
for(int i=0;i<spaceship_bank_hash.size();i++)
{
    if(hash_for_test==spaceship_bank_hash[i])
    {
        flag_kill_spaceship=1;
    }
}

```

```

        break;
    }
}
//cout<<"hash_for_test--> "<<hash_for_test<<endl<<endl;
//cout<<"flag_kill_spaceship="<<flag_kill_spaceship<<endl;
//TODO
//error in spaceship_hash_bank
//maybe incorrect hashes are stored in file
//or we incorrectly generate hash_for_test
if(flag_kill_spaceship==1)
{
    if(flag_kill_spaceship==1)
    {
        fp.open("glider_N-E.txt");
        while(fp>>where)
        {
            //cout<<"N-E--> "<<where<<endl;
            if(hash_for_test==where)
            {
                directions.push_back(0);
                flag_kill_spaceship=2;
                break;
            }
        }
        fp.close();
    }
    if(flag_kill_spaceship==1)
    {
        fp.open("glider_S-E.txt");
        while(fp>>where)
        {
            //cout<<"S-E--> "<<where<<endl;
            if(hash_for_test==where)
            {
                directions.push_back(1);
                flag_kill_spaceship=2;
                break;
            }
        }
        fp.close();
    }
    if(flag_kill_spaceship==1)
    {
        fp.open("glider_S-W.txt");
        while(fp>>where)
        {
            //cout<<"S-W--> "<<where<<endl;
            if(hash_for_test==where)
            {
                directions.push_back(2);
                flag_kill_spaceship=2;
                break;
            }
        }
        fp.close();
    }
}

```

```

}
if(flag_kill_spaceship==1)
{
    fp.open("glider_N-W.txt");
    while(fp>>where)
    {
        //cout<<"N-W--> "<<where<<endl;
        if(hash_for_test==where)
        {
            directions.push_back(3);
            flag_kill_spaceship=2;
            break;
        }
    }
    fp.close();
}
if(flag_kill_spaceship==1)
{
    for(int i=coat_min_x;i<coat_max_x;i++)
    {
        for(int u=coat_min_y;u<coat_max_y;u++)
        {
            if(MM[i][u]==1)
            {
                MM[i][u]=0;
            }
        }
    }
}

if(flag_kill_spaceship==2)
{
    //cout<<"Kill"<<endl;
    coats.push_back(coat_min_x);
    coats.push_back(coat_max_x);
    coats.push_back(coat_min_y);
    coats.push_back(coat_max_y);
}
}
void select_hash_db_directions(map<string,int> &direct)
{
    // map<int,int> test;
    // map<int,int>::iterator it;
    // test[7]=10;
    // test[8]=11;
    // it = test.find(1);
    // if (it == test.end())
    //     cout<<"error"<<endl;
    // else
    // {
    //     cout<<"ok"<<endl;
    //     //test.erase(it);
    // }
}

```

```

// for(auto it = test.cbegin(); it != test.cend(); ++it)
// {
//   cout<<it->first<<" "<<it->second<<endl;
// }

MYSQL *con; // the connection
MYSQL_RES *res; // the results
MYSQL_ROW row; // the results rows (array)

struct connection_details mysqlD;

mysqlD.server = "localhost"; // where the mysql database is
mysqlD.user = "phpmyadmin"; // user
mysqlD.password = "dragon3000"; // the password for the database
mysqlD.database = "LIFE_GAME"; // the database
// "select `UNIQUE_CASES_ID` from `UNIQUE_CASES` WHERE
`HASH`='"+temp_hash+"';"
string my_query="";
my_query+="select `HASH`, `DIRECTION` ";
my_query+="from GLIDER_DIRECTIONS ";
my_query+="WHERE 1;";

con = mysql_connection_setup(mysqlD);
res = mysql_perform_query(con, my_query.c_str());
string key_for_map;
int value_for_map;
// cout<<"HERE1"<<endl;
while ((row = mysql_fetch_row(res)) != NULL)
{
    // key_for_map=row['HASH'];
    // value_for_map=atoi(row['DIRECTION']);
    direct[row[0]]=atoi(row[1]);
    // cout<<row['HASH']<<" "<<value_for_map<<endl;
    // printf("%s \n",row[1]);
}
// cout<<"HERE2"<<endl;
mysql_free_result(res);
mysql_close(con);
}
void catch_vicinity_exp_db(int x, int y, vector<int> &directions, vector<int> &coats)
{
    // directions
    // N-E -> 0
    // S-E -> 1
    // S-W -> 2
    // N-W -> 3

    // coats
    // coat_min_x -> 0
    // coat_max_x -> 1
    // coat_min_y -> 2
    // coat_max_y -> 3
    int min_x=-1,max_x=-1,min_y=-1,max_y=-1;
    int coat_min_x,coat_max_x,coat_min_y,coat_max_y;

```

```

map<string,int> direct;
map<string,int>::iterator it;
//cout<<"BEFORE EVERYTHIG DIED"<<endl;
select_hash_db_directions(direct);
//print_map_string_int(direct);
if(x-VICINITY>0)
{
    coat_min_x=x-VICINITY;
}
else
{
    coat_min_x=0;
}
if(y-VICINITY>0)
{
    coat_min_y=y-VICINITY;
}
else
{
    coat_min_y=0;
}
if(x+VICINITY<roww)
{
    coat_max_x=x+VICINITY;
}
else
{
    coat_max_x=roww;
}
if(y+VICINITY<coll)
{
    coat_max_y=y+VICINITY;
}
else
{
    coat_max_y=coll;
}

free(M_reduction);
free(r_reduction);

roww_red=coat_max_x-coat_min_x;
coll_red=coat_max_y-coat_min_y;

M_reduction=(int**)calloc((roww_red),sizeof(int*));
r_reduction=(int*)calloc(((roww_red)*(coll_red)),sizeof(int));

for(int i=0;i<roww_red;i++)
{
    M_reduction[i]=r_reduction+i*coll_red;
}

for(int i=coat_min_x;i<coat_max_x;i++)
{
    for(int u=coat_min_y;u<coat_max_y;u++)

```

```

    {
        M_reduction[i-coat_min_x][u-coat_min_y]=MM[i][u];
    }
}
//print_matrix(MM,roww,coll);
//print_matrix(M_reduction,roww_red,coll_red);
reduction();
//print_matrix(M_reduction,roww_red,coll_red);
string hash_for_test=create_hash_from_array_reduction();
select_db_to_vector_spaceship();
int flag_kill_spaceship=0;
fstream fp;
string where;
for(int i=0;i<spaceship_bank_hash.size();i++)
{
    if(hash_for_test==spaceship_bank_hash[i])
    {
        flag_kill_spaceship=1;
        break;
    }
}

```

```

it = direct.find(hash_for_test);
if(flag_kill_spaceship==1)
{
    if(flag_kill_spaceship==1)
    {
        if (it != direct.end())
        {
            directions.push_back(it->second);
            flag_kill_spaceship=2;
        }
    }
    if(flag_kill_spaceship==1)
    {
        for(int i=coat_min_x;i<coat_max_x;i++)
        {
            for(int u=coat_min_y;u<coat_max_y;u++)
            {
                if(MM[i][u]==1)
                {
                    MM[i][u]=0;
                }
            }
        }
    }
}

```

```

if(flag_kill_spaceship==2)
{
    //cout<<"Kill"<<endl;
    coats.push_back(coat_min_x);
    coats.push_back(coat_max_x);
}

```

```

        coats.push_back(coat_min_y);
        coats.push_back(coat_max_y);
    }
}

int murder_or_mercy(vector<int> directions)
{
    int mercy=0;
    for(int i=0;i<directions.size()-1;i++)
    {
        cout<<i+1<<" "<<directions[i+1]<<endl;
        if(i+1<directions.size() && (directions[i+1]-directions[i]==-1 ||
directions[i+1]-directions[i]==3))
        {
            mercy=1;
            break;
        }
    }
    return mercy;
}

void kill_spaceship(vector<int> coats)
{
    int coat_min_x,coat_max_x,coat_min_y,coat_max_y;
    for(int k=0;k<coats.size();k+=4)
    {
        coat_min_x=coats[k];
        coat_max_x=coats[k+1];
        coat_min_y=coats[k+2];
        coat_max_y=coats[k+3];
        for(int i=coat_min_x;i<coat_max_x;i++)
        {
            for(int u=coat_min_y;u<coat_max_y;u++)
            {
                if(MM[i][u]==1)
                {
                    MM[i][u]=0;
                }
            }
        }
    }
}

void catch_vicinity_experiment()
{
    int n, life_kolvo;
    n=import_matrix_size("experiment1.txt");
    create_memory(n);
    import_matrix("experiment1.txt");
    life_kolvo=count_life_kolvo_MM();
    catch_vicinity(1,1);
}

void ar_create(int n)
{

```



```

FILE* fp=fopen("Interesting_combinations.txt","w");
fclose(fp);
create_memory(n);

//-----
int life_kolvo;
int* ar=(int*)calloc(n,sizeof(int));
int flag=0;
for(int i=0;i<n;i++)
{
    ar[i]=0;
}
//int counter=0;
enum_counter=0;
int w;
int result_bank_proverka, result_dimension_collapse;
int counter_kill;
vector<int> directions;
vector<int> coats;
int mercy;
int E,W,S,N;
vector<int> E_d;
vector<int> W_d;
vector<int> S_d;
vector<int> N_d;
int stop_flight;
int flag_mass[4];
int flag1;
while(flag==0)
{
    counter_kill=0;
    flag=1;
    living.clear();
    cout<<"----- combination num "<<enum_counter<<"-----"<<endl;
    fill_matrix(MM,roww,coll,ar);
    life_kolvo=count_life_kolvo(ar,n);
    w=0;
    int stop=0;
    int result_dimension_collapse=dimension_collapse(roww);
    if(result_dimension_collapse)
    {
        result_bank_proverka = bank_proverka();
        if(result_bank_proverka==1)
        {
            life_hash.push_back(create_hash_from_array());
            life_pop.push_back(life_kolvo);
            life_exp.insert(life_exp.end(), {0,0,0,0});
            while(stop==0)
            {
                life(life_kolvo);
                life_pop.push_back(life_kolvo);
                life_hash.push_back(create_hash_from_array());

                new_life.clear();
                old_life.clear();
            }
        }
    }
}

```

```

living_candidats.clear();
killing_candidats.clear();

if(w<SUSPICIOUS_SECTOR)
{
    if(bank_proverka()==0)
    {
        for(int i=0;i<w;i++)
        {
            bank_hash.push_back(life_hash[i]);
        }
        break;
    }
}
if(life_kolvo<=0)
{
    break;
}
/*if(w%30==0)
{
    E_d.insert(E_d.end(), {1,0,0,0});
    W_d.insert(W_d.end(), {0,1,0,0});
    S_d.insert(S_d.end(), {0,0,1,0});
    N_d.insert(N_d.end(), {0,0,0,1});
    stop_flight=0;
    flag_mass[0]=0;
    flag_mass[1]=0;
    flag_mass[2]=0;
    flag_mass[3]=0;

    flag_mass[0]=flight_step(flag_mass[0], E_d);
    flag_mass[1]=flight_step(flag_mass[1], W_d);
    flag_mass[2]=flight_step(flag_mass[2], S_d);
    flag_mass[3]=flight_step(flag_mass[3], N_d);

    //North
    if(flag_mass[3]==1)
    {
        flag1=0;
        for(int i=1;i<roww-1 && flag1==0;i++)
        {
            for(int u=1;u<coll-1 && flag1==0;u++)
            {
                if(MM[i][u]==1 && flag1==0)
                {
                    //cout<<"N SUS "<<w<<endl;
                    counter_kill++;
                    catch_vicinity(i,u);
                    flag1=1;
                }
            }
        }
    }
}
}

```

```

//South
if(flag_mass[2]==1)
{
    flag1=0;
    for(int i=roww-2;i>0 && flag1==0;i--)
    {
        for(int u=coll-2;u>0 && flag1==0;u--)
        {
            if(MM[i][u]==1 && flag1==0)
            {
                //cout<<"S SUS "<<w<<endl;
                counter_kill++;
                catch_vicinity(i,u);
                flag1=1;
            }
        }
    }
}

//East
if(flag_mass[0]==1)
{
    flag1=0;
    for(int i=coll-2;i>0 && flag1==0;i--)
    {
        for(int u=roww-2;u>0 && flag1==0;u--)
        {
            if(MM[u][i]==1 && flag1==0)
            {
                //cout<<"E SUS "<<w<<endl;
                counter_kill++;
                catch_vicinity(u,i);
                flag1=1;
            }
        }
    }
}

//West
if(flag_mass[1]==1)
{
    flag1=0;
    for(int i=1;i<coll-1 && flag1==0;i++)
    {
        for(int u=1;u<roww-1 && flag1==0;u++)
        {
            if(MM[u][i]==1 && flag1==0)
            {
                //cout<<"W SUS "<<w<<endl;
                counter_kill++;
                catch_vicinity(u,i);
                flag1=1;
            }
        }
    }
}

```

```

    }
}*/

if(w%30==0)
{
    E_d.insert(E_d.end(), {1,0,0,0});
    W_d.insert(W_d.end(), {0,1,0,0});
    S_d.insert(S_d.end(), {0,0,1,0});
    N_d.insert(N_d.end(), {0,0,0,1});
    stop_flight=0;
    flag_mass[0]=0;
    flag_mass[1]=0;
    flag_mass[2]=0;
    flag_mass[3]=0;

    flag_mass[0]=flight_step(flag_mass[0], E_d);
    flag_mass[1]=flight_step(flag_mass[1], W_d);
    flag_mass[2]=flight_step(flag_mass[2], S_d);
    flag_mass[3]=flight_step(flag_mass[3], N_d);

    //North
    if(flag_mass[3]==1)
    {
        //cout<<"N SUS"<<w<<endl;
        //for(int i=1;i<roww-1 && flag1==0;i++)
        for(int i=1;i<2;i++)
        {
            for(int u=1;u<coll-1;u++)
            {
                if(MM[i][u]==1)
                {
                    //cout<<"N SUS "<<w<<endl;
                    counter_kill++;
                    //catch_vicinity(i,u);

                    //cout<<"catch_vicinity_exp start"<<w<<endl;
                    catch_vicinity_exp(i,u,directions,coats);
                    //cout<<"catch_vicinity_exp finish"<<w<<endl;
                    u+=VICINITY;
                }
            }
        }
        //cout<<"directions1"<<endl;
        //print_vec(directions);
        //cout<<"mercy start"<<w<<endl;
        mercy=murder_or_mercy(directions);
        //cout<<"mercy finish"<<w<<endl;
        //cout<<"mercy="<<mercy<<" "<<w<<endl;
        if(mercy==0)
        {
            //cout<<"mercy="<<mercy<<endl;
            //cout<<"kill_spaceship start"<<w<<endl;
            kill_spaceship(coats);
            //cout<<"kill_spaceship finish"<<w<<endl;
        }
    }
}

```

```

        directions.clear();
        coats.clear();
    }

    //South
    if(flag_mass[2]==1)
    {
        //cout<<"S SUS"<<w<<endl;
        for(int i=roww-2;i>roww-3;i--)
        {
            for(int u=coll-2;u>0;u--)
            {
                if(MM[i][u]==1)
                {
                    //cout<<"S SUS "<<w<<endl;
                    counter_kill++;
                    //catch_vicinity(i,u);
                    //cout<<"catch_vicinity_exp start"<<w<<endl;
                    catch_vicinity_exp(i,u,directions,coats);
                    //cout<<"catch_vicinity_exp finish"<<w<<endl;
                    u-=VICINITY;
                }
            }
        }
        //cout<<"directions2"<<endl;
        //print_vec(directions);
        //cout<<"mercy start"<<w<<endl;
        mercy=murder_or_mercy(directions);
        //cout<<"mercy finish"<<w<<endl;
        //cout<<"mercy="<<mercy<<" "<<w<<endl;
        if(mercy==0)
        {
            //cout<<"mercy="<<mercy<<endl;
            //cout<<"kill_spaceship start"<<w<<endl;
            kill_spaceship(coats);
            //cout<<"kill_spaceship finish"<<w<<endl;
        }
        directions.clear();
        coats.clear();
    }

    //East
    if(flag_mass[0]==1)
    {
        //cout<<"E SUS"<<w<<endl;
        for(int i=coll-2;i>coll-3;i--)
        {
            //for(int u=roww-2;u>0;u--)
            for(int u=1;u<roww-1;u++)
            {
                if(MM[u][i]==1)
                {
                    //cout<<"E SUS "<<w<<endl;
                    counter_kill++;
                    //catch_vicinity(u,i);
                }
            }
        }
    }

```

```

        //cout<<"catch_vicinity_exp start"<<w<<endl;
        catch_vicinity_exp(u,i,directions,coats);
        //cout<<"catch_vicinity_exp finish"<<w<<endl;
        u+=VICINITY;
    }
}
//cout<<"directions3"<<endl;
//print_vec(directions);
//cout<<"mercy start"<<w<<endl;
mercy=murder_or_mercy(directions);
//cout<<"mercy finish"<<w<<endl;
//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"mercy="<<mercy<<endl;
    //cout<<"kill_spaceship start"<<w<<endl;
    kill_spaceship(coats);
    //cout<<"kill_spaceship finish"<<w<<endl;
}
directions.clear();
coats.clear();
}

//West
if(flag_mass[1]==1)
{
    //cout<<"W SUS"<<w<<endl;
    for(int i=1;i<2;i++)
    {
        //for(int u=1;u<roww-1;u++)
        for(int u=roww-2;u>0;u--)
        {
            if(MM[u][i]==1)
            {
                //cout<<"u="<<u<<" i="<<i<<endl;
                //cout<<"W SUS "<<w<<endl;
                counter_kill++;
                //catch_vicinity(u,i);
                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp(u,i,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u-=VICINITY;
            }
        }
    }
//cout<<"directions4"<<endl;
//print_vec(directions);
//cout<<"mercy start"<<w<<endl;
mercy=murder_or_mercy(directions);
//cout<<"mercy finish"<<w<<endl;
//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"mercy="<<mercy<<endl;

```

```

        //cout<<"kill_spaceship start"<<w<<endl;
        kill_spaceship(coats);
        //cout<<"kill_spaceship finish"<<w<<endl;
    }
    directions.clear();
    coats.clear();
}

stop=life_proverka_hash(ar, n);
w++;
if(w>WAIT && stop==0)
{
cout<<enum_counter<<"----->>Interesting----->>"<<count
er_kill<<endl;
    for(int j=0;j<SECTOR;j++)
    {
        bank_hash.push_back(life_hash[j]);
    }
    file_print_hash(ar,n,enum_counter,0);
    stop=1;
}
}
life_exp.clear();
life_pop.clear();
life_hash.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;
    }
    enum_counter++;
}
}

void ar_create_with_db(int n)
{
    // cout<<"Here 1 ";
    // FILE* fp=fopen("Interesting_combinations.txt","w");
    // cout<<"Here 1.1";
    // fclose(fp);
    // cout<<"Here 1.2"<<endl;
    // fp=fopen("Interesting_combination_phase.txt","w");

```

```

// cout<<"Here 1.3"<<endl;
// fclose(fp);
// cout<<"Here 1.4"<<endl;
create_memory(n);
//cout<<"Here 2"<<endl;
//-----
int life_kolvo;
int* ar=(int*)calloc(n,sizeof(int));
int flag=0;
for(int i=0;i<n;i++)
{
    ar[i]=0;
}
//int counter=0;
enum_counter=0;
int w;
int result_bank_proverka, result_dimension_collapse;
int counter_kill;
vector<int> directions;
vector<int> coats;
int mercy;
int E,W,S,N;
vector<int> E_d;
vector<int> W_d;
vector<int> S_d;
vector<int> N_d;
int stop_flight;
int flag_mass[4];
int flag1;
time_t start=time(NULL);
time_t middle1=start,middle2;
//cout<<"Here"<<endl;
while(flag==0)
{
    counter_kill=0;
    flag=1;
    living.clear();
    //cout<<"---- combination num "<<enum_counter<<"----";//<<endl;
    fill_matrix(MM,roww,coll,ar);
    life_kolvo=count_life_kolvo(ar,n);
    w=0;
    int stop=0;
    int result_dimension_collapse=dimension_collapse(roww);
    if(result_dimension_collapse)
    {
        result_bank_proverka = bank_proverka_with_db();
        if(result_bank_proverka==1)
        {
            life_hash.push_back(create_hash_from_array());
            life_pop.push_back(life_kolvo);
            life_exp.insert(life_exp.end(), {0,0,0,0});
            while(stop==0)
            {
                //cout<<"-"<<w<<"-"<<endl;
                life(life_kolvo);
            }
        }
    }
}

```



```

life_pop.push_back(life_kolvo);
life_hash.push_back(create_hash_from_array());

new_life.clear();
old_life.clear();
living_candidats.clear();
killing_candidats.clear();

if(w<SUSPICIOUS_SECTOR)
{
    if(bank_proverka_with_db()==0)
    {
        break;
    }
}
if(life_kolvo<=0)
{
    break;
}
// if(enum_counter==87)
// {
//     file_print_phase(w);
// }
if(w%30==0)
{
    E_d.insert(E_d.end(), {1,0,0,0});
    W_d.insert(W_d.end(), {0,1,0,0});
    S_d.insert(S_d.end(), {0,0,1,0});
    N_d.insert(N_d.end(), {0,0,0,1});
    stop_flight=0;
    flag_mass[0]=0;
    flag_mass[1]=0;
    flag_mass[2]=0;
    flag_mass[3]=0;

    flag_mass[0]=flight_step(flag_mass[0], E_d);
    flag_mass[1]=flight_step(flag_mass[1], W_d);
    flag_mass[2]=flight_step(flag_mass[2], S_d);
    flag_mass[3]=flight_step(flag_mass[3], N_d);

    //North
    if(flag_mass[3]==1)
    {
        //cout<<"N SUS"<<w<<endl;
        //for(int i=1;i<roww-1 && flag1==0;i++)
        for(int i=1;i<2;i++)
        {
            for(int u=1;u<coll-1;u++)
            {
                if(MM[i][u]==1)
                {
                    //cout<<"N SUS "<<w<<endl;
                    //catch_vicinity(i,u);

                    //cout<<"catch_vicinity_exp start"<<w<<endl;
                }
            }
        }
    }
}

```

```

        catch_vicinity_exp_db(i,u,directions,coats);
        //cout<<"catch_vicinity_exp finish"<<w<<endl;
        u+=VICINITY;
    }
}
//cout<<"directions1"<<endl;
//cout<<"mercy start"<<w<<endl;
if(directions.size()==0)
{
    mercy=1;
}
else
{
    //print_vec(directions);
    mercy=murder_or_mercy(directions);
}
//cout<<"mercy finish"<<w<<endl;
//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"- kill="<<counter_kill<<" - coats="<<coats.size()/4<<" -
";//<<endl;
    counter_kill+=coats.size()/4;

    //cout<<"----kill "<<counter_kill<<endl;
    //cout<<"mercy="<<mercy<<endl;
    //cout<<"kill_spaceship start"<<w<<endl;
    kill_spaceship(coats);
    //cout<<"kill_spaceship finish"<<w<<endl;
}
directions.clear();
coats.clear();
}

//South
if(flag_mass[2]==1)
{
    //cout<<"S SUS"<<w<<endl;
    for(int i=roww-2;i>roww-3;i--)
    {
        for(int u=coll-2;u>0;u--)
        {
            if(MM[i][u]==1)
            {
                //cout<<"S SUS "<<w<<endl;
                //catch_vicinity(i,u);
                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp_db(i,u,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u-=VICINITY;
            }
        }
    }
}
//cout<<"directions2"<<endl;

```

```

//cout<<"mercy start"<<w<<endl;
if(directions.size()==0)
{
    mercy=1;
}
else
{
    //print_vec(directions);
    mercy=murder_or_mercy(directions);
}
//cout<<"mercy finish"<<w<<endl;
//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"- kill="<<counter_kill<<" - coats="<<coats.size()/4<<" -
";//<<endl;

    counter_kill+=coats.size()/4;
    //cout<<"----kill "<<counter_kill<<endl;
    //cout<<"mercy="<<mercy<<endl;
    //cout<<"kill_spaceship start"<<w<<endl;
    kill_spaceship(coats);
    //cout<<"kill_spaceship finish"<<w<<endl;
}
directions.clear();
coats.clear();
}

//East
if(flag_mass[0]==1)
{
    //cout<<"E SUS"<<w<<endl;
    for(int i=coll-2;i>coll-3;i--)
    {
        //for(int u=roww-2;u>0;u--)
        for(int u=1;u<roww-1;u++)
        {
            if(MM[u][i]==1)
            {
                //cout<<"E SUS "<<w<<endl;
                //catch_vicinity(u,i);
                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp_db(u,i,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u+=VICINITY;
            }
        }
    }
    //cout<<"directions3"<<endl;
    //cout<<"mercy start"<<w<<endl;
    if(directions.size()==0)
    {
        mercy=1;
    }
    else
    {

```

```

        //print_vec(directions);
        mercy=murder_or_mercy(directions);
    }
    //cout<<"mercy finish"<<w<<endl;
    //cout<<"mercy="<<mercy<<" "<<w<<endl;
    if(mercy==0)
    {
        //cout<<"- kill="<<counter_kill<<" - coats="<<coats.size()/4<<" -
";//<<endl;

        counter_kill+=coats.size()/4;
        //cout<<"----kill "<<counter_kill<<endl;
        //cout<<"mercy="<<mercy<<endl;
        //cout<<"kill_spaceship start"<<w<<endl;
        kill_spaceship(coats);
        //cout<<"kill_spaceship finish"<<w<<endl;
    }
    directions.clear();
    coats.clear();
}

//West
if(flag_mass[1]==1)
{
    //cout<<"W SUS"<<w<<endl;
    for(int i=1;i<2;i++)
    {
        //for(int u=1;u<roww-1;u++)
        for(int u=roww-2;u>0;u--)
        {
            if(MM[u][i]==1)
            {
                //cout<<"u="<<u<<" i="<<i<<endl;
                //cout<<"W SUS "<<w<<endl;
                //catch_vicinity(u,i);
                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp_db(u,i,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u-=VICINITY;
            }
        }
    }
    //cout<<"directions4"<<endl;
    //cout<<"mercy start"<<w<<endl;
    if(directions.size()==0)
    {
        mercy=1;
    }
    else
    {
        //print_vec(directions);
        mercy=murder_or_mercy(directions);
    }
    //cout<<"mercy finish"<<w<<endl;
    //cout<<"mercy="<<mercy<<" "<<w<<endl;
    if(mercy==0)

```

```

        {
            //cout<<"- kill="<<counter_kill<<" - coats="<<coats.size()/4<<" -
";//<<endl;
            counter_kill+=coats.size()/4;
            //cout<<"----kill " <<counter_kill<<endl;
            //cout<<"mercy="<<mercy<<endl;
            //cout<<"kill_spaceship start"<<w<<endl;
            kill_spaceship(coats);
            //cout<<"kill_spaceship finish"<<w<<endl;
        }
        directions.clear();
        coats.clear();
    }
}

stop=life_proverka_hash(ar, n);
w++;
if(w>WAIT && stop==0)
{
//cout<<enum_counter<<"----->>Interesting----->>"<<cou
nter_kill;//<<endl;
    insert_hash_db_interesting(life_hash[0],ar,counter_kill);
    //file_print_hash(ar,n,enum_counter,0);
    stop=1;
}
}
}
life_exp.clear();
life_pop.clear();
life_hash.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;
    }
}
if(enum_counter%8000==0)
{
    middle2=time(NULL);
    insert_db_log(middle2-middle1);
    //cout<<" -----Num " <<enum_counter<<" seconds
"<<middle2-middle1;//<<endl;
    middle1=middle2;
    //cout<<" -----Num " <<enum_counter<<" seconds " <<middle-start;//<<endl;
}
}

```

```

        enum_counter++;
    }
    time_t finish=time(NULL);
    cout<<" Time taken in seconds "<<finish-start;//<<endl;
}

void experiment_expen()
{
    //int ar_temp2[9]={1, 1, 1, 1, 1, 1, 1, 0, 1};
    //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(ar_temp2,9);
    //cout<<endl<<stop<<endl;
    //return;

    //life_pop.insert(life_pop.end(), {0,0,5,0,1,0,3,0,4});
    //life_exp.insert(life_exp.end(), {-1, 1, 0, -2, 1, 1, 0, 1, 1, -1, 0, 1, -1, 1, 0, -2, 1, 1, 0, 1, 1,
-1, 0, 1, -1, 1, 0, -2, 1, 1, 0, 1, 1, -1, 0, 1});
    //life_pop.insert(life_pop.end(), {3,3});
    //life_exp.insert(life_exp.end(), {0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0});
    //int stop=life_proverka(ar_temp2,9);
    //cout<<endl<<stop<<endl;
    //-----
    FILE* fp=fopen("Interesting_combinations.txt","w");
    fclose(fp);
    int n,life_kolvo,w;
    int mode=0;
    living.clear();

    //int* ar=(int*)calloc(n,sizeof(int));
    //int ar[25]={0,0,0,0,0, 0,1,0,0,1, 1,0,0,0,0, 1,0,0,0,1, 1,1,1,1,0};
    //int ar[9]={0,0,1, 1,0,1, 1,1,1};

    /*
    int ar_temp[9]={1,0,0, 1,0,1, 1,1,1}; //axial
    int ar_temp2[9]={1,1,1, 1,0,1, 0,0,1}; //180 left
    int ar_temp3[9]={1,1,1, 0,0,1, 0,1,1}; //90 left
    int ar_temp4[9]={1,1,1, 1,0,0, 1,1,0}; // axial 270 left
    int ar_temp5[9]={1,1,1, 1,1,0, 1,1,0}; //error
    */
    //int ar[9]={1,2,3, 4,5,6, 7,8,9};
    //int ar[25]={1,2,3,4,5, 6,7,8,9,10, 11,12,13,14,15, 16,17,18,19,20,
21,22,23,24,25};
    //int ar_test1[16]={0,0,0,0, 0,0,0,1, 0,0,0,1, 1,1,1,0};
    int ar_test2[16]={0,0,0,0, 0,0,0,1, 1,0,1,1, 0,1,1,0};

    /*if(mode==1)
    {

```

```

n=9;
create_memory(n);
fill_matrix(MM,roww,coll,ar);
//print_matrix(MM,roww,coll);

//life_kolvo=count_life_kolvo(ar,n);
}
else if(mode==2)
{
n=import_matrix_size("experiment2.txt");
create_memory(n);
//cout<<"help"<<endl;
import_matrix("experiment2.txt");
//print_deq2(living);
//print_matrix(MM,roww,coll);
//cout<<"roww="<<roww<<" coll="<<coll<<endl;
life_kolvo=count_life_kolvo_MM();
//cout<<"life_kolvo="<<life_kolvo<<endl;
}*/
//bank_hash.push_back(create_hash_from_array());
//create_memory(n);
//fill_matrix(MM,roww,coll,ar_temp5);
//int test=bank_proverka();
//cout<<"test="<<test<<endl;

//print_matrix_test(MM,roww,coll);
//rotate_counterclockwise();
//axial_symmetry();
//print_matrix_test(MM,roww,coll);
int ar[16]={0,0,0,0, 0,0,0,1, 0,0,0,1, 1,1,1,0};
n=16;
create_memory(n);

int stop,counter_test=0,flag=0;
while(counter_test<2)
{
living.clear();
cout<<"here3"<<endl;
fill_matrix(MM,roww,coll,ar);
cout<<"here4"<<endl;
stop=0;
w=0;
life_kolvo=count_life_kolvo(ar,n);
life_pop.push_back(life_kolvo);
life_hash.push_back(create_hash_from_array());
life_exp.insert(life_exp.end(), {0,0,0,0});
cout<<"here5"<<endl;
cout<<"life_kolvo= "<<life_kolvo<<endl;
print_matrix(MM,roww,coll);
while(stop==0)
{
life(life_kolvo);
cout<<"phase "<<w<<endl;
//print_matrix(MM,roww,coll);
file_print_phase(w);

```

```

life_pop.push_back(life_kolvo);
life_hash.push_back(create_hash_from_array());

if(w<SUSPICIOUS_SECTOR)
{
    if(bank_proverka()==0)
    {
        cout<<"SUS"<<endl;
        break;
    }
}

new_life.clear();
old_life.clear();
living_candidats.clear();
killing_candidats.clear();
if(life_kolvo<=0)
{
    cout<<"life_kolvo = "<<life_kolvo<<endl;
    break;
}
stop=life_proverka_hash(ar,n);
if(stop==1)
{
    cout<<"file_print_hash"<<endl;
    file_print_hash(ar,n,counter_test,1);
    break;
}
else if(w>WAIT && stop==0)
{
    cout<<"w ="<<w<<endl;
    for(int j=0;j<SECTOR;j++)
    {
        bank_hash.push_back(life_hash[j]);
    }
    file_print_hash(ar,n,counter_test,1);
    print_vec_string(bank_hash);
    cout<<"size = "<<bank_hash.size()<<endl;
    stop=1;
}
else if(stop==-1)
{
    cout<<"file_print_hash no period"<<endl;
    file_print_hash(ar,n,counter_test,1);
}
w++;
}
life_exp.clear();
life_pop.clear();
life_hash.clear();
create_memory(n);
cout<<"here1"<<endl;
if(flag==0)
{
    for(int u=0;u<n;u++)

```



```

        {
            ar[u]=ar_test2[u];
        }
        flag=1;
    }
    cout<<"here2"<<endl;
    counter_test++;
}
}
void experiment_expen_first_gen()
{
    FILE* fp=fopen("Interesting_combinations.txt","w");
    fclose(fp);
    int n,life_kolvo,w;
    int mode=0;
    int stop;
    living.clear();
    int ar[16]={0,0,0,0, 0,0,1,1, 0,1,0,1, 1,1,0,1};
    //int ar[9]={1,0,1, 0,1,1, 1,1,0};
    n=16;
    create_memory(n);

    //cout<<"here3"<<endl;
    fill_matrix(MM,roww,coll,ar);
    //cout<<"here4"<<endl;
    stop=0;
    w=0;
    life_kolvo=count_life_kolvo(ar,n);
    life_pop.push_back(life_kolvo);
    life_exp.insert(life_exp.end(), {0,0,0,0});
    life_hash.push_back(create_hash_from_array());
    //cout<<"here5"<<endl;
    //cout<<"life_kolvo= "<<life_kolvo<<endl;
    print_matrix(MM,roww,coll);
    while(stop==0)
    {
        life(life_kolvo);
        cout<<"phase "<<w<<endl;
        print_matrix(MM,roww,coll);
        file_print_phase(w);
        cout<<"new_life="<<new_life.size()/2<<" old_life="<<old_life.size()/2<<"
life_kolvo="<<life_kolvo<<endl;
        print_deq2(new_life);
        life_pop.push_back(life_kolvo);
        life_hash.push_back(create_hash_from_array());
        new_life.clear();
        old_life.clear();
        living_candidats.clear();
        killing_candidats.clear();
        if(life_kolvo<=0)
        {
            cout<<"life_kolvo = "<<life_kolvo<<endl;
            break;
        }
    }
    stop=life_proverka_hash(ar,n);
}

```

```

    if(stop==1)
    {
        cout<<"file_print_hash"<<endl;
        file_print_hash(ar,n,1,1);
        break;
    }
    else if(w>WAIT && stop==0)
    {
        cout<<"w ="<<w<<endl;
        for(int j=0;j<SECTOR;j++)
        {
            bank_hash.push_back(life_hash[j]);
        }
        file_print_hash(ar,n,1,1);
        print_vec_string(bank_hash);
        cout<<"size = "<<bank_hash.size()<<endl;
        stop=1;
    }
    else if(stop==-1)
    {
        cout<<"file_print_hash no period"<<endl;
        file_print_hash(ar,n,1,1);
    }
    w++;
}
print_vec(life_pop);
life_exp.clear();
life_pop.clear();
life_hash.clear();
create_memory(n);
}
void experiment_matrix_rotate()
{
    int ar[9]={1,0,1, 0,1,1, 1,1,0};
    int n=9;
    create_memory(n);
    fill_matrix(MM,roww,coll,ar);
    //MM=expansion_E(MM,&rr,roww,coll,1,1,1);
    for(int i=0;i<4;i++)
    {
        virtual_rotateMN_clockwise(i);
    }
}
void experiment_spaceship_collide()
{
    int n, life_kolvo;
    FILE* fp=fopen("Interesting_combination_phase.txt","w");
    fclose(fp);
    int* ar=(int*)calloc(n,sizeof(int));
    int flag=0;
    for(int i=0;i<n;i++)
    {
        ar[i]=0;
    }
}

```

```

}
int counter=0;
int w;
int result_bank_proverka, result_dimension_collapse;
int counter_kill;
int E,W,S,N;
vector<int> E_d;
vector<int> W_d;
vector<int> S_d;
vector<int> N_d;

vector<int> directions;
vector<int> coats;
int stop_flight;
int flag_mass[4];
int flag1;
int mercy;
counter_kill=0;
flag=1;
living.clear();

n=import_matrix_size("experiment1.txt");
create_memory(n);
import_matrix("experiment1.txt");
life_kolvo=count_life_kolvo_MM();

w=0;
int stop=0;
life_hash.push_back(create_hash_from_array());
life_pop.push_back(life_kolvo);
life_exp.insert(life_exp.end(), {0,0,0,0});
while(stop==0)
{
    life(life_kolvo);
    life_pop.push_back(life_kolvo);
    life_hash.push_back(create_hash_from_array());

    new_life.clear();
    old_life.clear();
    living_candidats.clear();
    killing_candidats.clear();

    if(life_kolvo<=0)
    {
        break;
    }
    if(w%30==0)
    {
        E_d.insert(E_d.end(), {1,0,0,0});
        W_d.insert(W_d.end(), {0,1,0,0});
        S_d.insert(S_d.end(), {0,0,1,0});
        N_d.insert(N_d.end(), {0,0,0,1});
        stop_flight=0;
        flag_mass[0]=0;
        flag_mass[1]=0;
    }
}

```

```

flag_mass[2]=0;
flag_mass[3]=0;

flag_mass[0]=flight_step(flag_mass[0], E_d);
flag_mass[1]=flight_step(flag_mass[1], W_d);
flag_mass[2]=flight_step(flag_mass[2], S_d);
flag_mass[3]=flight_step(flag_mass[3], N_d);

//North
if(flag_mass[3]==1)
{
    //cout<<"N SUS"<<w<<endl;
    //for(int i=1;i<roww-1 && flag1==0;i++)
    for(int i=1;i<2;i++)
    {
        for(int u=1;u<coll-1;u++)
        {
            if(MM[i][u]==1)
            {
                //cout<<"N SUS " <<w<<endl;
                counter_kill++;
                //catch_vicinity(i,u);

                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp(i,u,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u+=VICINITY;
            }
        }
    }
    //cout<<"directions1"<<endl;
    //print_vec(directions);
    //cout<<"mercy start"<<w<<endl;
    mercy=murder_or_mercy(directions);
    //cout<<"mercy finish"<<w<<endl;
    //cout<<"mercy="<<mercy<<" "<<w<<endl;
    if(mercy==0)
    {
        //cout<<"mercy="<<mercy<<endl;
        //cout<<"kill_spaceship start"<<w<<endl;
        kill_spaceship(coats);
        //cout<<"kill_spaceship finish"<<w<<endl;
    }
    directions.clear();
    coats.clear();
}

//South
if(flag_mass[2]==1)
{
    //cout<<"S SUS"<<w<<endl;
    for(int i=roww-2;i>roww-3;i--)
    {
        for(int u=coll-2;u>0;u--)
        {

```

```

        if(MM[i][u]==1)
        {
            //cout<<"S SUS "<<w<<endl;
            counter_kill++;
            //catch Vicinity(i,u);
            //cout<<"catch Vicinity_exp start"<<w<<endl;
            catch_Vicinity_exp(i,u,directions,coats);
            //cout<<"catch Vicinity_exp finish"<<w<<endl;
            u-=VICINITY;
        }
    }
}
//cout<<"directions2"<<endl;
//print_vec(directions);
//cout<<"mercy start"<<w<<endl;
mercy=murder_or_mercy(directions);
//cout<<"mercy finish"<<w<<endl;
//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"mercy="<<mercy<<endl;
    //cout<<"kill spaceship start"<<w<<endl;
    kill_spaceship(coats);
    //cout<<"kill spaceship finish"<<w<<endl;
}
directions.clear();
coats.clear();
}

//East
if(flag_mass[0]==1)
{
    //cout<<"E SUS"<<w<<endl;
    for(int i=coll-2;i>coll-3;i--)
    {
        //for(int u=roww-2;u>0;u--)
        for(int u=1;u<roww-1;u++)
        {
            if(MM[u][i]==1)
            {
                //cout<<"E SUS "<<w<<endl;
                counter_kill++;
                //catch Vicinity(u,i);
                //cout<<"catch Vicinity_exp start"<<w<<endl;
                catch_Vicinity_exp(u,i,directions,coats);
                //cout<<"catch Vicinity_exp finish"<<w<<endl;
                u+=VICINITY;
            }
        }
    }
}
//cout<<"directions3"<<endl;
//print_vec(directions);
//cout<<"mercy start"<<w<<endl;
mercy=murder_or_mercy(directions);
//cout<<"mercy finish"<<w<<endl;

```

```

//cout<<"mercy="<<mercy<<" "<<w<<endl;
if(mercy==0)
{
    //cout<<"mercy="<<mercy<<endl;
    //cout<<"kill_spaceship start"<<w<<endl;
    kill_spaceship(coats);
    //cout<<"kill_spaceship finish"<<w<<endl;
}
directions.clear();
coats.clear();
}

//West
if(flag_mass[1]==1)
{
    //cout<<"W SUS"<<w<<endl;
    for(int i=1;i<2;i++)
    {
        //for(int u=1;u<roww-1;u++)
        for(int u=roww-2;u>0;u--)
        {
            if(MM[u][i]==1)
            {
                //cout<<"u="<<u<<" i="<<i<<endl;
                //cout<<"W SUS "<<w<<endl;
                counter_kill++;
                //catch_vicinity(u,i);
                //cout<<"catch_vicinity_exp start"<<w<<endl;
                catch_vicinity_exp(u,i,directions,coats);
                //cout<<"catch_vicinity_exp finish"<<w<<endl;
                u-=VICINITY;
            }
        }
    }
    //cout<<"directions4"<<endl;
    //print_vec(directions);
    //cout<<"mercy start"<<w<<endl;
    mercy=murder_or_mercy(directions);
    //cout<<"mercy finish"<<w<<endl;
    //cout<<"mercy="<<mercy<<" "<<w<<endl;
    if(mercy==0)
    {
        //cout<<"mercy="<<mercy<<endl;
        //cout<<"kill_spaceship start"<<w<<endl;
        kill_spaceship(coats);
        //cout<<"kill_spaceship finish"<<w<<endl;
    }
    directions.clear();
    coats.clear();
}
}

stop=life_proverka_hash(ar, n);
w++;
if(w>WAIT && stop==0)

```

```

    {
cout<<counter<<"----->>Interesting----->>"<<counter_kill
<<endl;
    //file_print_hash(ar,n,1,0);
    stop=1;
    }
    file_print_phase(w);
}
life_exp.clear();
life_pop.clear();
life_hash.clear();
}
void create_spaceship_hash_bank()
{
    string word="",word_pre="spaceship_",word_post=".txt";
    int n, life_kolvo;
    int ar[1]={0};
    //FILE* fp=fopen("spaceship_hash_bank.txt","w");
    // fstream fp;
    // fp.open("spaceship_hash_bank.txt", ios::out);
    // fp.close();
    for(int i=1;i<5;i++)
    {
        living.clear();

        word=word_pre+to_string(i)+word_post;
        n=import_matrix_size_string(word);
        create_memory(n);
        import_matrix_string(word);
        //print_matrix(MM,roww,coll);

        /*copy_matrix();
        reduction();
        print_matrix(M_reduction,roww_red,coll_red);
        fp<<create_hash_from_array_reduction()<<endl;*/

        //fill_matrix(MM,roww,coll,ar);
        int stop=0;
        int w=0;
        life_kolvo=count_life_kolvo_MM();
        life_pop.push_back(life_kolvo);
        life_hash.push_back(create_hash_from_array());
        life_exp.insert(life_exp.end(), {0,0,0,0});
        //print_vec_string(life_hash);
        //print_vec(life_pop);
        //cout<<roww<<" "<<coll<<endl;
        cout<<"life_kolvo= "<<life_kolvo<<endl;
        //print_matrix(MM,roww,coll);
        while(stop==0)
        {
            life(life_kolvo);
            cout<<"phase "<<w<<endl;

```

```

//print_matrix(MM,roww,coll);

copy_matrix();
reduction();
print_matrix(M_reduction,roww_red,coll_red);
//fp<<create_hash_from_array_reduction()<<endl;
spaceship_bank_fill_db();

//file_print_phase(w);
life_pop.push_back(life_kolvo);
life_hash.push_back(create_hash_from_array());

/*if(w<SUSPICIOUS_SECTOR)
{
    if(bank_proverka()==0)
    {
        cout<<"SUS"<<endl;
        break;
    }
}*/

new_life.clear();
old_life.clear();
living_candidats.clear();
killing_candidats.clear();
if(life_kolvo<=0)
{
    cout<<"life_kolvo = "<<life_kolvo<<endl;
    break;
}
stop=life_proverka_hash(ar,n);
if(stop==1)
{
    cout<<"file_print_hash"<<endl;
    //file_print_hash(ar,n,1,1);
    break;
}
else if(w>2 && stop==0)
{
    cout<<"w ="<<w<<endl;
    /*for(int j=0;j<SECTOR;j++)
    {
        bank_hash.push_back(life_hash[j]);
    }*/
    //file_print_hash(ar,n,1,1);
    //print_vec_string(bank_hash);
    //cout<<"size = "<<bank_hash.size()<<endl;
    stop=1;
}
else if(stop==-1)
{
    cout<<"file_print_hash no period"<<endl;
    //file_print_hash(ar,n,1,1);
}
w++;

```



```

    }
    life_exp.clear();
    life_pop.clear();
    life_hash.clear();

    //create_memory(n);

    //cout<<word<<endl;
    //fp_sp.close();
}
//file_to_vector_spaceship();

//fclose(fp);
}

void create_glider_directions_hash_bank()
{
    int n, life_kolvo;
    int ar[1]={0};
    //FILE* fp=fopen("spaceship_hash_bank.txt","w");
    living.clear();

    n=import_matrix_size_string("spaceship_1.txt");
    create_memory(n);
    import_matrix_string("spaceship_1.txt");
    print_matrix(MM,roww,coll);

    /*copy_matrix();
    reduction();
    print_matrix(M_reduction,roww_red,coll_red);
    fp<<create_hash_from_array_reduction()<<endl;*/

    //fill_matrix(MM,roww,coll,ar);
    int stop=0;
    int w=0;
    life_kolvo=count_life_kolvo_MM();
    life_pop.push_back(life_kolvo);
    life_hash.push_back(create_hash_from_array());
    life_exp.insert(life_exp.end(), {0,0,0,0});
    //print_vec_string(life_hash);
    //print_vec(life_pop);
    //cout<<roww<<" "<<coll<<endl;
    cout<<"life_kolvo= "<<life_kolvo<<endl;
    //print_matrix(MM,roww,coll);
    while(stop==0)
    {
        life(life_kolvo);
        cout<<"phase "<<w<<endl;
        //print_matrix(MM,roww,coll);

        copy_matrix();
        reduction();
    }
}

```

```

print_matrix(M_reduction,roww_red,coll_red);
//fp<<create_hash_from_array_redution()<<endl;
glider_directions_bank_fill_db();

//file_print_phase(w);
life_pop.push_back(life_kolvo);
life_hash.push_back(create_hash_from_array());

new_life.clear();
old_life.clear();
living_candidats.clear();
killing_candidats.clear();
if(life_kolvo<=0)
{
    cout<<"life_kolvo = "<<life_kolvo<<endl;
    break;
}
stop=life_proverka_hash(ar,n);
if(stop==1)
{
    cout<<"file_print_hash"<<endl;
    //file_print_hash(ar,n,1,1);
    break;
}
else if(w>2 && stop==0)
{
    cout<<"w ="<<w<<endl;
    /*for(int j=0;j<SECTOR;j++)
    {
        bank_hash.push_back(life_hash[j]);
    }*/
    //file_print_hash(ar,n,1,1);
    //print_vec_string(bank_hash);
    //cout<<"size = "<<bank_hash.size()<<endl;
    stop=1;
}
else if(stop==-1)
{
    cout<<"file_print_hash no period"<<endl;
    //file_print_hash(ar,n,1,1);
}
w++;
}
life_exp.clear();
life_pop.clear();
life_hash.clear();
}
int main()
{
    //cout<<"Hello World! 1 ";
    //cout<<"Hello World! 2 ";
    int result = truncate_db_unique();
    //select_hash_db_directions();

```

```

//cout<<"Hello World! 3 ";
//catch_vicinity_experiment();
//experiment_spaceship_collide();
//create_glider_directions_hash_bank();
//create_spaceship_hash_bank();
//ar_create(EXPERIMENT_SIZE);
ar_create_with_db(EXPERIMENT_SIZE);
//cout<<"Hello World! 4";//<<endl;

// MYSQL *con; // the connection
// MYSQL_RES *res; // the results
// MYSQL_ROW row; // the results rows (array)

// struct connection_details mysqlID;

// mysqlID.server = "localhost"; // where the mysql database is
// mysqlID.user = "phpmyadmin"; // user
// mysqlID.password = "dragon3000"; // the password for the database
// mysqlID.database = "LIFE_GAME"; // the database

// con = mysql_connection_setup(mysqlID);

// // get the results from executing commands
// res = mysql_perform_query(con, "select * from UNIQUE_CASES;");

// std::cout << ("Database Output:\n") << std::endl;

// while ((row = mysql_fetch_row(res)) != NULL){
// // // the below row[] parametes may change depending on the size of the table and
your objective
// // std::cout << row[0] << " | " << row[1] << " | " << row[2] << std::endl << std::endl;
// // }

// // clean up the database result
// mysql_free_result(res);

// // close database connection
// mysql_close(con);
return 0;
}

```

CPP sha2.cpp

```

-----
#include <cstring>
#include <fstream>
#include "sha2.h"

const unsigned int SHA256::sha256_k[64] = //UL = uint32
    {0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
    0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
    0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
    0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,

```

```

0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
0x90beffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2};
const unsigned long long SHA512::sha512_k[80] = //ULL = uint64
{0x428a2f98d728ae22ULL, 0x7137449123ef65cdULL,
0xb5c0fbcfec4d3b2fULL, 0xe9b5dba58189dbbcULL,
0x3956c25bf348b538ULL, 0x59f111f1b605d019ULL,
0x923f82a4af194f9bULL, 0xab1c5ed5da6d8118ULL,
0xd807aa98a3030242ULL, 0x12835b0145706fbeULL,
0x243185be4ee4b28cULL, 0x550c7dc3d5ffb4e2ULL,
0x72be5d74f27b896fULL, 0x80deb1fe3b1696b1ULL,
0x9bdc06a725c71235ULL, 0xc19bf174cf692694ULL,
0xe49b69c19ef14ad2ULL, 0xefbe4786384f25e3ULL,
0x0fc19dc68b8cd5b5ULL, 0x240ca1cc77ac9c65ULL,
0x2de92c6f592b0275ULL, 0x4a7484aa6ea6e483ULL,
0x5cb0a9dcdbd41fbd4ULL, 0x76f988da831153b5ULL,
0x983e5152ee66dfabULL, 0xa831c66d2db43210ULL,
0xb00327c898fb213fULL, 0xbf597fc7beef0ee4ULL,
0xc6e00bf33da88fc2ULL, 0xd5a79147930aa725ULL,
0x06ca6351e003826fULL, 0x142929670a0e6e70ULL,
0x27b70a8546d22ffcULL, 0x2e1b21385c26c926ULL,
0x4d2c6dfc5ac42aedULL, 0x53380d139d95b3dfULL,
0x650a73548baf63deULL, 0x766a0abb3c77b2a8ULL,
0x81c2c92e47edaee6ULL, 0x92722c851482353bULL,
0xa2bfe8a14cf10364ULL, 0xa81a664bbc423001ULL,
0xc24b8b70d0f89791ULL, 0xc76c51a30654be30ULL,
0xd192e819d6ef5218ULL, 0xd69906245565a910ULL,
0xf40e35855771202aULL, 0x106aa07032bbd1b8ULL,
0x19a4c116b8d2d0c8ULL, 0x1e376c085141ab53ULL,
0x2748774cdf8eeb99ULL, 0x34b0bcb5e19b48a8ULL,
0x391c0cb3c5c95a63ULL, 0x4ed8aa4ae3418acbULL,
0x5b9cca4f7763e373ULL, 0x682e6ff3d6b2b8a3ULL,
0x748f82ee5defb2fcULL, 0x78a5636f43172f60ULL,
0x84c87814a1f0ab72ULL, 0x8cc702081a6439ecULL,
0x90beffa23631e28ULL, 0xa4506cebde82bde9ULL,
0xbef9a3f7b2c67915ULL, 0xc67178f2e372532bULL,
0xca273eceea26619cULL, 0xd186b8c721c0c207ULL,
0xead7dd6cde0eb1eULL, 0xf57d4f7fee6ed178ULL,
0x06f067aa72176fbaULL, 0x0a637dc5a2c898a6ULL,
0x113f9804bef90daeULL, 0x1b710b35131c471bULL,
0x28db77f523047d84ULL, 0x32caab7b40c72493ULL,
0x3c9ebe0a15c9bebcULL, 0x431d67c49c100d4cULL,
0x4cc5d4becb3e42b6ULL, 0x597f299cfc657e2aULL,
0x5fcb6fab3ad6faecULL, 0x6c44198c4a475817ULL};

void SHA224::init()
{
    m_h[0]=0xc1059ed8;
    m_h[1]=0x367cd507;

```

```

    m_h[2]=0x3070dd17;
    m_h[3]=0xf70e5939;
    m_h[4]=0xffc00b31;
    m_h[5]=0x68581511;
    m_h[6]=0x64f98fa7;
    m_h[7]=0xbefa4fa4;
    m_len = 0;
    m_tot_len = 0;
}

```

```

void SHA224::update(const unsigned char *message, unsigned int len)
{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;
    tmp_len = SHA224_256_BLOCK_SIZE - m_len;
    rem_len = len < tmp_len ? len : tmp_len;
    memcpy(&m_block[m_len], message, rem_len);
    if (m_len + len < SHA224_256_BLOCK_SIZE) {
        m_len += len;
        return;
    }
    new_len = len - rem_len;
    block_nb = new_len / SHA224_256_BLOCK_SIZE;
    shifted_message = message + rem_len;
    transform(m_block, 1);
    transform(shifted_message, block_nb);
    rem_len = new_len % SHA224_256_BLOCK_SIZE;
    memcpy(m_block, &shifted_message[block_nb << 6], rem_len);
    m_len = rem_len;
    m_tot_len += (block_nb + 1) << 6;
}

```

```

void SHA224::final(unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
    int i;
    block_nb = (1 + ((SHA224_256_BLOCK_SIZE - 9)
        < (m_len % SHA224_256_BLOCK_SIZE)));
    len_b = (m_tot_len + m_len) << 3;
    pm_len = block_nb << 6;
    memset(m_block + m_len, 0, pm_len - m_len);
    m_block[m_len] = 0x80;
    SHA2_UNPACK32(len_b, m_block + pm_len - 4);
    transform(m_block, block_nb);
    for (i = 0 ; i < 7; i++) {
        SHA2_UNPACK32(m_h[i], &digest[i << 2]);
    }
}

```

```

void SHA256::transform(const unsigned char *message, unsigned int block_nb)
{
    uint32 w[64];

```

```

uint32 wv[8];
uint32 t1, t2;
const unsigned char *sub_block;
int i;
int j;
for (i = 0; i < (int) block_nb; i++) {
    sub_block = message + (i << 6);
    for (j = 0; j < 16; j++) {
        SHA2_PACK32(&sub_block[j << 2], &w[j]);
    }
    for (j = 16; j < 64; j++) {
        w[j] = SHA256_F4(w[j - 2]) + w[j - 7] + SHA256_F3(w[j - 15]) + w[j - 16];
    }
    for (j = 0; j < 8; j++) {
        wv[j] = m_h[j];
    }
    for (j = 0; j < 64; j++) {
        t1 = wv[7] + SHA256_F2(wv[4]) + SHA2_CH(wv[4], wv[5], wv[6])
            + sha256_k[j] + w[j];
        t2 = SHA256_F1(wv[0]) + SHA2_MAJ(wv[0], wv[1], wv[2]);
        wv[7] = wv[6];
        wv[6] = wv[5];
        wv[5] = wv[4];
        wv[4] = wv[3] + t1;
        wv[3] = wv[2];
        wv[2] = wv[1];
        wv[1] = wv[0];
        wv[0] = t1 + t2;
    }
    for (j = 0; j < 8; j++) {
        m_h[j] += wv[j];
    }
}
}

```

```
void SHA256::init()
```

```

{
    m_h[0] = 0x6a09e667;
    m_h[1] = 0xbb67ae85;
    m_h[2] = 0x3c6ef372;
    m_h[3] = 0xa54ff53a;
    m_h[4] = 0x510e527f;
    m_h[5] = 0x9b05688c;
    m_h[6] = 0x1f83d9ab;
    m_h[7] = 0x5be0cd19;
    m_len = 0;
    m_tot_len = 0;
}

```

```
void SHA256::update(const unsigned char *message, unsigned int len)
```

```

{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;
    tmp_len = SHA224_256_BLOCK_SIZE - m_len;
}

```

```

rem_len = len < tmp_len ? len : tmp_len;
memcpy(&m_block[m_len], message, rem_len);
if (m_len + len < SHA224_256_BLOCK_SIZE) {
    m_len += len;
    return;
}
new_len = len - rem_len;
block_nb = new_len / SHA224_256_BLOCK_SIZE;
shifted_message = message + rem_len;
transform(m_block, 1);
transform(shifted_message, block_nb);
rem_len = new_len % SHA224_256_BLOCK_SIZE;
memcpy(m_block, &shifted_message[block_nb << 6], rem_len);
m_len = rem_len;
m_tot_len += (block_nb + 1) << 6;
}

```

```

void SHA256::final(unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
    int i;
    block_nb = (1 + ((SHA224_256_BLOCK_SIZE - 9)
        < (m_len % SHA224_256_BLOCK_SIZE)));
    len_b = (m_tot_len + m_len) << 3;
    pm_len = block_nb << 6;
    memset(m_block + m_len, 0, pm_len - m_len);
    m_block[m_len] = 0x80;
    SHA2_UNPACK32(len_b, m_block + pm_len - 4);
    transform(m_block, block_nb);
    for (i = 0 ; i < 8; i++) {
        SHA2_UNPACK32(m_h[i], &digest[i << 2]);
    }
}

```

```

void SHA384::init()
{
    m_h[0] = 0xcbbb9d5dc1059ed8ULL;
    m_h[1] = 0x629a292a367cd507ULL;
    m_h[2] = 0x9159015a3070dd17ULL;
    m_h[3] = 0x152fec8f70e5939ULL;
    m_h[4] = 0x67332667ffc00b31ULL;
    m_h[5] = 0x8eb44a8768581511ULL;
    m_h[6] = 0xdb0c2e0d64f98fa7ULL;
    m_h[7] = 0x47b5481dbefa4fa4ULL;
    m_len = 0;
    m_tot_len = 0;
}

```

```

void SHA384::update(const unsigned char *message, unsigned int len)
{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;

```

```

tmp_len = SHA384_512_BLOCK_SIZE - m_len;
rem_len = len < tmp_len ? len : tmp_len;
memcpy(&m_block[m_len], message, rem_len);
if (m_len + len < SHA384_512_BLOCK_SIZE) {
    m_len += len;
    return;
}
new_len = len - rem_len;
block_nb = new_len / SHA384_512_BLOCK_SIZE;
shifted_message = message + rem_len;
transform(m_block, 1);
transform(shifted_message, block_nb);
rem_len = new_len % SHA384_512_BLOCK_SIZE;
memcpy(m_block, &shifted_message[block_nb << 7], rem_len);
m_len = rem_len;
m_tot_len += (block_nb + 1) << 7;
}

void SHA384::final(unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
    int i;
    block_nb = (1 + ((SHA384_512_BLOCK_SIZE - 17)
        < (m_tot_len % SHA384_512_BLOCK_SIZE)));
    len_b = (m_tot_len + m_len) << 3;
    pm_len = block_nb << 7;
    memset(m_block + m_len, 0, pm_len - m_len);
    m_block[m_len] = 0x80;
    SHA2_UNPACK32(len_b, m_block + pm_len - 4);
    transform(m_block, block_nb);
    for (i = 0 ; i < 6; i++) {
        SHA2_UNPACK64(m_h[i], &digest[i << 3]);
    }
}

void SHA512::transform(const unsigned char *message, unsigned int block_nb)
{
    uint64 w[80];
    uint64 wv[8];
    uint64 t1, t2;
    const unsigned char *sub_block;
    int i, j;
    for (i = 0; i < (int) block_nb; i++) {
        sub_block = message + (i << 7);
        for (j = 0; j < 16; j++) {
            SHA2_PACK64(&sub_block[j << 3], &w[j]);
        }
        for (j = 16; j < 80; j++) {
            w[j] = SHA512_F4(w[j - 2]) + w[j - 7] + SHA512_F3(w[j - 15]) + w[j - 16];
        }
        for (j = 0; j < 8; j++) {
            wv[j] = m_h[j];
        }
    }
}

```



```

    for (j = 0; j < 80; j++) {
        t1 = wv[7] + SHA512_F2(wv[4]) + SHA2_CH(wv[4], wv[5], wv[6])
            + sha512_k[j] + w[j];
        t2 = SHA512_F1(wv[0]) + SHA2_MAJ(wv[0], wv[1], wv[2]);
        wv[7] = wv[6];
        wv[6] = wv[5];
        wv[5] = wv[4];
        wv[4] = wv[3] + t1;
        wv[3] = wv[2];
        wv[2] = wv[1];
        wv[1] = wv[0];
        wv[0] = t1 + t2;
    }
    for (j = 0; j < 8; j++) {
        m_h[j] += wv[j];
    }
}
}

```

```

void SHA512::init()
{
    m_h[0] = 0x6a09e667f3bcc908ULL;
    m_h[1] = 0xbb67ae8584caa73bULL;
    m_h[2] = 0x3c6ef372fe94f82bULL;
    m_h[3] = 0xa54ff53a5f1d36f1ULL;
    m_h[4] = 0x510e527fade682d1ULL;
    m_h[5] = 0x9b05688c2b3e6c1fULL;
    m_h[6] = 0x1f83d9abfb41bd6bULL;
    m_h[7] = 0x5be0cd19137e2179ULL;
    m_len = 0;
    m_tot_len = 0;
}

```

```

void SHA512::update(const unsigned char *message, unsigned int len)
{
    unsigned int block_nb;
    unsigned int new_len, rem_len, tmp_len;
    const unsigned char *shifted_message;
    tmp_len = SHA384_512_BLOCK_SIZE - m_len;
    rem_len = len < tmp_len ? len : tmp_len;
    memcpy(&m_block[m_len], message, rem_len);
    if (m_len + len < SHA384_512_BLOCK_SIZE) {
        m_len += len;
        return;
    }
    new_len = len - rem_len;
    block_nb = new_len / SHA384_512_BLOCK_SIZE;
    shifted_message = message + rem_len;
    transform(m_block, 1);
    transform(shifted_message, block_nb);
    rem_len = new_len % SHA384_512_BLOCK_SIZE;
    memcpy(m_block, &shifted_message[block_nb << 7], rem_len);
    m_len = rem_len;
    m_tot_len += (block_nb + 1) << 7;
}

```

```

}

void SHA512::final(unsigned char *digest)
{
    unsigned int block_nb;
    unsigned int pm_len;
    unsigned int len_b;
    int i;
    block_nb = 1 + ((SHA384_512_BLOCK_SIZE - 17)
        < (m_len % SHA384_512_BLOCK_SIZE));
    len_b = (m_tot_len + m_len) << 3;
    pm_len = block_nb << 7;
    memset(m_block + m_len, 0, pm_len - m_len);
    m_block[m_len] = 0x80;
    SHA2_UNPACK32(len_b, m_block + pm_len - 4);
    transform(m_block, block_nb);
    for (i = 0 ; i < 8; i++) {
        SHA2_UNPACK64(m_h[i], &digest[i << 3]);
    }
}

std::string sha224(std::string input)
{
    unsigned char digest[SHA224::DIGEST_SIZE];
    memset(digest,0,SHA224::DIGEST_SIZE);
    SHA224 ctx = SHA224();
    ctx.init();
    ctx.update((unsigned char*)input.c_str(), input.length());
    ctx.final(digest);

    char buf[2*SHA224::DIGEST_SIZE+1];
    buf[2*SHA224::DIGEST_SIZE] = 0;
    for (int i = 0; i < SHA224::DIGEST_SIZE; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    return std::string(buf);
}

std::string sha256(std::string input)
{
    unsigned char digest[SHA256::DIGEST_SIZE];
    memset(digest,0,SHA256::DIGEST_SIZE);

    SHA256 ctx = SHA256();
    ctx.init();
    ctx.update( (unsigned char*)input.c_str(), input.length());
    ctx.final(digest);

    char buf[2*SHA256::DIGEST_SIZE+1];
    buf[2*SHA256::DIGEST_SIZE] = 0;
    for (int i = 0; i < SHA256::DIGEST_SIZE; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    return std::string(buf);
}

std::string sha384(std::string input)

```

```

{
    unsigned char digest[SHA384::DIGEST_SIZE];
    memset(digest,0,SHA384::DIGEST_SIZE);
    SHA384 ctx = SHA384();
    ctx.init();
    ctx.update((unsigned char*)input.c_str(), input.length());
    ctx.final(digest);

    char buf[2*SHA384::DIGEST_SIZE+1];
    buf[2*SHA384::DIGEST_SIZE] = 0;
    for (int i = 0; i < SHA384::DIGEST_SIZE; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    return std::string(buf);
}

```

std::string sha512(std::string input)

```

{
    unsigned char digest[SHA512::DIGEST_SIZE];
    memset(digest,0,SHA512::DIGEST_SIZE);
    SHA512 ctx = SHA512();
    ctx.init();
    ctx.update((unsigned char*)input.c_str(), input.length());
    ctx.final(digest);

    char buf[2*SHA512::DIGEST_SIZE+1];
    buf[2*SHA512::DIGEST_SIZE] = 0;
    for (int i = 0; i < SHA512::DIGEST_SIZE; i++)
        sprintf(buf+i*2, "%02x", digest[i]);
    return std::string(buf);
}

```

H sha2.h

```

-----
#ifndef SHA2_H
#define SHA2_H
#include <string>

class SHA2
{
public:
    virtual void init() = 0;
    virtual void update(const unsigned char *message, unsigned int len) = 0;
    virtual void final(unsigned char *digest) = 0;

protected:
    typedef unsigned char uint8;
    typedef unsigned int uint32;
    typedef unsigned long long uint64;
};

class SHA256 : public SHA2
{
protected:

```

```

    const static uint32 sha256_k[];
    static const unsigned int SHA224_256_BLOCK_SIZE = (512/8);
public:
    void init();
    void update(const unsigned char *message, unsigned int len);
    void final(unsigned char *digest);
    static const unsigned int DIGEST_SIZE = ( 256 / 8);
protected:
    void transform(const unsigned char *message, unsigned int block_nb);
    unsigned int m_tot_len;
    unsigned int m_len;
    unsigned char m_block[2*SHA224_256_BLOCK_SIZE];
    uint32 m_h[8];
};

```

```

class SHA224 : public SHA256
{
public:
    void init();
    void update(const unsigned char *message, unsigned int len);
    void final(unsigned char *digest);
    static const unsigned int DIGEST_SIZE = ( 224 / 8);
};

```

```

class SHA512 : public SHA2
{
protected:
    const static uint64 sha512_k[];
    static const unsigned int SHA384_512_BLOCK_SIZE = (1024/8);

public:
    void init();
    void update(const unsigned char *message, unsigned int len);
    void final(unsigned char *digest);
    static const unsigned int DIGEST_SIZE = ( 512 / 8);

protected:
    void transform(const unsigned char *message, unsigned int block_nb);
    unsigned int m_tot_len;
    unsigned int m_len;
    unsigned char m_block[2 * SHA384_512_BLOCK_SIZE];
    uint64 m_h[8];
};

```

```

class SHA384 : public SHA512
{
public:
    void init();
    void update(const unsigned char *message, unsigned int len);
    void final(unsigned char *digest);
    static const unsigned int DIGEST_SIZE = ( 384 / 8);
};

```

```
std::string sha224(std::string input);
std::string sha256(std::string input);
std::string sha384(std::string input);
std::string sha512(std::string input);
```

```
#define SHA2_SHFR(x, n)  (x >> n)
#define SHA2_ROTTR(x, n) ((x >> n) | (x << ((sizeof(x) << 3) - n)))
#define SHA2_ROTL(x, n) ((x << n) | (x >> ((sizeof(x) << 3) - n)))
#define SHA2_CH(x, y, z) ((x & y) ^ (~x & z))
#define SHA2_MAJ(x, y, z) ((x & y) ^ (x & z) ^ (y & z))
#define SHA256_F1(x) (SHA2_ROTTR(x, 2) ^ SHA2_ROTTR(x, 13) ^ SHA2_ROTTR(x, 22))
#define SHA256_F2(x) (SHA2_ROTTR(x, 6) ^ SHA2_ROTTR(x, 11) ^ SHA2_ROTTR(x, 25))
#define SHA256_F3(x) (SHA2_ROTTR(x, 7) ^ SHA2_ROTTR(x, 18) ^ SHA2_SHFR(x, 3))
#define SHA256_F4(x) (SHA2_ROTTR(x, 17) ^ SHA2_ROTTR(x, 19) ^ SHA2_SHFR(x, 10))
#define SHA512_F1(x) (SHA2_ROTTR(x, 28) ^ SHA2_ROTTR(x, 34) ^ SHA2_ROTTR(x, 39))
#define SHA512_F2(x) (SHA2_ROTTR(x, 14) ^ SHA2_ROTTR(x, 18) ^ SHA2_ROTTR(x, 41))
#define SHA512_F3(x) (SHA2_ROTTR(x, 1) ^ SHA2_ROTTR(x, 8) ^ SHA2_SHFR(x, 7))
#define SHA512_F4(x) (SHA2_ROTTR(x, 19) ^ SHA2_ROTTR(x, 61) ^ SHA2_SHFR(x, 6))
#define SHA2_UNPACK32(x, str) \
{ \
    *((str) + 3) = (uint8) ((x) >> 24); \
    *((str) + 2) = (uint8) ((x) >> 16); \
    *((str) + 1) = (uint8) ((x) >> 8); \
    *((str) + 0) = (uint8) ((x) >> 0); \
}
#define SHA2_PACK32(str, x) \
{ \
    *(x) = ((uint32) *((str) + 3) << 24) \
    | ((uint32) *((str) + 2) << 16) \
    | ((uint32) *((str) + 1) << 8) \
    | ((uint32) *((str) + 0) << 0); \
}
#define SHA2_UNPACK64(x, str) \
{ \
    *((str) + 7) = (uint8) ((x) >> 56); \
    *((str) + 6) = (uint8) ((x) >> 48); \
    *((str) + 5) = (uint8) ((x) >> 40); \
    *((str) + 4) = (uint8) ((x) >> 32); \
    *((str) + 3) = (uint8) ((x) >> 24); \
    *((str) + 2) = (uint8) ((x) >> 16); \
    *((str) + 1) = (uint8) ((x) >> 8); \
    *((str) + 0) = (uint8) ((x) >> 0); \
}
#define SHA2_PACK64(str, x) \
{ \
    *(x) = ((uint64) *((str) + 7) << 56) \
    | ((uint64) *((str) + 6) << 48) \
    | ((uint64) *((str) + 5) << 40) \
    | ((uint64) *((str) + 4) << 32) \
    | ((uint64) *((str) + 3) << 24) \
    | ((uint64) *((str) + 2) << 16) \
    | ((uint64) *((str) + 1) << 8) \
    | ((uint64) *((str) + 0) << 0); \
}
```

#endif