main

```
#include <iostream>
#include <cmath>
#include <map>
#include <cstdlib>
#include <deque>
#include <vector>
#include "sha2.h"

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;
int life_pop_max;
int roww,coll;
int **MM;
int *rr;
int print_counter = 1;

//wait till period
int WAIT=1000;

//hash constant
int HASH=50;

//initial hash sector
int SECTOR=10;

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

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_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]<<" ";
    }
    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;
            }
        }
    }
}

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 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;
```

```
    int counter=1;
    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 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++;
    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_combinations.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;
}

void rotate_anticlockwise()
{
    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;
        }
    }
}

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 life(int &life_kolvo)
{
    int first,second,alive_sosed,flag;
    int first_new,second_new;
    for(int k=0;k<living.size();k+=2)
    {
        first=living[k];
        second=living[k+1];
        alive_sosed=0;
        for(int i=first-1;i<=first+1;i++)
        {
            for(int u=second-1;u<=second+1;u++)
            {
                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)
    {
        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);
    }
}

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);
            }
        }
    }
  }
}

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

//EXPANSION
      int max_x,max_y,min_x,min_y;
      int flag1=0, 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;
      }
   }
}

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);
            }
        }

//BINARY INSERT
for(int i=0;i<living_candidats.size();i+=2)
{
    insert_binary_search(living_candidats[i],living_candidats[i+1],living);
}

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;
    }
}

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);
        }
    }

    for(int i=killing_candidats.size()-1;i>=0;i--)
    {
        living.erase(living.begin()+killing_candidats[i]);
        living.erase(living.begin()+killing_candidats[i]);
    }
    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 counter=0;
    for(int i=1;i<n-1;i++)
    {
        for(int j=1;j<n-1;j++)
        {
            if(MM[i][j]==1)
            {
                counter++;
                break;
            }
        }
    }
    if(counter==n-2)
    {
        return 1;
```

```
}
counter=0;
for(int j=1;j<n-1;j++)
{
   for(int i=1;i<n-1;i++)
   {
      if(MM[i][j]==1)
      {
         counter++;
         break;
      }
   }
}
if(counter==n-2)
{
   return 1;
}
return 0;*/
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=create_hash_from_array();
            //print_matrix(MM,roww,coll);
            //printf("\n%s\n",temp_hash);
            //cout<<temp_hash<<endl;
            if(serch_in_bank(temp_hash))
            {
                return 0;
            }
            rotate_anticlockwise();
        }
        //cout<<"<<<<<<<<<<<<<<<<<<<<<<<<"<<endl;
        //print_matrix(MM,roww,coll);
        axial_symmetry();
        //print_matrix(MM,roww,coll);
        //cout<<"<<<<<<<<<<<<<<<<<<<<<<<<"<<endl;
        for(int i=0;i<4;i++)
        {
            temp_hash=create_hash_from_array();
            //print_matrix(MM,roww,coll);
            //printf("\n%s\n",temp_hash);
            //cout<<temp_hash<<endl;
            if(serch_in_bank(temp_hash))
            {
                return 0;
            }
            rotate_anticlockwise();
        }
        //cout<<">>>>>>>>>>>>>>>>>>>>>>>>"<<endl;
```

```
        //print_matrix(MM,roww,coll);
        axial_symmetry();
        //print_matrix(MM,roww,coll);
        return 1;
    }
    return 1;
}

int life_proverka_hash(int* ar, int n)
{
    string last_element=life_hash[life_hash.size()-1];
    int flag;
    int simptom=0;
    int number_of_periods=HASH;
    int nop;
    int period_length;

    for(int i=life_hash.size()-1;i>=0;i--)
    {
        if(i!=life_hash.size()-1 && life_hash[i]==last_element && simptom==0)
        {
            if(life_hash.size()-1-i>life_hash.size()/2)
            {
                simptom=-1;
            }
            else
            {
                flag=0;
                nop=1;
                while(nop<=number_of_periods)
                {
                    for(int u=i-(life_hash.size()-1-i)*nop+1;u<i-(life_hash.size()-1-i)*(nop-1);u++)
                    {
                        if(u<0 || life_hash[u]!=life_hash[u+life_hash.size()-1-i])
                        {
                            flag=1;
                            break;
                        }
                    }
                    nop++;
                }
                if(flag==0)
                {
                    period_length=life_hash.size()-1-i;
                    cout<<" period_length="<<period_length<<endl;
                    simptom=1;
                }
            }
        }
        if(life_pop[i]>life_pop_max)
        {
            life_pop_max=life_pop[i];
        }
    }
    //if(simptom==-1)
```

```
    //{
    //    file_print_hash(ar,n);
    //}
    return simptom;
}

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;
    int w;
    int result_bank_proverka;
    while(flag==0)
    {
        flag=1;
        living.clear();
        cout<<"----- combination num "<<counter<<"-----"<<endl;
        print_init_matrix(ar,n);
        fill_matrix(MM,roww,coll,ar);
        life_kolvo=count_life_kolvo(ar,n);
        w=0;
        int stop=0;
        result_bank_proverka = bank_proverka();
        cout<<"result_bank_proverka="<<result_bank_proverka<<endl;
        //print_vec_string(bank_hash);
        if(dimension_collapse(roww))
        {
            if(result_bank_proverka==1)
            {
                life_hash.push_back(create_hash_from_array());
                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;
                    }
                    stop=life_proverka_hash(ar, n);
```

```
                w++;
                if(w>WAIT && stop==0)
                {
                    cout<<"---------------------------------------------->>Interesting"<<endl;
                    //bank_hash.push_back(create_hash_from_array());
                    for(int j=0;j<SECTOR;j++)
                    {
                        bank_hash.push_back(life_hash[j]);
                    }
                    file_print_hash(ar,n,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;
        }
    }

    counter++;
  }
}
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;
}
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 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=2;
  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[16]={1,2,3,4,5,   6,7,8,9,10,     11,12,13,14,15,     16};

    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_anticlockwise();
    //axial_symmetry();
    //print_matrix_test(MM,roww,coll);
    w=0;
    int stop=0;

    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());
        new_life.clear();
        old_life.clear();
        living_candidats.clear();
        killing_candidats.clear();
        //cout<<"w="<<w<<" life_kolvo="<<life_kolvo<<endl;
        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);
            break;
        }
        else if(w>1000 && stop==0)
        {
            cout<<"w ="<<w<<endl;
            file_print_hash(ar,n,1);
            stop=1;
        }
        else if(stop==-1)
        {
            cout<<"file_print_hash no period"<<endl;
            file_print_hash(ar,n,1);
        }
        w++;
    }
    life_exp.clear();
    life_pop.clear();
    life_hash.clear();
    create_memory(n);
}
//0 0 1 0 1 0 1 1 1
int main()
{

    //life();
    ar_create(16);
    //experiment_expen();
    /*string input = "grape";
    string output1 = sha224(input);
    string output2 = sha256(input);
    string output3 = sha384(input);
    string output4 = sha512(input);

    cout << "sha224('"<< input << "'):" << output1 << endl;
    cout << "sha256('"<< input << "'):" << output2 << endl;
    cout << "sha384('"<< input << "'):" << output3 << endl;
    cout << "sha512('"<< input << "'):" << output4 << endl;*/
```

```
    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,
             0x90befffa, 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,
             0x5cb0a9dcbd41fbd4ULL, 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,
        0x90befffa23631e28ULL, 0xa4506cebde82bde9ULL,
        0xbef9a3f7b2c67915ULL, 0xc67178f2e372532bULL,
        0xca273eceea26619cULL, 0xd186b8c721c0c207ULL,
        0xeada7dd6cde0eb1eULL, 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] = 0x152fecd8f70e5939ULL;
    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_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_ROTR(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_ROTR(x,  2) ^ SHA2_ROTR(x, 13) ^ SHA2_ROTR(x, 22))
#define SHA256_F2(x) (SHA2_ROTR(x,  6) ^ SHA2_ROTR(x, 11) ^ SHA2_ROTR(x, 25))
#define SHA256_F3(x) (SHA2_ROTR(x,  7) ^ SHA2_ROTR(x, 18) ^ SHA2_SHFR(x,  3))
#define SHA256_F4(x) (SHA2_ROTR(x, 17) ^ SHA2_ROTR(x, 19) ^ SHA2_SHFR(x, 10))
#define SHA512_F1(x) (SHA2_ROTR(x, 28) ^ SHA2_ROTR(x, 34) ^ SHA2_ROTR(x, 39))
#define SHA512_F2(x) (SHA2_ROTR(x, 14) ^ SHA2_ROTR(x, 18) ^ SHA2_ROTR(x, 41))
#define SHA512_F3(x) (SHA2_ROTR(x,  1) ^ SHA2_ROTR(x,  8) ^ SHA2_SHFR(x,  7))
#define SHA512_F4(x) (SHA2_ROTR(x, 19) ^ SHA2_ROTR(x, 61) ^ SHA2_SHFR(x,  6))
#define SHA2_UNPACK32(x, str)                 \
{                                             \
    *((str) + 3) = (uint8) ((x)      );       \
    *((str) + 2) = (uint8) ((x) >>  8);       \
    *((str) + 1) = (uint8) ((x) >> 16);       \
    *((str) + 0) = (uint8) ((x) >> 24);       \
}
#define SHA2_PACK32(str, x)                   \
{                                             \
    *(x) =   ((uint32) *((str) + 3)      )    \
           | ((uint32) *((str) + 2) <<  8)    \
           | ((uint32) *((str) + 1) << 16)    \
           | ((uint32) *((str) + 0) << 24);   \
}
#define SHA2_UNPACK64(x, str)                 \
{                                             \
    *((str) + 7) = (uint8) ((x)      );       \
```

```
    *((str) + 6) = (uint8) ((x) >>  8);       \
    *((str) + 5) = (uint8) ((x) >> 16);       \
    *((str) + 4) = (uint8) ((x) >> 24);       \
    *((str) + 3) = (uint8) ((x) >> 32);       \
    *((str) + 2) = (uint8) ((x) >> 40);       \
    *((str) + 1) = (uint8) ((x) >> 48);       \
    *((str) + 0) = (uint8) ((x) >> 56);       \
}
#define SHA2_PACK64(str, x)                   \
{                                             \
    *(x) =   ((uint64) *((str) + 7)      )    \
           | ((uint64) *((str) + 6) <<  8)    \
           | ((uint64) *((str) + 5) << 16)    \
           | ((uint64) *((str) + 4) << 24)    \
           | ((uint64) *((str) + 3) << 32)    \
           | ((uint64) *((str) + 2) << 40)    \
           | ((uint64) *((str) + 1) << 48)    \
           | ((uint64) *((str) + 0) << 56);   \
}

#endif
```