P1
The probability that the last two people can take their own seats are 1/3. Let's assume the drunk man takes #1 room by change, then it's clear all the rest of the people will have the correct seats. If he takes #49 or #50, then one of the last two people will not get their seat, one of them will seat at #1. The probabilities that the drunk man seat at #1, #49, #50 are equal. So the probabilities that the last two people can take their own seats are 1/3. Otherwise let's assume the drunk man takes the n-th seat, where n is a number between 2 and 48, Everyone between 2 and (n-1) will get his own seat. That means the n-th person essentially becomes the new "drunk" guy with designated seat #1. We could continue the previous analysis. Since at all jump points there's an 2/3 chance for the "drunk" guy to choose seat #49,#50 1/3 change choose his own seat. The probability that the last two people can take their own seats are 1/3.

P7
Stability: Numerical errors which are generated during the solution of discretized equations should not be magnified
Explicit methods calculate the state of an ordinary or partial differential equations system at a later time using information from the state of the system at current time. While for an implicit method, we need to solve an equation containing information from both current and later time to get the state of system at later time.

P13 Implement the Sudoko algorithm

```
#include<iostream>
#include<ostream>
#include<istream>

using namespace std;

class Sudoku{
public:
  //constructor;
  Sudoku();
  void Print();
  bool Solve();
  void setBoardValue(int nx, int ny, int value );

private:
```

```
  int board[9][9];
  bool Solve(int nx, int ny);
  bool verify(int nx, int ny);
};

Sudoku::Sudoku(){
  for(int i=0; i<9; i++)
    for(int j=0; j<9; j++){
      board[i][j]=0;
    }
}
bool Sudoku::Solve(){
  return Solve(0,0);
}

void Sudoku::setBoardValue(int nx, int ny, int value){
  board[nx][ny]=value;
}

void Sudoku::Print(){
  for(int j=0; j<9; j++){
    if( j%3 == 0 ){
      cout<< "-------------------------------" << endl;
    }
    for(int i=0; i<9; i++){
      if( i%3==0 ){
          cout << "|";
      }
      if(board[i][j] !=0){
          cout<<" "<<board[i][j]<<" ";
      } else {
          cout<<" * ";
      }
    }
    cout<<"|"<<endl;
  }
  cout << "-------------------------------" << endl;
}
```

```
bool Sudoku::Solve(int nx, int ny){
  if( board[nx][ny] != 0 ){
    if(verify(nx, ny)){
      if( nx==8 && ny== 8){
          return true;
      }
      int next_x = nx+1;
      int next_y = ny;

      if(next_x >=9 ){
          next_x = 0;
          next_y++;
      }
      return Solve(next_x, next_y);
    } else{
      return false;
    }
  }

  for (int val=1; val<10; val++){
    setBoardValue(nx, ny, val);
    if(verify(nx,ny)){
      if( nx == 8 && ny==8 ){
          return true;
      }
      int next_x = nx+1;
      int next_y = ny;
      if(next_x>=9){
          next_x = 0;
          next_y++;
      }
      if(Solve(next_x,next_y)){
          return true;
      }
    }
  }

  board[nx][ny] = 0;
  return false;
}
```

```
bool Sudoku::verify(int nx , int ny ){
  int i, j;

  for(i=nx - (nx%3) ; i< nx-(nx%3) + 3;i++ )
    for(j=ny - (ny%3) ; j<ny-(ny%3)+ 3; j++){
      if ( i==nx && j==ny ) continue;
      if (board[i][j] == board[nx][ny] ) return false;
    }
  for(i=0; i<9; i++){
    if( i == nx ) continue;
    if( board[i][ny] == board[nx][ny] ) return false;
  }
  for(j=0; j<9; j++){
    if( j==ny ) continue;
    if( board[nx][j] == board[nx][ny] ) return false;
  }
  return true;
}


int main(){

  Sudoku puzzle;

  puzzle.setBoardValue(0,0,2);
  puzzle.setBoardValue(1,1,5);
  puzzle.setBoardValue(2,2,9);

  puzzle.setBoardValue(3,2,6);
  puzzle.setBoardValue(4,1,2);
  puzzle.setBoardValue(5,0,7);

  puzzle.setBoardValue(6,2,5);
  puzzle.setBoardValue(7,0,9);
  puzzle.setBoardValue(8,1,8);

  puzzle.setBoardValue(0,5,6);
  puzzle.setBoardValue(1,4,7);
```

```
  puzzle.setBoardValue(2,3,5);

  puzzle.setBoardValue(3,3,3);
  puzzle.setBoardValue(4,4,8);
  puzzle.setBoardValue(5,5,4);

  puzzle.setBoardValue(6,3,9);
  puzzle.setBoardValue(8,4,2);

  puzzle.setBoardValue(0,6,3);
  puzzle.setBoardValue(1,7,4);
  puzzle.setBoardValue(2,8,7);

  puzzle.setBoardValue(6,8,3);
  puzzle.setBoardValue(7,6,1);
  puzzle.setBoardValue(8,7,7);

  puzzle.Print();

  cout << endl;

  if(puzzle.Solve()){
    cout<<"Found solution:"<<endl;
    puzzle.Print();
  } else {
    cout <<"Puzzle is ill posed!";
  }
  cout<<endl;
  return 0;
}


#include<iostream>
#include<ostream>
#include<istream>

using namespace std;

class Sudoku{
public:
```

```
  //constructor;
  Sudoku();
  void Print();
  bool Solve();
  void setBoardValue(int nx, int ny, int value );

private:
  int board[9][9];
  bool Solve(int nx, int ny);
  bool verify(int nx, int ny);
};

Sudoku::Sudoku(){
  for(int i=0; i<9; i++)
    for(int j=0; j<9; j++){
      board[i][j]=0;
    }
}
bool Sudoku::Solve(){
  return Solve(0,0);
}

void Sudoku::setBoardValue(int nx, int ny, int value){
  board[nx][ny]=value;
}

void Sudoku::Print(){
  for(int j=0; j<9; j++){
    if( j%3 == 0 ){
      cout<< "-----------------------------" << endl;
    }
    for(int i=0; i<9; i++){
      if( i%3==0 ){
          cout << "|";
      }
      if(board[i][j] !=0){
          cout<<" "<<board[i][j]<<" ";
      } else {
          cout<<" * ";
      }
```

```
    }
    cout<<"|"<<endl;
  }
  cout << "-------------------------------" << endl;
}


bool Sudoku::Solve(int nx, int ny){
  if( board[nx][ny] != 0 ){
    if(verify(nx, ny)){
      if( nx==8 && ny== 8){
          return true;
      }
      int next_x = nx+1;
      int next_y = ny;

      if(next_x >=9 ){
          next_x = 0;
          next_y++;
      }
      return Solve(next_x, next_y);
    } else{
      return false;
    }
  }

  for (int val=1; val<10; val++){
    setBoardValue(nx, ny, val);
    if(verify(nx,ny)){
      if( nx == 8 && ny==8 ){
          return true;
      }
      int next_x = nx+1;
      int next_y = ny;
      if(next_x>=9){
          next_x = 0;
          next_y++;
      }
      if(Solve(next_x,next_y)){
          return true;
```

```
      }
    }
  }

  board[nx][ny] = 0;
  return false;
}


bool Sudoku::verify(int nx , int ny ){
  int i, j;

  for(i=nx - (nx%3) ; i< nx-(nx%3) + 3;i++ )
    for(j=ny - (ny%3) ; j<ny-(ny%3)+ 3; j++){
      if ( i==nx && j==ny ) continue;
      if (board[i][j] == board[nx][ny] ) return false;
    }
  for(i=0; i<9; i++){
    if( i == nx ) continue;
    if( board[i][ny] == board[nx][ny] ) return false;
  }
  for(j=0; j<9; j++){
    if( j==ny ) continue;
    if( board[nx][j] == board[nx][ny] ) return false;
  }
  return true;
}


int main(){

  Sudoku puzzle;

  puzzle.setBoardValue(0,0,2);
  puzzle.setBoardValue(1,1,5);
  puzzle.setBoardValue(2,2,9);

  puzzle.setBoardValue(3,2,6);
  puzzle.setBoardValue(4,1,2);
  puzzle.setBoardValue(5,0,7);
```

```
  puzzle.setBoardValue(6,2,5);
  puzzle.setBoardValue(7,0,9);
  puzzle.setBoardValue(8,1,8);

  puzzle.setBoardValue(0,5,6);
  puzzle.setBoardValue(1,4,7);
  puzzle.setBoardValue(2,3,5);

  puzzle.setBoardValue(3,3,3);
  puzzle.setBoardValue(4,4,8);
  puzzle.setBoardValue(5,5,4);

  puzzle.setBoardValue(6,3,9);
  puzzle.setBoardValue(8,4,2);

  puzzle.setBoardValue(0,6,3);
  puzzle.setBoardValue(1,7,4);
  puzzle.setBoardValue(2,8,7);

  puzzle.setBoardValue(6,8,3);
  puzzle.setBoardValue(7,6,1);
  puzzle.setBoardValue(8,7,7);

  puzzle.Print();

  cout << endl;

  if(puzzle.Solve()){
    cout<<"Found solution:"<<endl;
    puzzle.Print();
  } else {
    cout <<"Puzzle is ill posed!";
  }
  cout<<endl;
  return 0;
}
```

```
#include<iostream>
#include<ostream>
#include<istream>

using namespace std;

class Sudoku{
public:
  //constructor;
  Sudoku();
  void Print();
  bool Solve();
  void setBoardValue(int nx, int ny, int value );

private:
  int board[9][9];
  bool Solve(int nx, int ny);
  bool verify(int nx, int ny);
};

Sudoku::Sudoku(){
  for(int i=0; i<9; i++)
    for(int j=0; j<9; j++){
      board[i][j]=0;
    }
}
bool Sudoku::Solve(){
  return Solve(0,0);
}

void Sudoku::setBoardValue(int nx, int ny, int value){
  board[nx][ny]=value;
}

void Sudoku::Print(){
  for(int j=0; j<9; j++){
    if( j%3 == 0 ){
      cout<< "-------------------------------" << endl;
    }
```

```
    for(int i=0; i<9; i++){
      if( i%3==0 ){
          cout << "|";
      }
      if(board[i][j] !=0){
          cout<<" "<<board[i][j]<<" ";
      } else {
          cout<<" * ";
      }
    }
    cout<<"|"<<endl;
  }
  cout << "-------------------------------" << endl;
}


bool Sudoku::Solve(int nx, int ny){
  if( board[nx][ny] != 0 ){
    if(verify(nx, ny)){
      if( nx==8 && ny== 8){
          return true;
      }
      int next_x = nx+1;
      int next_y = ny;

      if(next_x >=9 ){
          next_x = 0;
          next_y++;
      }
      return Solve(next_x, next_y);
    } else{
      return false;
    }
  }

  for (int val=1; val<10; val++){
    setBoardValue(nx, ny, val);
    if(verify(nx,ny)){
      if( nx == 8 && ny==8 ){
          return true;
```

```
      }
      int next_x = nx+1;
      int next_y = ny;
      if(next_x>=9){
          next_x = 0;
          next_y++;
      }
      if(Solve(next_x,next_y)){
          return true;
      }
    }
  }

  board[nx][ny] = 0;
  return false;
}


bool Sudoku::verify(int nx , int ny ){
  int i, j;

  for(i=nx - (nx%3) ; i< nx-(nx%3) + 3;i++ )
    for(j=ny - (ny%3) ; j<ny-(ny%3)+ 3; j++){
      if ( i==nx && j==ny ) continue;
      if (board[i][j] == board[nx][ny] ) return false;
    }
  for(i=0; i<9; i++){
    if( i == nx ) continue;
    if( board[i][ny] == board[nx][ny] ) return false;
  }
  for(j=0; j<9; j++){
    if( j==ny ) continue;
    if( board[nx][j] == board[nx][ny] ) return false;
  }
  return true;
}


int main(){
```

```
Sudoku puzzle;

puzzle.setBoardValue(0,0,2);
puzzle.setBoardValue(1,1,5);
puzzle.setBoardValue(2,2,9);

puzzle.setBoardValue(3,2,6);
puzzle.setBoardValue(4,1,2);
puzzle.setBoardValue(5,0,7);

puzzle.setBoardValue(6,2,5);
puzzle.setBoardValue(7,0,9);
puzzle.setBoardValue(8,1,8);

puzzle.setBoardValue(0,5,6);
puzzle.setBoardValue(1,4,7);
puzzle.setBoardValue(2,3,5);

puzzle.setBoardValue(3,3,3);
puzzle.setBoardValue(4,4,8);
puzzle.setBoardValue(5,5,4);

puzzle.setBoardValue(6,3,9);
puzzle.setBoardValue(8,4,2);

puzzle.setBoardValue(0,6,3);
puzzle.setBoardValue(1,7,4);
puzzle.setBoardValue(2,8,7);

puzzle.setBoardValue(6,8,3);
puzzle.setBoardValue(7,6,1);
puzzle.setBoardValue(8,7,7);

puzzle.Print();

cout << endl;

if(puzzle.Solve()){
  cout<<"Found solution:"<<endl;
  puzzle.Print();
```

```
  } else {
    cout <<"Puzzle is ill posed!";
  }
  cout<<endl;
  return 0;
}
```