CS 4613

Sudoku Project

Santino Ricatto

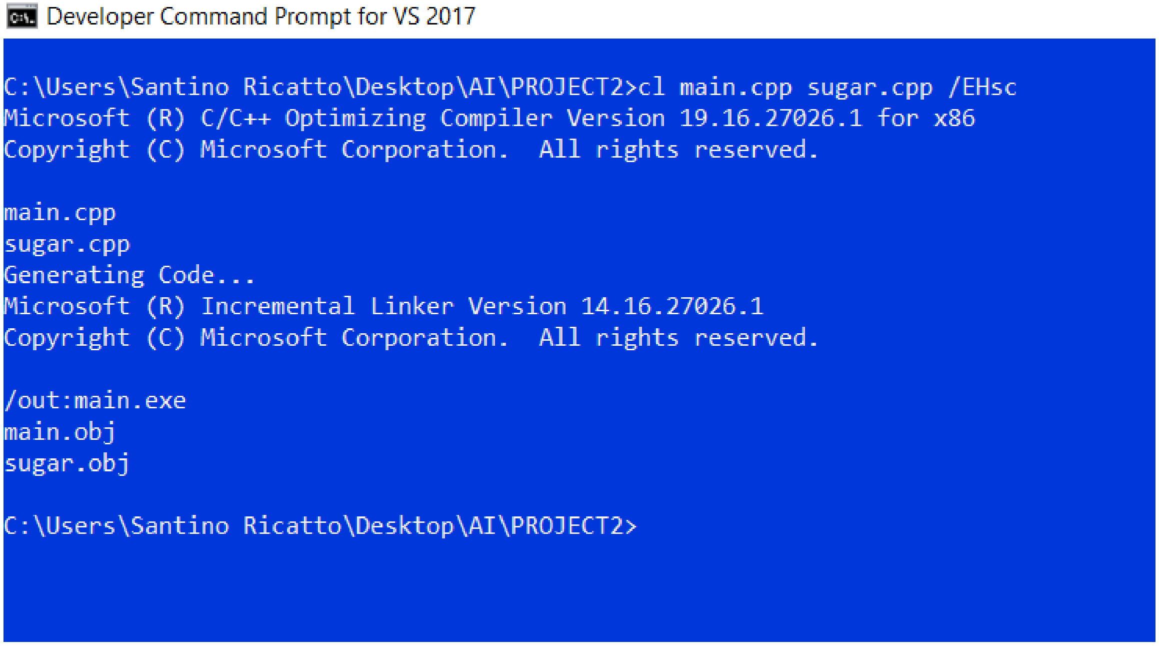
Spring 2019

Contents

Table of Contents

HOW TO COMPILE

Simply use the command “cl main.cpp sugar.cpp /EHsc” in the developer command prompt in Windows.
Note: Code may not compile on other systems, I have only tested windows.
The output should be main.exe
See below:

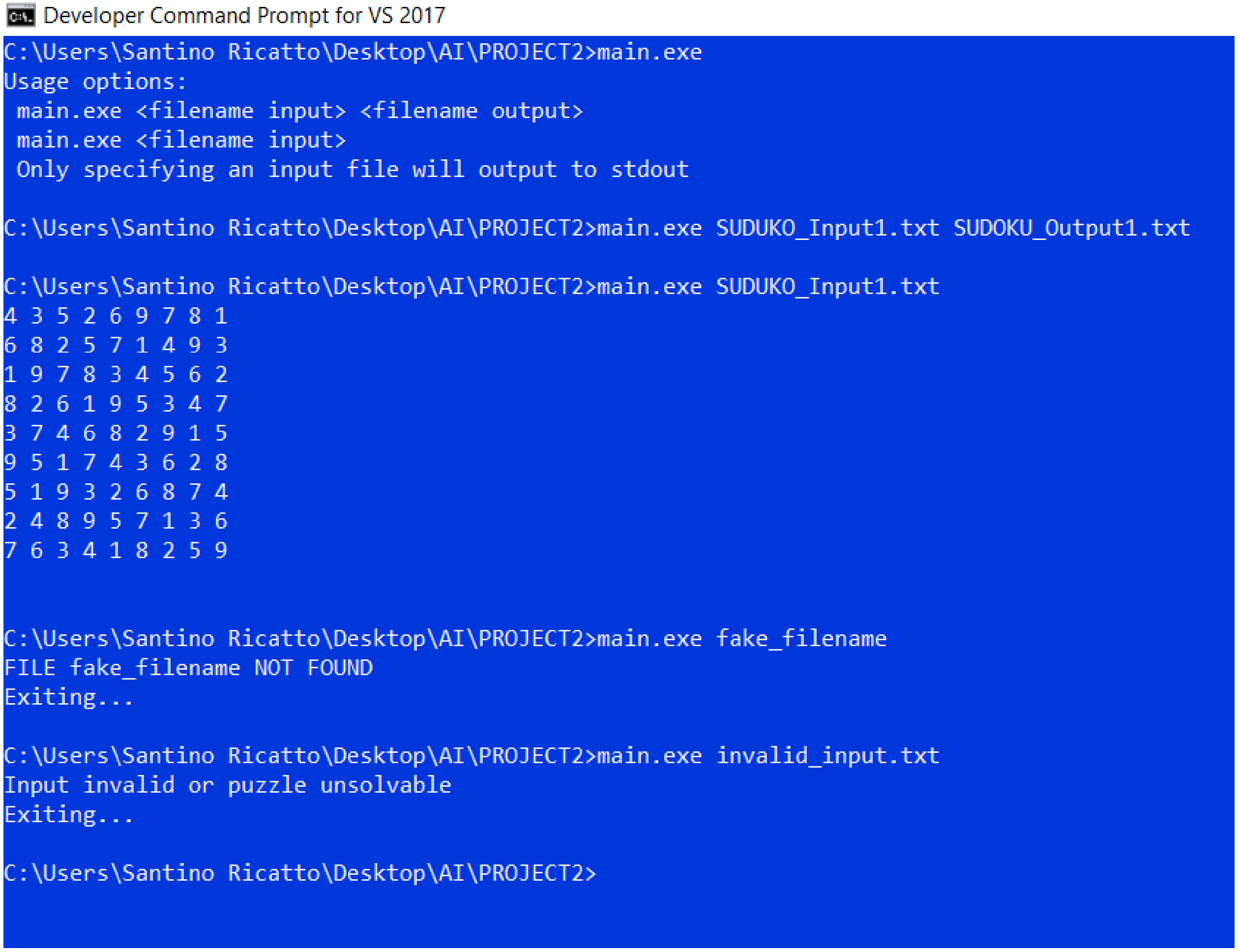


Developer Command Prompt for VS 2017

```
C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>cl main.cpp sugar.cpp /EHsc
Microsoft (R) C/C++ Optimizing Compiler Version 19.16.27026.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

main.cpp
sugar.cpp
Generating Code...
Microsoft (R) Incremental Linker Version 14.16.27026.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:main.exe
main.obj
sugar.obj

C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>
```

HOW TO RUN

Main.exe is a command line utility.
Usage:
        main.exe <input filename> <output filename>
If no output file is specified, the output will be printed to the screen
If it can’t find a file, it’ll say so.
If it is given an unsolvable puzzle, it’ll say so.
See below:



```
C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>main.exe
Usage options:
 main.exe <filename input> <filename output>
 main.exe <filename input>
 Only specifying an input file will output to stdout

C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>main.exe SUDUKO_Input1.txt SUDOKU_Output1.txt

C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>main.exe SUDUKO_Input1.txt
4 3 5 2 6 9 7 8 1
6 8 2 5 7 1 4 9 3
1 9 7 8 3 4 5 6 2
8 2 6 1 9 5 3 4 7
3 7 4 6 8 2 9 1 5
9 5 1 7 4 3 6 2 8
5 1 9 3 2 6 8 7 4
2 4 8 9 5 7 1 3 6
7 6 3 4 1 8 2 5 9


C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>main.exe fake_filename
FILE fake_filename NOT FOUND
Exiting...

C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>main.exe invalid_input.txt
Input invalid or puzzle unsolvable
Exiting...

C:\Users\Santino Ricatto\Desktop\AI\PROJECT2>
```

THE SOURCE CODE

Sugar.h

```
#ifndef tyuityuityuityuityuifghjfghjfghjvbnvbnmvnbm
#define tyuityuityuityuityuifghjfghjfghjvbnvbnmvnbm

//these #define are what I used to iterate over the neighbor
set or domain set of a num
#define decode(a) decoder decodername(a);for(int
i=decodername.first();i!=-1;i=decodername.next())
#define neighborSet(a) allNeighbors decodername(a);for(int
i=decodername.first();i!=-1;i=decodername.next())

class decoder{
   int domain;
   public:
   decoder(int in);
   int first();
   int next();
};
struct HSHLL{
   int mine;
   HSHLL* next;
};
class HashLL{
   HSHLL* first;
   public:
   HashLL();
   void push(int neue);
   bool exists();
   int pop();
};
class allNeighbors{
   HashLL neighbors;
   public:
   allNeighbors(int in);
   int first();
   int next();
};

#endif
```

Info

The #define statements defining decode(a) and neighborSet(a) correspond to the classes defined below.

These are essentially iterators.

To store the values of allNeighbors for neighborSet(a) I used a linked list that I called HSHLL

The text in orange prevents the compiler from accessing this file more than once.

Sugar.cpp

```
//This is a syntactical sugar thing
//to make code more legible
#include "sugar.h"
#include <iostream>
#include <string>
#include <vector>
#include <fstream>

decoder::decoder(int in){domain=in;}
//decoder will output items from the domain in an iterable
fasion.
//not implemented with the standardized iterable.
int decoder::first(){
    return this->next();
}
int decoder::next(){
    if (domain==0)
        return -1;
    for (int i = 0; i<9; i++){
        if (domain & (1<<i)){ //if i'th bit exists (starts at 1 goes to
256 aka 9)
            domain = domain ^ (1<<i);//remove a bit
            return i+1; //return that i existed
        }
    }
    return -1;
}
```

Sugar.cpp

The first par of sugar.cpp is the implementation of the decoder.

The decoder outputs the domain of a num in an iterable fashion.

It gets the domain as a binary item in the form of an int.

decoder::first() and decoder::next() remove one item from the domain and return it.

Sugar.cpp

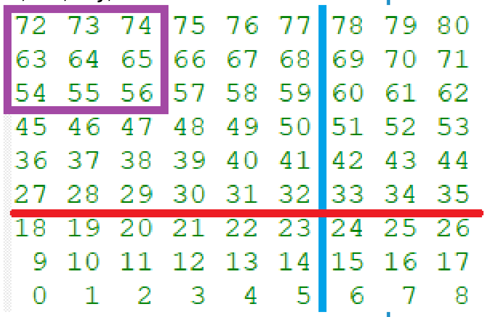
This next part of sugar.cpp is the implementation of a linked list that the allNeighbors class uses to generate output for the neighborSet(a) iterator.

This code is adapted from part of my project 1 code and as such it has a weird name.

The implementation for allNeighbors is below.

```
//Code for an int linked list stolen from my project 1 code.
//We're generating the neighbors every time we access. Might make more optimal later.

  HashLL::HashLL(){
    first = new HSHLL();
    first->mine = -1;
    first->next = NULL;
  }
  void HashLL::push(int neue){
    HSHLL* guy = new HSHLL;
    guy->mine = neue;
    guy->next = first;
    first = guy;
    return;
  }
  bool HashLL::exists(){
    return first!= NULL;
  }

  int HashLL::pop(){
    HSHLL* guy = first;
    int ret = first->mine;
    first = first->next;
    delete guy;
    return ret;
  }
```

```
allNeighbors::allNeighbors(int in){
   //from "in" generate all neighbors push into HashLL
   neighbors = HashLL();
   //Verticals
   for(int i=0;i<9;i++)
      neighbors.push(  (in%9)+(i*9)  );
   //Horizontals
   int ndexs[]={80,71,62,53,44,35,26,17, 8};
   int k;
   for(k=0;k<9;k++)
      if(in<=ndexs[8-k])
         goto there;
   there:;
   for(int i=0;i<9;i++)
      neighbors.push(  ndexs[8-k] - i

   // 3x3 excludes duplicates.
   int b,n,m,f,j;
   k=ndexs[8-k]-8;
   for(b=0;b<9;b++)
      if ((9*b)==k)
         break;

   if(b%3==0){
      f=9;j=18;
   }
   else if(b%3==1){
      f=9;j=-9;
   }
   else if(b%3==2){
      f=-9;j=-18;
   }

   if(in%3==0){
```



```
72 73 74 75 76 77 78 79 80
63 64 65 66 67 68 69 70 71
54 55 56 57 58 59 60 61 62
45 46 47 48 49 50 51 52 53
36 37 38 39 40 41 42 43 44
27 28 29 30 31 32 33 34 35
18 19 20 21 22 23 24 25 26
 9 10 11 12 13 14 15 16 17
 0  1  2  3  4  5  6  7  8
```

Sugar.cpp

An allNeighbors instance outputs the neighbor locations of a num in an iterable fashion.

It gets the location of the num as an int.

Verticals:
in % 9 gets the bottom
adding 9 gets the item on top of that.

Horizontals:
adds 9 to 0 until it is larger than in
then adds one 9 times

3X3:
This gets the four items that the vertical and horizontal missed.

:

```
        n=in+1;m=in+2;
```

```
    }
    else if(in%3==1){
        n=in+1;m=in-1;
    }
    else if(in%3==2){
        n=in-1;m=in-2;
    }

    neighbors.push(n+f);
    neighbors.push(m+f);
    neighbors.push(n+j);
    neighbors.push(m+j);
} //End of allNeighbors init




int allNeighbors::first(){
    return this->next();
}
int allNeighbors::next(){
    if (neighbors.exists())
        return neighbors.pop();
    return -1;
}
```

Sugar.cpp

The top is the tail end of allNeighbors::allNeighbors(int in)

The bottom functions make the class iterable.

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <bitset>
#include "sugar.h"
using namespace std;


// Everything above this line is global constants and includes
/*
Sugar.h info:

Adds the decode(int) loop
essentially decodes a num
into ints accessable by the
name i, an int.
*/
```

main.cpp

This is the top of main.cpp

Gives a tl;dr of sugar.h

Main.cpp
Class num

Each item in the state is a num

From here down only important notes are made.

Important comments will be highlighted green.

```
class num{
    unsigned int self;
    // A bitwise representation of a number.
    //bits [31-19] Unused
    //bits [9-17] Domain Set 1-9
    //bits [0-8] Number 1-9
    public:
    num(){
        self = 0b111111111000000000; //init with domain
123456789
    }
    num(int neue){
        self = ((neue==0)?0b111111111000000000:(1 << neue-
1));
    }
    num(int old, bool spec){
        self = old;
    }

int getNum() const {
    //Returns a numerical representation of num
    //If domain only or failure returns 0;
        if (self == 0)return 0;
        if (self > 256)return 0;
        if (self<=256){
            for(int i =0;i<9;i++){
                if (  (self & (1<<i) ) != 0 )
                    return i+1;
            }
        }
        return -90210;
    }
    void setNum(int neue){
        self = ((neue==0)?0b111111111000000000:(1 << neue-
1));
}
```

```
unsigned int getInt() const {return self;} //Returns the unmodified "self"
num& becomeDomain(){ // shift Domain to Final
   self = self >> 9;
   return *this;
}
bool isComplete()const{
   return ( (self<512) && (self > 0) ); //returns if a num has a final value
}
num& operator-= (const num& rhs){//Removes a number to the Domain set  bits [9-17]
   int neue = rhs.getNum();
   this->removeFromDomain(neue);
   return *this;
}
int countRemainingDomain()const{ //counts how many things in domain
   if (this->isComplete())
      return 0;
   int count=0;
   decode(self>>9){ //decode iterates for each domain item.
      count++;
   }
   return count;
}
void removeFromDomain(int the){ //removes item from domain
   if( self & (1<<(the+9-1)))
      self-=(1<<(the+9-1));

}
int nextPossibleDomain(){ //removes an item from domain and returns it
   decode(self>>9){
      this->removeFromDomain(i);//remove the first and
      return i;//dont continue loop.
   }
   return -1;
}
```

```
};

ostream& operator<<(ostream& os, const num& n){ //Allows us to pump num instances into
file or stdout
    os<<n.getNum();
    return os;
}

/*
class num info:

a num is a representation of a
single spot in the 9x9 grid
the low 18 bits represent the Domain and Final
Unused:[31-18]  Domain:[17-9]  Final Setting:[8-0]
bits in the domain and final work like this:
987654321 and 9876543231 respectively
so if a domain looks like 010000100
then the domain is 8 and 3.


*/
```

State

State is the problem definition, it also acts as the problem state.

It stores 81 num::num()s in a linear array that represents the sudoku board.

Shown here is the default constructor and the deep copy constructor

```
struct state{
    num* layout;
    state(){
        layout = new num[81] ;
    }
    state(state* old){
        layout = new num[81];
        for(int i=0;i<81;i++){
            layout[i] = num(old->layout[i].getInt(), true);
        }
    }
```

//State::doForwardChecking(int)
Takes a location and updates the domains of all the neighbors of that location.

```
void doForwardChecking(int changedValueLocation){
      neighborSet(changedValueLocation){
         if (!(layout[i].isComplete() )){ //for all neighbors if dont have final value, update
domain            layout[i].removeFromDomain( layout[changedValueLocation].getNum() );
         }
      }
   }
   void globalForwardChecking(){//for all values if completed (has a final value) do forward
checking for his neighbors
      for(int l=0;l<81;l++){
         if (layout[l].isComplete()){
            this->doForwardChecking(l);
         }
      }
   }


bool checkValidityOfDomainItem(int loc, int domItem)const{  //Returns if a domain value is valid in
the current context. This is not needed but it parallels the slide code
      neighborSet(loc){
         if(layout[i].getNum() == domItem && i!=loc){
            return false; //If a value at any neighbor location of  loc is domItem, ret false
         }
      }
      return true;
   }
   num* selectUnassignedVariable()const{ //Using the heuristic gets an unassigned num
      return this->minimumRemainingValueWithDegreeHeuristic();
   }
```

```
 int getDegreeAtLoc(int loc)const{ //returns the number of unassigned neighbors, the
“degree” of a num in the context of the problem
    int count=0;
    neighborSet(loc){ //a loop where i is loc of each neighbor
      if (!(layout[i].isComplete())){
        count++;
      }
    }
    return count;
  }

  int getLocationOf(num* teh)const{ //finds the location as a memory offset from
&layout[0]
    for(int row=72;row>=0;row-=9){
      for(int col=0;col<9;col++){
        if( &layout[row+col] == teh)
          return (row+col);
      }
    }
    return -1;
  }
```

```
num* minimumRemainingValueWithDegreeHeuristic()const{
    num* a = layout;
    for(int I=0;I<81;I++){
      if (!(layout[I].isComplete())){a = &layout[I];break;} //
Force a to be an incomplete value
    }
    for(int I=0;I<81;I++){
      if (!(layout[I].isComplete())){
        if (layout[I].countRemainingDomain() < a-
>countRemainingDomain()) //If mrv I  < mrv a  ; a = I
          a = &layout[I];
        else if (layout[I].countRemainingDomain() == a-
>countRemainingDomain())            //if equal
          if (this->getDegreeAtLoc(I)  > this-
>getDegreeAtLoc(  this->getLocationOf(a)  ) )// and degree I >
degree a
            a = &layout[I];
      }
    }
    return a;
  }

  bool goalTest()const{
    for(int I=0;I<81;I++){
      if (!(layout[I].isComplete())){//if I'th item not complete
        return false;
      }
      neighborSet(I){//check correctness (if bad input this'll
catch it)
        if ((layout[I].getNum() == layout[i].getNum()) && (i !=
I)){
          return false;
        }
      }
    }
    return true;
  }
```

Main.cpp state

Minimum Remaining Value With Degree Heuristic

This is the same as the ORDER-DOMAIN-VALUES function seen on the slides.

The primary difference is that this struct uses forward checking as its INFERENCE and is therefore not “dynamic”
So instead of sorting and keeping a list, I searched for what the first thing on the INFERENCES list would have been.

goalTest checks every possible point of failure and returns if layout is in a goal state.

Technically these run in constant time lol. The code can be wasteful because even the hardest puzzle can be solved in less than 10 seconds.

```
    void removeFromMemory(){ //Frees memory allocated for the state
        delete layout;
        return;
    }


    state* backTrack(); //Backtrack is declared here and defined just above the main function
};
/*
state struct info:
This is the problem definition

*/
ostream& operator<<(ostream&os , const state& st){ //Allows us to chuck a state into a file or stdout
    for(int row=72;row>=0;row-=9){
        for(int col=0;col<9;col++){
            os<<st.layout[row+col]<<" ";
        }
        os<<"\n";
    }
    return os;
}
```

Main.cpp Filehandler

This function is the start of the filehandler code. This function allows us to quickly sanitize the input from the file.

Invalid input like letters and whitespace and stuff will become a negative one rendering the puzzle unsolvable.

```
int convertChar(char a){
    if (a=='0'){
        return 0;
    }
    else if (a=='1'){
        return 1;
    }
    else if (a=='2'){
        return 2;
    }
    else if (a=='3'){
        return 3;
    }
    else if (a=='4'){
        return 4;
    }
    else if (a=='5'){
        return 5;
    }
    else if (a=='6'){
        return 6;
    }
    else if (a=='7'){
        return 7;
    }
    else if (a=='8'){
        return 8;
    }
    else if (a=='9'){
        return 9;
    }
    return -1;
}
```

Main.cpp Filehandler

The filehandler class only handles getting a state from a text file.

If it is passed a filename of a file that does not exist, it'll cry and exit.

```
class FileHandler{ //A file handler for problems
    string filename;
    state * st;
    public:
    FileHandler(string fileName, state* a){
        st = a;
        filename = fileName;
        ifstream file;
        string temp="";
        file.open(fileName);
        if (file.is_open()){
            for(int row=72;row>=0;row-=9){
                getline(file,temp);
                for(int col=0;col<9;col++){
                    a->layout[row+col] =
num(convertChar(temp[col*2]));
                                        //The number at place in file
                }
            }
            file.close();
        }
        else{
            cerr << "FILE "<< fileName <<" NOT
FOUND"<<endl<<"Exiting..."<<endl;
            exit(5);
        }
        a->globalForwardChecking();
    }
};
```

Main.cpp Unit tests

The unit test code has been highlighted in light green.

It is how I tested the implementation of the num and state.

Features have been exhaustively tested.

This is just what was left in the unit testing code when development was done.

It should not be relevant to grading though it has been left in for completeness.

```
void unitTests(){
    //This is how I tested every possible issue.
    //Ignore this.
    int failures = 0;
    num* J;
    num* L = new num();
    cout<<"Starting tests\n";
    for(int i=1;i<10;++i){
        cout<<"Testing with final: "<<i<<"\n";
        J = new num(i);
        cout<<"\tgetNum: "<<J->getNum()<<" should be: "<<i<<endl;
        if(J->getNum() != i) failures++;
        cout<<"\tgetInt: "<<J->getInt()<<" should be: "<<(1<<i-1)<<endl;
        if(J->getInt() != (1<<i-1)) failures++;
        cout<<"\tisComplete: "<<J->isComplete()<<" should be: "<< true <<endl;
        if(J->isComplete() != true) failures++;
        cout<<"\tcountRemainingDomain: "<<J->countRemainingDomain()<<" should be: "<<0<<endl;
        if(J->countRemainingDomain() != 0 ) failures++;
        cout<<"\tnextPossibleDomain: "<<J->nextPossibleDomain()<<" should be: "<<-1<<endl;
        if(J->nextPossibleDomain() != -1) failures++;
        cout<<"END\n\n";
        delete J;
    }
    cout<<"Testing with Domainset: FULL\n";
    cout<<"\tgetNum: "<<L->getNum()<<" should be: "<<0<<endl;
    if(L->getNum() != 0) failures++;
    cout<<"\tgetInt: "<<L->getInt()<<" should be: "<<(0b111111111<<9)<<endl;
    if(L->getInt() != (0b111111111<<9)) failures++;
    cout<<"\tisComplete: "<<L->isComplete()<<" should be: "<< false <<endl;
    if(L->isComplete() != false)failures++;
    cout<<endl;
    cout<<"\tcountRemainingDomain: "<<L->countRemainingDomain()<<" should be:
"<<9<<endl;
    if(L->countRemainingDomain() != 9)failures++;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<1<<endl;

    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<2<<endl;
    *L -= num(3);
    cout<<"\tINFO removed 3 from domain with -=\n";
    cout<<endl;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<4<<endl;
    cout<<"\tcountRemainingDomain: "<<L->countRemainingDomain()<<" should be: "<<(9-
4)<<endl;
    cout<<endl;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<5<<endl;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<6<<endl;
    cout<<endl;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<7<<endl;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<8<<endl;
    cout<<"\tcountRemainingDomain: "<<L->countRemainingDomain()<<" should be:
"<<1<<endl;
    failures+=(L->countRemainingDomain()-1);
    L->becomeDomain();
    cout<<"\tINFO did L->becomeDomain()\n";
    cout<<endl;
    cout<<"\tgetNum: "<<L->getNum()<<" should be: "<<9<<endl;
    if(L->getNum() != 9)failures++;
    cout<<"\tgetInt: "<<L->getInt()<<" should be: "<<(1<<8)<<endl;
    if(L->getInt() != (1<<8))failures++;
    cout<<endl;
    cout<<"\tisComplete: "<<L->isComplete()<<" should be: "<< true <<endl;
    if(!L->isComplete())failures++;
    cout<<"\tcountRemainingDomain: "<<L->countRemainingDomain()<<" should be:
"<<0<<endl;
```

```
    if(L->countRemainingDomain()!=0)failures++;
    cout<<"\tnextPossibleDomain: "<<L->nextPossibleDomain()<<" should be: "<<-1<<endl;
    if(L->nextPossibleDomain()!=-1)failures++;
    cout<<"END of domainSet FULL tests\n\n";


    cout<<"The following assume an input text of SUDOKU_Input1.txt:\n0 0 0 2 6 0 7 0 1\n6
  8 0 0 7 0 0 9 0\n1 9 0 0 0 4 5 0 0\n"
            <<"8 2 0 1 0 0 0 4 0\n"
            <<"0 0 4 6 0 2 9 0 0\n"
            <<"0 5 0 0 0 3 0 2 8\n"
            <<"0 4 0 0 5 0 0 3 6\n"
            <<"7 0 3 0 1 8 0 0 0\n\n";

    state theState;
    FileHandler fh("SUDUKO_Input1.txt", &theState);

    cout<<"Testing forward checking:\n\t";
    cout<<(bitset<32>(theState.layout[37].getInt()))<<" Should be:\n\t";
    cout<<(bitset<32>(0b111111111<<9))<<"\n\t";
    neighborSet(37)
      theState.doForwardChecking(i);
    cout<<(bitset<32>(theState.layout[37].getInt()))<<" Should be:\n\t";
    cout<<(bitset<32>(0b001000101<<9))<<"\n\n";

    cout<<" MRVw/DH:"<<theState.minimumRemainingValueWithDegreeHeuristic()<<"\n";



    unsigned int test;
    cout<<"testing domain traversal and global forward checking. (pt1)\n";
    theState.globalForwardChecking();
    theState.globalForwardChecking();
    theState.globalForwardChecking();
    for(int l=0;l<81;l++){
            test = theState.layout[l].getInt();
            if(theState.layout[l].isComplete()){
                decode(test>>9){
                    cout<<"Failure: \n\tdomain found: "<<i<<"\n";
                    failures++;
                }
            }
            else{
                test = theState.layout[l].countRemainingDomain();
                neighborSet(l){
                    if(theState.layout[i].isComplete()){
                        theState.layout[l].removeFromDomain( theState.layout[i].getNum() );
                    }
                }
                if(theState.layout[l].countRemainingDomain() != test){
                    cout<<"Domain not updated properly at:"<<l<<"\n";
                    failures++;
                }
            }
        }
        cout<<"\t";
        cout<<(bitset<32>(theState.layout[37].getInt()))<<" Should be:\n\t";
        cout<<(bitset<32>(0b001000101<<9))<<"\n\n";

        cout<<theState.layout[79].getInt()<<"\n\n";
        cout<<theState<<endl;

        cout<<"END\nTOTAL NUMBER OF FAILURES: "<<failures<<"\n";
    }
```

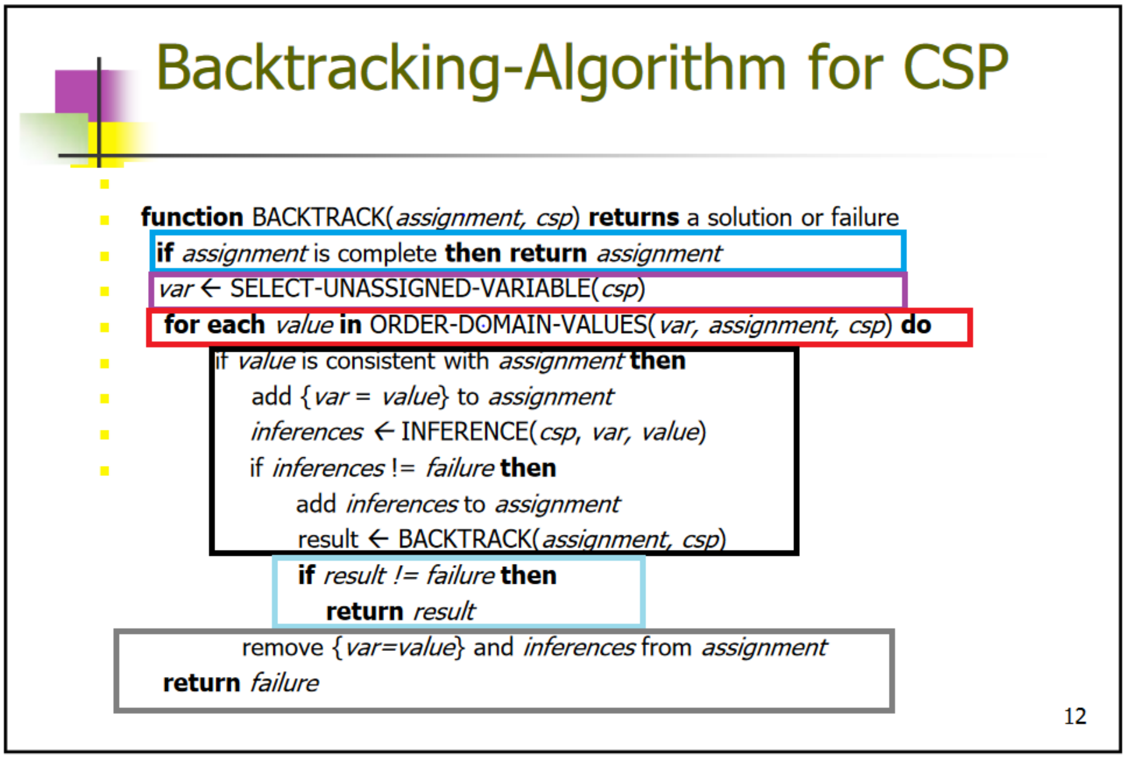
//The only important part is that it'll report how many software errors are detected if it is enabled
(highlighted in blue)

backTrack algorithm implementation

```
state* state::backTrack(){
   if (this->goalTest() == true)
      return this;
   state* result;
   state* duplicate;
   int location;
   num* var = this->selectUnassignedVariable();
   location = this->getLocationOf(var);
   for(int domainItem = var->nextPossibleDomain();
domainItem != -1; domainItem = var->nextPossibleDomain() ){
      if ( this->checkValidityOfDomainItem(location,
domainItem) ){ //If domainItem consistent with assignment
         duplicate = new state(this);
         duplicate->layout[location].setNum(domainItem);
         duplicate->doForwardChecking(location);
         result = duplicate->backTrack();
         if (result != NULL){
            this->removeFromMemory(); //Will remove self from
memory because goal state found and the recursive "I" am  no
longer needed
            return result;
         }
         duplicate->removeFromMemory();//remove from
assignment
      }
   }//nextPossibleDomain will remove the domain item from var
   return NULL; //NULL is failure.
}
```

Main.cpp state:: backtrack()

The colors here are meant to correspond with the slide on the next page adapted from the one shown in class.



Backtracking-Algorithm for CSP

```
function BACKTRACK(assignment, csp) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences != failure then
              add inferences to assignment
              result ← BACKTRACK(assignment, csp)
              if result != failure then
                  return result
          remove {var=value} and inferences from assignment
  return failure
```

Main function

```
int main(int argc, char *argv[]){
   if((argc == 1)||(argc > 3)){
      cout<<"Usage options:\n main.exe <filename input>
<filename output>"
         <<"\n main.exe <filename input>\n Only specifying an
input file will output to stdout\n";
      exit(5);
   }
   state theState;
   state* ans;
   FileHandler fh(argv[1], &theState);
   ans = theState.backTrack();
   if(ans == NULL){
      cout<<"Input invalid or puzzle unsolvable\nExiting...\n";
      exit(0);
   }
   if(argc == 3){
      ofstream Ofile;
      Ofile.open(argv[2]);
      Ofile<< *ans;
   }
   else{
      cout<< *ans << endl;
   }
   exit(0);
}
```

Main.cpp main function

If the program was called with too many or too few parameters, it’ll complain and exit.

It makes an instance of state called theState which becomes the initial state from the input file and a state pointer to point at the goal state that will exist on the heap.

Calling backtrack on theState returns a pointer to the goal state on the heap… or NULL if there was no valid answer.

Then depending on user preference it will print or write to a file the answer.