このプログラムは、9×9の対角線ナンバープレースパズルの問題を.txtファイルから読み込み、解いてその結果を表示するプログラムである。以下にその説明と考察を記す。

説明

[numplace\_main.c]

このファイルはプログラムの中心となる部分である。

ナンバープレースパズル(以後、ナンプレと呼ぶことにする)の答えを記録する表をint型3次元配列tableで宣言する。この3次元の1次元目は、2回目の絞り込み操作をする上で、数値を仮定する際に前の表の状態に戻すために必要な次元である。2次元目と3次元目はそれぞれ表の行と列を表す次元である(ここでは、表の横方向を行、縦方向を列と呼ぶことにする)。

まず、”numplace\_print.h”に含まれるinitialize関数を呼び出し、例外処理をしつつ、コマンドライン引数で指定された問題ファイル(.txt)から問題を読み込んで端末上に表示し、表に書き込む。

次に “numplace\_refine.h”に含まれるsolve関数を呼び出し、各マスの候補となる数値を絞り込み、一定数まで減らしたら、どこか一つのマスについて数値を仮定して、絞り込みを進めるといった操作を繰り返し、 問題を解く(問題設定に不備があり、答えが求められない場合は異常終了する)。

最後に”numplace\_print.h”に含まれるfinalPrintTable関数を呼び出し、答えを端末上に表示して、プログラムを終了する。

[numplace\_print.c]

このファイルは、ナンプレの表示とプログラム実行時初期の例外処理等が含まれた関数が集まったものである。

initialize関数は、int型変数argc、char型変数へのポインタへのポインタ(文字列変数へのポインタ)argvとint型二次元配列変数tableを引数として取る関数である。

char型変数配列bufferは問題ファイルに書き込まれている文字列を一時的に取り込むための変数である。

問題ファイルの指定がなかった場合、コマンドライン引数が多すぎた場合(プログラムを実行する”./numplace”で既にargcは1なので、引数が不正な場合は、3以上の場合しか考えられない)の例外処理を行う。いずれも不正な場合にはそのことを端末上に表示して異常終了する。

その後、ファイルをコマンドライン第2引数で指定された名前で開き、開けなかった場合は異常終了する。

ファイルオープンに成功した場合には、確認のため、問題を表示する。fgets関数で一時的に、問題ファイルの文字列を左から9文字分、上から順に読み込んでいく(fgets関数で指定されている文字数はchar型配列の最後の要素に終端文字'\0'を書き込むために、1文字分余計に指定している)。この時、読み込んだ文字列の最初の文字が';'の場合にはコメントにあたる行なので、読み込まず、そのまま破棄する。一方そうでない場合には、取り込んだ文字を1文字ずつ表示し、数値の場合には文字列をint型の数値に変換してtableに取り込み、空白の場合には、数値が未定の部分として、tableに123456789の数値を代入する(ここに存在している数字がこのマスに入る可能性のある候補を表すものである)。この時、3文字ごとに縦と横の罫線を引くために、ループ変数iを3で割った余りを利用して(今回は余りが0の時)表示する。先頭から数字は行方向に9つだけ並んでいるので、ファイルからは9文字分だけ使用したことになっている。

見やすさのために改行し、ファイルを閉じてこの関数を終了する。

finalPrintTable関数は、解き終わった表を表示するための関数である。

initialize関数の後半部分と同様に、ループ変数の3で割った余りを利用して、縦横の罫線を表示しつつ、左上の数字から順に表示していく。

[numplace\_refine.c]

このファイルにはナンプレの候補を絞り込んで決定する操作や、あるマスの数値を仮定して、候補の絞り込みを進めていく操作などが実装された関数が収録されている。

記憶クラスが”static”に指定されたint型グローバル変数paradoxは0で初期化されて、宣言されている。staticが修飾されているため、このグローバル変数は外部のファイルからアクセスすることはできず、このファイル内からのみ参照することができる。

また、後に宣言される関数rowRefine、columnRefine、squareRefine、diagonalRefine内で、そのさらに後で宣言される関数refineを呼び出す部分があるので、プログラム冒頭でプロトタイプ宣言をしておく。

delete0関数は、int型変数candis(後に記すcanDelete関数で引数として呼ばれるcandisと同様に、table内のあるマスに入る可能性のある数の候補を表す数値)へのポインタを引数に取る関数である。

この関数は、candisが0でない場合に、その変数の一の位が0以外になるまで、candisを10で割り続ける(すなわち、下位の0を消去する)関数である。

canDeleteは、int型変数candis(table上のあるマスの数の候補を表す数値)へのポインタとint型変数x(候補から取り除く数そのもの)を引数に取り、candisの中に含まれるxの値を消去して、残ったcandisの値の範囲によってint型変数の値0または1を返す関数である。

まず、candisの値とxが一致した場合には、グローバル変数paradoxに1だけ足す。

candisの値が10以上の場合(つまり、候補が2つ以上残っている場合)には、candisからxを取り除く操作を実行する(候補が一つ(つまりそのマスの数が確定している)なら、戻り値のみを返して終了)。この操作は、forループ(ループ変数i)でxの値が含まれるcandisの位を探し、それを見つけた場合には、candisからx\*10^iを引く(つまり、xを取り除いてその位を0に変える)といった操作である。現在、この関数では引数としてcandisへのポインタを取っているので、このような操作を実行する際に、candisそのものを操作してしまうと、関数外のcandisそのものが操作されてしまうため、int型変数tmpに一時的にコピーした後に上記のような操作を実行する。そして、candisの中でxがどの位に含まれるのかを正確に特定したうえで、最後にcandisからxを取り除くといった方法を用いている。

この一連の操作が終わった後、candisの下位には0が連続している可能性があるため、見やすくするために、candisから0を取り除く関数delete0を呼び出し、最下位を0以外の整数にしておく。

ここまでの操作でcandisが1以上9以下になった場合(候補が一つに絞り切れて確定した場合)、戻り値として返すansに1だけ足して、ansを返す。これ以外の場合ではansは初期値の0のまま返される。

rowRefineはint型二次元配列変数table、int型変数iとjを引数として取る関数である。

ここでは、候補から一つの数字に確定したマスの行成分(i)と列成分(j)を引数として、(i,j)を除くi行の候補の中から(i,j)の数を取り除く操作をする関数である。

具体的には、forループで列をk=1~9まで動かし、kとjが一致しなければ、canDelete関数で候補の絞り込みを実行する。この時、canDelete関数の戻り値から、候補がただ1つに絞り切れた場合には、refine関数を呼び出し、その(i,k)成分の属するi行、k列、正方領域、(対角線上の成分であれば対角線上)の他のマスの候補から(i,k)の数を取り除く操作をすぐに実行する。refine関数の説明については、後の記述を参照。

columnRefineもrowRefineと同様にj列について(i,j)以外のマスの候補から(i.j)の数を外していく関数である。

squareRefineについても上記二つの関数と同様に、(i,j)のマスの属する正方領域について絞り込みを行う関数である。

正方領域の最も左上のマスの行と列は常に3の倍数であるので、iに最も近いi以下の3の倍数、jに最も近いj以下の3の倍数の位置を、その正方領域の左上で、消去の起点としている。ここから、”kがiと一致しない”かつ”lがjと一致しない”成分を消去することとしている。

rowRefineやcolumnRefineでの不一致条件を考えると、k=iかつl=jの否定なので、k!=iまたはl!=jの否定が妥当だと思われるが、このsquareRefine関数が呼び出されるタイミングは、常にrowRefineやcolumnRefineの後であるから、(i,j)と行や列を共有する成分は既に絞り込み操作を実行済みとなっている。このことから、計算量を減らすという意味でも、k!=iかつk!=lをcanDelete関数の呼び出し条件として設定している。

diagonalRefineは、上記3つの絞り込み関数と同様に、対角線上のマスの候補の数値を絞り込む関数である。しかし、指定されたマスは一般的には常に対角成分であるとは限らないため、呼び出し前には必ず、対角線上のマスかどうかの判定(今回であれば、i=jまたはi+j=8)を行ってから呼び出す。

関数内の操作も、i=jの場合とi+j=8の場合で分けて実行している。i=jの時は、表の左上から右下へ至る対角線上に、i+j=8の時には、表の左下から右上に至る対角線上にそのマスが存在している。よってその該当する対角線上にある別のマスの絞り込み操作を条件に合わせて実行する。

refine関数は行、列、正方領域、対角線上の4つの絞り込みを順に呼び出す関数である。つまり、この関数には、確定したマスの数値とその成分を代入することで、このマスの周りの絞り込みが全て1度にできるということになる。

但し、この絞り込み操作の実行には制限があり、canDelete関数でparadoxが加算された場合については、それぞれの絞り込みは呼び出されない。つまり、あるマスについて仮定をしたうえで絞り込みを実行し矛盾が生じた時には、仮にこれ以上絞り込んだとしても、矛盾が更に増えてしまうので絞り込みを一切行われない。

solveはint型三次元配列tableを引数に取り、問題の初期設定と仮定から未確定マスの候補を絞り込んで解を求める関数である。

pcount関数は、矛盾が生じた際に仮定する前の表の状態に戻すうえで必要な変数である。具体的にはtableの一次元目の現在のポインタ()を記録するための変数であり、仮定をする度にこの変数を一つずつ増やしていく。

まず、最初の二重forループで与えられた初期設定をもとに左上のマスから順に、確定した周りのマスの候補を減らしていく。この時、refine関数内で呼び出されるRefineと名の付く4つの関数内でもrefineが実装されており、候補を絞ってマスの数字が確定した瞬間にすぐに絞り込みが呼び出されるようになっている。これがrefineを冒頭でプロトタイプ宣言した理由に当たる。すぐに絞り込みを実行しない場合、他の確定マス周りの絞り込みで、確定マスを矛盾と認定してしまい、問題が解けなくなってしまう可能性がある。確定したマスに対してすぐに絞り込みをかけることがこのプログラムのポイントの一つである。

次に、絞り込みのみでは確定できないマスに対して、仮定を使って更に候補を絞っていく操作に移る。この操作の終了条件は、”numplace\_check.h”内に含まれる関数、solvedCheck関数によって、問題が解けたと判定されるまで繰り返し行われる。

まず、pcountをインクリメントする。次に、一つ前の表の状態(pcount-1の時)を現在のpcountの時の表にまるごとコピーする(“numplace\_matop.h“内の関数matcpy関数を使う)。更に、pcountの表に対して、”numplace\_matop.h”内の関数、minCandiSerchを使い、候補数が最小のマスを探し、その座標をint型変数iとjに書き込む。その後、そのマスの候補の一番下の位の数字を、そのマスの数字だと仮定して絞り込みを進めていく(最下位の数字は候補の値を10で割った余りに等しい)。

ここでの絞り込みで、矛盾が生じなかった場合には、表の状態がpcount-1の時とpcountの時で変化していることを確認し、ループで戻って、再び数値を仮定する。仮に、pcount-1の時とpcountの時で表の状態が変化していなかったとすれば、その問題はそれ以上候補を絞り込むことができず、問題の初期設定に何らかの不備があることが分かる。この時には、問題が解けないことを端末上に表示して、異常終了する。

絞り込み完了後に矛盾が生じた場合は、矛盾を解消するために、更にループに入る。

まず、一旦表を仮定する前の状態に戻す(すなわちpcountの値を一つ戻す)。この時に、pcountが0を下回ってしまった場合、負の回数だけ仮定したこととなってしまい、これは現実には起こりえない。よって、この場合にも問題設定に不備があり、解けないことを端末に表示して異常終了する。

pcountが0以上の場合には、そのまま、再び候補数が最小の候補を検索し、そのマスの2番目の候補を仮定して絞り込みを進める(前回の最小候補の検索と全く同じ手順なので、必ず前回仮定したマスにたどり着食ことができる)。具体的には候補の値を10で割ることによって、最下位の桁を消去する。

この時、候補の値が0でない(つまり、まだ候補が1つ以上残っている)なら、delete0関数で、末尾の桁の0をあれば消去する。ここで絞り込まれた候補が一つで確定したのなら、またすぐにこのマスの周りに絞り込みをかける。絞り込みが完了した時点で、ループを抜ける。

逆に、最下位の値を消去した際に、候補の値が0となってしまった場合は、候補が一つのマスを消去してしまったこととなり、再び矛盾が生じる。この場合は、再び矛盾解消ループの最初に戻って、pcountが0以上で矛盾が生じ続ける限り、ずっとこのループを繰り返す。

以上のループを抜けた場合には、矛盾が解消されたことを示すために、paradoxを再び0に戻す。この後、pcountの状態の表とpcount-1の状態の表の比較を行い、一致すれば異常終了する。

ここまでの手順を踏んで、仮定のループの先頭に戻り、再び解けたかどうかの判定を行って、解けていない限りは仮定と絞り込みと仮定の取り消しを繰り返して、表を埋めていく。

問題を解くことができ、ループを抜けることができた場合には、その時のpcountの状態の表をpccount=0の場合の表にコピーし、この関数を終了する。

[numplace\_check.c]

このファイルには、ナンプレが解けたかどうかの判定を行う関数が集められている。

rowCheck関数は、i行の数字を左端から順に一つずつ比較していき、一つでも一致したものがあった際に、戻り値1を返すものとなっている。一つでも一致した場合にはそのナンプレは解けていないと判断することができるため、breakですぐにforループを抜けるようになっている。

columnCheck関数も、rowCheck関数と同様に列方向について数字の重複がないかをチェックする関数である。

同様に、squareCheck関数も、diagonalCheck関数も、いずれも正方領域と対角線上の数字の被りがないかをチェックする関数となっている(squareCheck関数内の条件が”かつ”となっている理由も、squareRefineの時と同様、既にrowCheckとcolumnCheckで調べており、計算量を減らすためである)。

refCheck関数は、各マスの候補が全て絞り切れているかを判定する関数で、一つでも候補が2つ以上のマスがあった場合には戻り値として1を返し、すべてのマスがただ一つの数字に絞り切れている場合には、戻り値0を返す。

solvedCheck関数は上記の5つの関数の判定を使って、解けたかどうかを確認する関数である。

5つの関数すべてが0(数字の被りが全ての条件において存在せず、候補もすべて確定されている)の場合には、戻り値として0を、そうでない場合には戻り値として1を返す。

[numplace\_matop.c]

このファイルには、主に表全体にかかわる操作を制御する関数が集められている。

matcpy関数は、int型二次元配列a、bを引数として取り、aの各マスにbの各マスを代入する、すなわち、二次元配列bをaにコピーする関数となっている。

minCandiSearch関数は検索対象となるint型二次元配列の表tableと検索結果の座標を書き込むために必要な、int型変数iへのポインタとjへのポインタを引数として取る。

初期値をとしてtmp=123456789を設定しておき、各マスについてtmpと大小比較を行い、そのマスの候補の値が10以上で(確定していないマスで)tmpよりも小さければ、そのマスの座標をiとjに書き込むといったものである。

matMatch関数はint型二次元配列table1とtable2が一致しているか否かを判定する関数である。

各マスを比較していき、一つでも一致していないマスがあれば、戻り値として1を返し、すべて一致していれば戻り値として0を返す。一つでも一致していないマスを見つけた後にはすぐにループを抜けるような設定になっている点は、check系の関数と同様である。

考察

未確定マスに対する仮定をして絞り込みを実行すると、矛盾が生じた時に戻したりするなどといった無駄な動作が多くなってしまい、非効率である(候補数の少ないマスに対して仮定をして、なるべく仮定を増やさないように工夫はしているが)。そこで、さらなる絞り込みのアルゴリズムとして「消去法型絞り込み」について考えてみる。

例えば、ある行について考えてみると、その行には3か所の未確定箇所があり、その候補は順に、{5,7,9}、{2,5,7}、{2,5,7}といった場合には、1つ目の未確定箇所は9であると決定できる。なぜなら、このマスが9でなければ、この行に9が入るマスがなく、行条件を満たせなくなってしまうからである。

こういった形で、候補同士の類似候補を見比べつつ、どこか一つだけ特徴的な候補が一つだけ存在した際に、そこのマスの数字を確定できるといったアルゴリズムを実装できれば、仮定部分の処理負担の軽減が期待できる。ただしこのアルゴリズムの欠点として、このような場面に遭遇する可能性が他の絞り込み操作に比べて少ないことが考えられる。また、今回のプログラムのような候補の保存方法では、1つ1つの候補を見比べることはできても、複数個の候補の見比べは非常に面倒で効率的に実行することは非常に困難であると推測される。

候補の保存方法にもなんらかの改善の余地が見られそうである。