プログラミングＡレポート（課題２）

担当教員：矢内 直人 教員
提出者：大上 由翔
学籍番号：09B20015

提出年月日：2020年8月6日

1　課題内容

ナンプレ問題を解くプログラムを作成せよ。

条件

- 答えを計算するプログラムであること。
- ナンプレの問題は 9 × 9 の二次元配列を用意し、二次元配列内を任意のデータに書き換えることで、任意の問題に対して動作すること。また、数字の決まっていないマスには、0 を入力する。
- 与えられたナンプレ問題に対して、問題、解答の順に出力すること。

段階

- 段階１…非常に簡単なナンプレ問題を解くプログラムの作成
- 段階２…得られた解がナンプレのルールを満たしているかどうか確認するプログラムの作成
- 段階３…標準的なナンプレ問題を解くプログラムの作成
- 段階４…発展的なナンプレ問題を解くプログラムの作成
- 段階５…その他拡張を行う

今回、提出するプログラムでは段階３まで作成した。つまり、非常に簡単なナンプレ問題と標準的なナンプレ問題を解くことができ、かつ、得られた解がナンプレルールを満たしているかどうか確認するプログラムである。

2　アルゴリズム

本プログラムは主に、ナンプレ問題の解を得る部分とその解を検討する部分から構成されている。プログラムの全体像は図 1 に示す。全ての処理をフローチャートに示すと煩雑になりかねないため、適宜赤枠でくくっている部分は、フローチャートが省略されている。別途、フローチャートを作成したため、そちらを参照されたい。まず、本プログラムでは、ヘッダファイルとして“stdio.h”と“stdlib.h”の 2 つを用いている。関数の構成としては、main 関数だけではなく、グローバル変数としてナンプレ問題を宣言し、さらに、2 次元配列をコピーする関数と 9 × 9 のマスを表示する関数を用いている。図 1 の青色で示した部分がこれらに該当する。

2.1　解を得るプログラム

はじめに、94 行目および 95 行目において、配列 masu のうち 0 が格納されている要素番号を記憶する。これはつまり、ナンプレ問題の空欄部分に相当する。この空欄に入る可能性がある数字を配列 find に格納している。しかし、find は、以下で見る処理を行い要素の書き換えを繰り返すため、あらかじめ 1 から 9 までの要素を順に並べた配列 answer を用意しておいた。以下の処理を行う際に、逐一 answer から find にコピーをしてくるようにプログラムされている。

そこで、解を得るアルゴリズムであるが、大きく 3 つに区分される。まず 1 つ目として、99 行目から 132 行目では着目した要素を含む行に与えられている数字を、find から消去していく（厳密には、switch 関数により、当該数字を 0 に上書きしている）。次に 2 つ目として、133 行目から 166 行目では着目した要素を含む列に与えられている数字を、find から消去していく（同様に、当該数字を 0 に上書きしている）。最後に 3 つ目として、167 行目から 204 行目では着目した要素を含む 3×3 のブロック[1]に与えられている数字を、find から消去していく（同様に、当該数字を 0 に上書きしている）。行からの消去法と、列からの消去法の実装は単純であるが、ブロックからの消去法はルールに従った 3×3 の取り方をするには、やや工夫が必要である。本プログラムでは、int 型の性質を利用して、行に対してはブロックでの行番号として $m = (a/3) \times 3$、列に対してはブロックでの列番号として $n = (b/3) \times 3$ を行うことで、a と b の値が必ず 0 か 3 か 6 のいずれかになる。そしてループでは、$m + 3$ または $n + 3$ を迎えるとループを終了させることでルールに従った 3×3 ブロックからの消去法を実現した。

その結果として、find の要素が「8 つの 0 と 1 つの数」という状態になれば、記憶した要素番号の真の要素（空欄に当てはまる答え）となる。この時に初めて、find のうち 0 でないものをこの要素に入れる処理を行う。もし、その find の要素において、複数の候補がある場合は、答えを決めることができないため、記憶した要素番号に対しては何も行わない。この「判定」をしているのが本プログラム 205 行目から 214 行目である。

次に、前述の消去法を行っている各部のアルゴリズムについて述べる。図 2・図 3・図 4 が、該当するフローチャートである。一見するとわかるように、行からの消去法・列からの消去法・ブロックからの消去法、すべてにおいて同一の配列 find を共有しており、switch 文で find の要素を 0 に上書きすることで、考えたい空欄の数字の候補を決定している。例えば、考えたい空欄を含む行に 1 が含まれていた場合、配列 find[0] が 1 であるから、find[0] を 0 に書き換えるという具合である。これを繰り返すと、どんどん考えたい空欄に入る可能性のある（候補となる）数字が絞られていき、もし、1 つに定まればそれを代入すればよいし、1 つに定まらなければすべてのマスを調べ終わってから再度検討すればよい。[2]

[1] ルールに従った取り方をしており、やみくもに 3×3 を取っているわけではない。
[2] 「再度検討」が 1 回で済むかどうかは問題による。

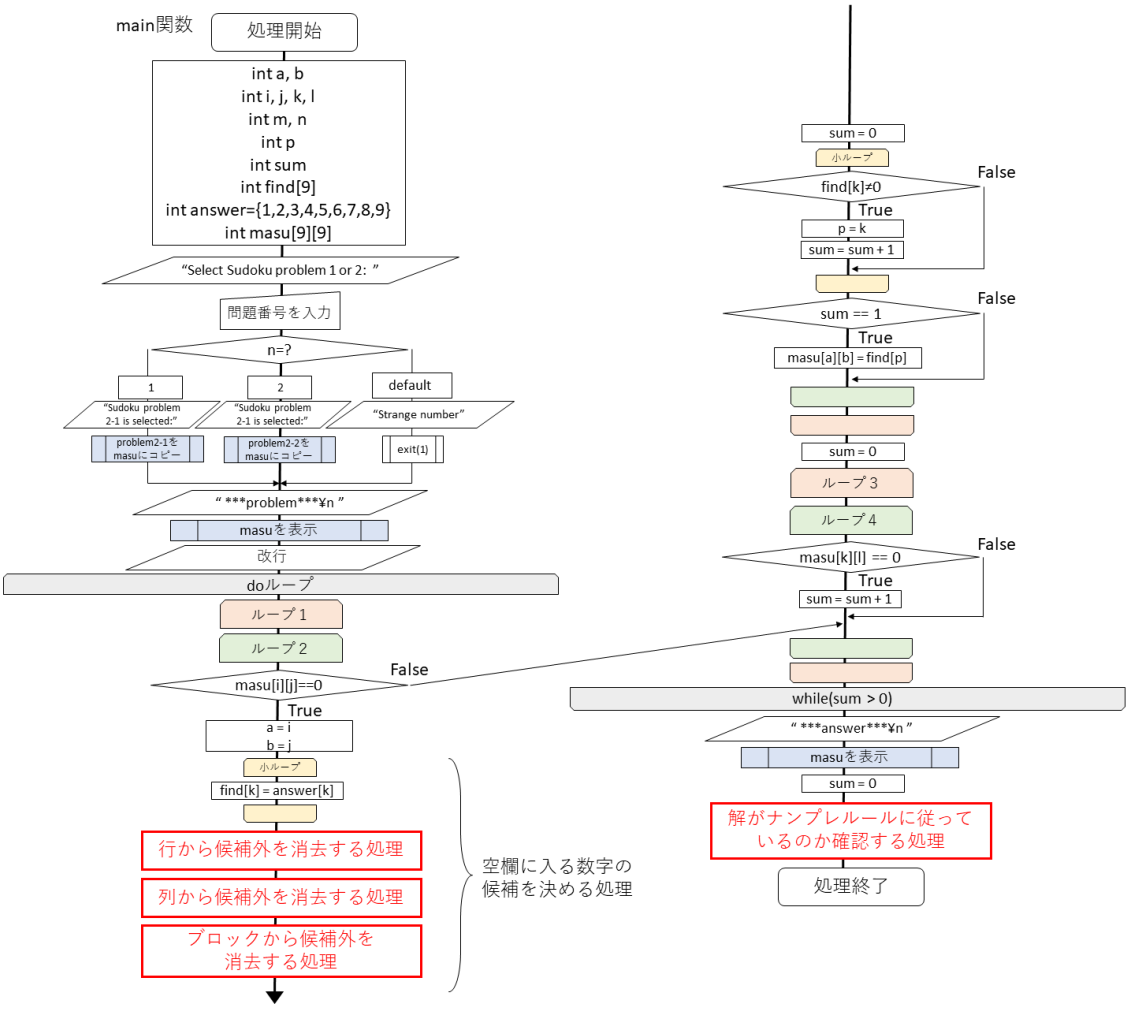

図 1: 本プログラム全体のフローチャート

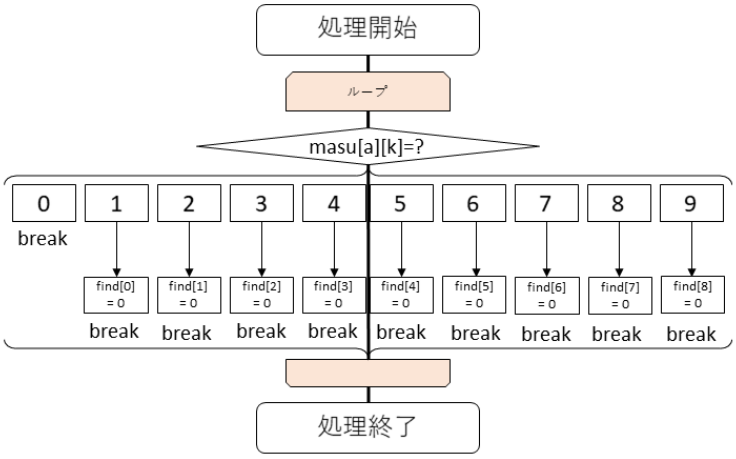

図 2: 「行からの消去法」のフローチャート

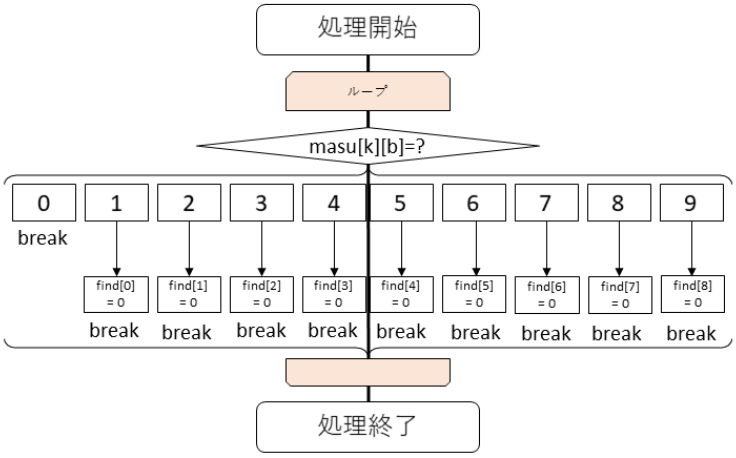

図 3: 「列からの消去法」のフローチャート

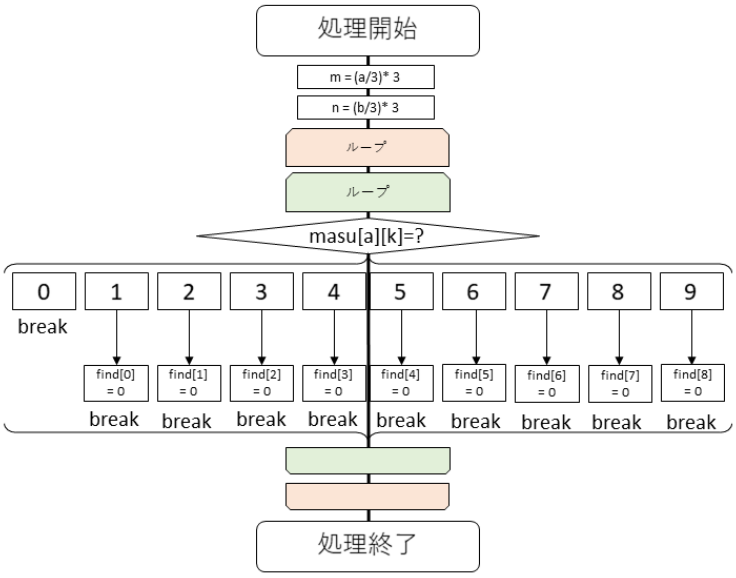

図 4: 「ブロックからの消去法」のフローチャート

2.2 解を検討するプログラム

解を検討するプログラムのフローチャートは図 5 に示すとおりである。ただし、switch 文などの繰り返しを避けるため、適宜割愛したところがある。

本プログラムにおいても switch 文を多用しており、おおむね同じ構造といえる。プログラムの詳細（カウンタの使い分け）に関しては付録のプログラムを参照されたい。

前サブセクションのアルゴリズムを踏襲しているため、基本的にアルゴリズムも同じであるが、異なる部分を述べる。異なる点としては、変数 sum を用いている点である。switch 文で各行・各列・各ブロックすべての組み合わせ 1 つ 1 つについて、それぞれ find を更新し（answer からのコピー）、見つけた数字と同じ数字（find における要素）をどんどん 0 にすることで、0 でないものの数を数えている。もし 0 でない要素が存在すれば sum に 1 を加える。そうすることで、各行・各列・各ブロックの検討後に sum が 0 でなければ、解は誤っていると判断できる。これが図 5 最後の条件分岐部分である。

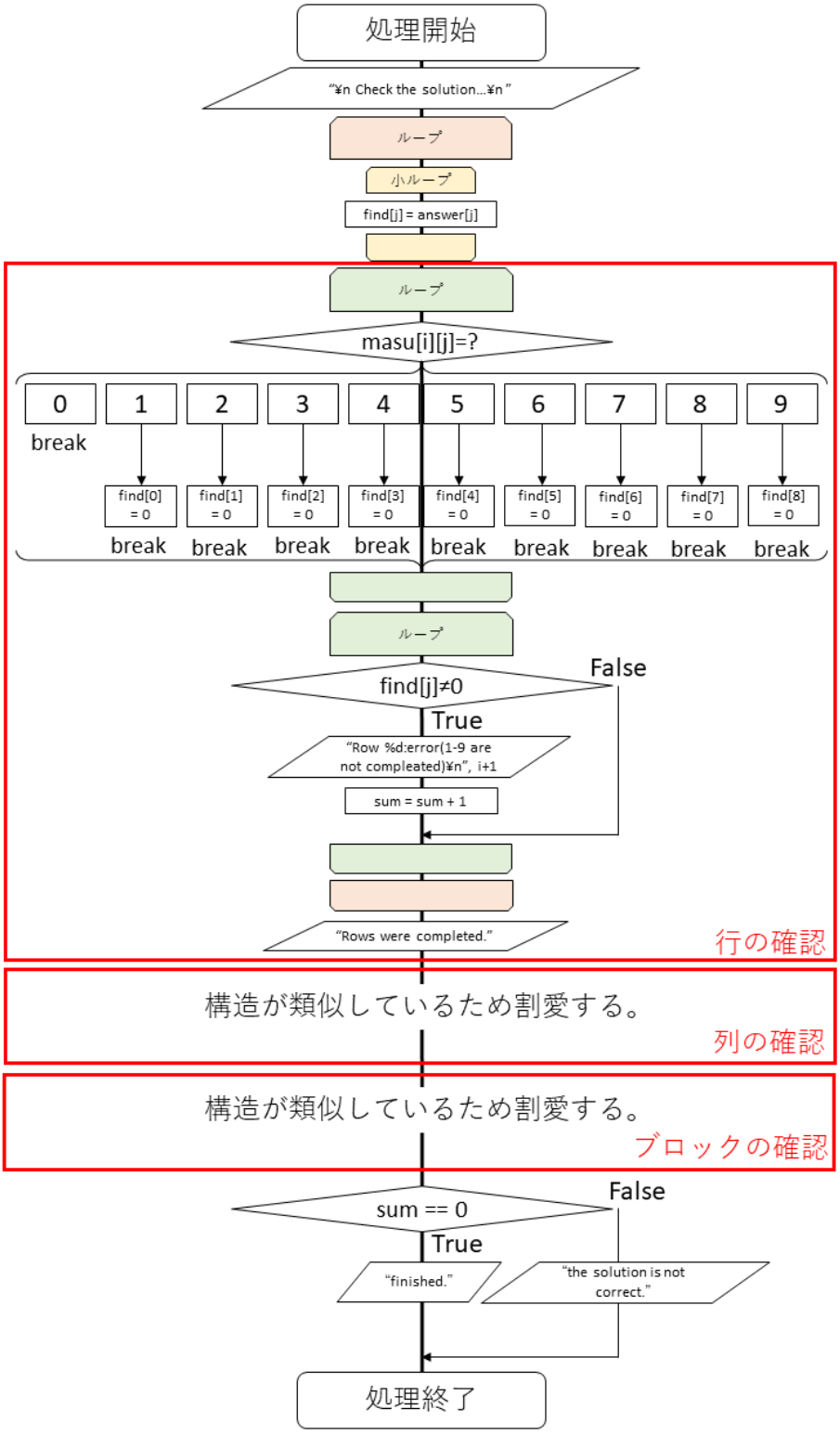

図 5: 解の検討をするプログラムのフローチャート

3　プログラムの設計

本節ではプログラムの設計について述べる。

3.1　入力形式の設計

本プログラムでは、二次元配列 masu に、利用者が選択した問題をコピーすることで与えられる。以下が、その該当箇所である。

```
31  // 2 次元配列をコピーする関数
32  void copy_problem(int masu[9][9], int problem[9][9])
33  {
34      int i, j;
35      for(i=0;i<9;i++)
36                  for(j=0;j<9;j++)
37                          masu[i][j] = problem[i][j];
38  }

70  // 問題を選択する
71      switch(n) {
72        case 1: // 問題 2-1
73         printf("Sudoku problem 2-1 is selected:\n");
74         copy_problem(masu, problem2_1);
75         break;
76        case 2: // 問題 2-2
77         printf("Sudoku problem 2-2 is selected:\n");
78         copy_problem(masu, problem2_2);
79         break;
80        default: // エラー
81         printf("Strange number.\n");
82         exit(1);
83      }
```

3.2　出力形式の設計

本プログラムでは、出力を専門とする関数として、display_masu 関数を独自に用意した。本プログラムは、ナンプレ問題を扱う特性上、配列では 0 と表示される要素を出力では“ - ”と置き換える必要があったため、条件分岐を用いてこれを実現した。

また、主部で得られた解（masu の各要素）を標準出力に出力する。その後、解がナンプレのルールに従っている場合は、その旨を標準出力に出力する。一方、ルールに従っていなかった場合には、その違反していた行・列・ボックスの番号を明示し、その旨を標準出力に出力する。以下が、その該当箇所である。なお、解の検討部分の標準出力に関しては、付録 A の 271 行目から 278 行目ほかを参照されたい。

```
40  // 9x9 のマスを表示する関数
41  void display_masu(int masu[9][9])
42  {
43          int i, j;
44          for (i=0;i<9;i++) {
45                  for (j=0;j<9;j++) {
46                          if (masu[i][j] == 0)
47                          printf("- ");
48                          else
49                          printf("%d ", masu[i][j]);
50                  }
51                  printf("\n");
52          }
53  }
```

3.3 主部の設計

主部は以下のように動作する。まず、利用者が選択したナンプレ問題を配列 masu に与え、2 章で述べたアルゴリズムに従い、ナンプレ問題の解を得る。

4 プログラムの説明

C 言語によりナンプレ問題を解くプログラムおよびその解がナンプレのルールに従っているかを確認するプログラムを作成した。作成したプログラムを付録 A に載せる。作成に当たっては、テキスト [1] を参考にした。

4.1 変数、定数の説明

本プログラムでは表 1 に示すような変数・定数を用いる。これは、プログラムリストの 58 行目から 65 行目に相当する。以下に該当箇所を抜粋したものを示す。

```
58      int a, b;
59      int i, j, k, l;
60      int m, n;
61      int p;
62      int sum;
63      int find[9];
64      int answer[9]={1, 2, 3, 4, 5, 6, 7, 8, 9};
65      int masu[9][9];
```

表 1: ナンプレ問題を解くプログラムで用いる変数、定数とその用途

変数名	型	用途
answer	int 型の 1 次元配列	あらかじめ 1〜9 のデータが与えられた配列
find	int 型の 1 次元配列	マスの空欄に入る数字を見つけるための配列
masu	int 型の 2 次元配列	ナンプレ問題を扱う配列
a	int 型	探索対象となる要素の行番号を表す変数
b	int 型	探索対象となる要素の列番号を表す変数
i, j, k, l	int 型	ループ処理に用いるカウンタ
m	int 型	ブロックの先頭要素の行番号を表す変数
n	int 型	ブロックの先頭要素の列番号を表す変数
p	int 型	空欄マスに入る数字を格納する配列 find の要素番号を記憶する変数
sum	int 型	条件分岐の際に用いるカウンタ

※注　a, b は、328 行目・329 行目においてのみ本来の用途とは別に、ループカウンタとして利用した。

4.2　主部の説明

主部では主に、2 章で述べたアルゴリズムに従って、ナンプレ問題の解を得て、これを検証する。

5　工夫した点

本プログラムの 8 行目から 17 行目と 20 行目から 29 行目に 2 題のナンプレ問題を用意しているが、これらの問題を新たな問題に差し替えることで、その問題の正解をすぐに得ることができる。または、本プログラムにさらに問題を追加することで、利用者がより多くのナンプレ問題を利用することが可能である。このように本プログラムでは、拡張性を持たせるよう工夫を行った。

6　プログラムの制限事項

本プログラムは、ナンプレの問題において行・列・3 × 3 のブロックを比較するだけで順々に値が定まる問題にのみ対応している。したがって、どのマスにも値が複数個入る可能性がある場合に、推定により数字を選定し解を決定するやや難しい問題には対応していない。もし対応していない問題をこのプログラムに解かした場合、問題を表示するだけで、解答は永久に表示されない（ループが回り続ける）。

7　実行例

プログラムの動作を確認するため、標準課題の１と２を実際に実行した。その結果、いずれの場合においても、正しく解が得られ、その解がナンプレのルールに従っていることも確認できた。以下は、順に標準課題１、標準課題２の実行結果である。

```
Select Sudoku problem 1 or 2: 1
Sudoku problem 2-1 is selected:
 *** problem ***
5 3 - - 7 - - - -
6 - - 1 9 5 - - -
- 9 8 - - - - 6 -
8 - - - 6 - - - 3
4 - - 8 - 3 - - 1
7 - - - 2 - - - 6
- 6 - - - - 2 8 -
- - - 4 1 9 - - 5
- - - - 8 - - 7 9

 *** answer ***
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9

Check the solution...
Rows were compleated.
Columns were compleated.
Blocks were compleated.
finished.
```

```
Select Sudoku problem 1 or 2: 2
Sudoku problem 2-2 is selected:
 *** problem ***
8 1 - - 2 - 4 - -
6 - 7 - 3 - 1 - -
- 2 9 - - 4 - - -
- - - - 9 2 - 3 1
- 7 1 4 - 3 6 9 -
9 3 - 6 8 - - - -
- - - 9 - - 2 8 -
- - 8 - 7 - 3 - 4
- - 2 - 4 - - 1 6

 *** answer ***
8 1 3 5 2 7 4 6 9
6 4 7 8 3 9 1 2 5
5 2 9 1 6 4 8 7 3
4 8 6 7 9 2 5 3 1
2 7 1 4 5 3 6 9 8
9 3 5 6 8 1 7 4 2
3 6 4 9 1 5 2 8 7
1 9 8 2 7 6 3 5 4
7 5 2 3 4 8 9 1 6

Check the solution...
Rows were compleated.
Columns were compleated.
Blocks were compleated.
finished.
```

筆者はナンプレをほとんど解いたことがなかったため、本プログラムと同様のアルゴリズム、すなわち行・列・3 × 3 のブロックの比較のみで解いた場合、解を得るのに 20 分以上要した。しかしながら、本プログラムを表 2 の環境で実行した場合、実行とほぼ同時に結果を得ることができた。

8 考察

プログラミング A 課題 1 では、バブルソートの実行時間に数十秒要していたため、本プログラムでも多くの処理時間を要するのではないかと予想していたが、実際には、ナンプレを解いて、その解が正しいことを確認するのに 1 秒もかかっていなかった。動作の確認にあたっては表 2 に示す環境で行った。

表 2: 性能評価に用いた環境

CPU	Intel i7 1.3 GHz
Memory	16 GB
OS	Windows 10
C コンパイラ	gcc

また、実際に解を検討するプログラムが正しく動作するのかどうかを確かめるために、本プログラムのうち「解を得るプログラム」の部分を削除し、解が得られていない（masu が初期の与えられた配列のまま）状態で、解の検討をするプログラムを実行してみた。以下は、その実行結果である。

```
 *** test ***
5 3 4 6 7 8 - 1 2
6 7 - 1 9 5 3 4 8
1 9 8 - 4 2 5 6 7
8 - 9 7 6 1 4 2 3
4 2 6 8 5 3 - 9 1
7 1 3 9 2 4 8 5 -
- 6 1 5 3 7 2 8 4
2 8 7 4 - 9 6 3 5
3 4 5 2 8 6 1 - 9

Check the solution...
Row 1:error(1-9 are not compleated)
Row 2:error(1-9 are not compleated)
Row 3:error(1-9 are not compleated)
Row 4:error(1-9 are not compleated)
Row 5:error(1-9 are not compleated)
Row 6:error(1-9 are not compleated)
Row 7:error(1-9 are not compleated)
Row 8:error(1-9 are not compleated)
Row 9:error(1-9 are not compleated)
Rows were compleated.
column 1:error(1-9 are not compleated)
column 2:error(1-9 are not compleated)
column 3:error(1-9 are not compleated)
column 4:error(1-9 are not compleated)
column 5:error(1-9 are not compleated)
column 7:error(1-9 are not compleated)
column 7:error(1-9 are not compleated)
column 8:error(1-9 are not compleated)
column 9:error(1-9 are not compleated)
Columns were compleated.
block 1-1:error(1-9 are not compleated)
block 1-2:error(1-9 are not compleated)
block 1-3:error(1-9 are not compleated)
block 2-1:error(1-9 are not compleated)
block 2-3:error(1-9 are not compleated)
block 2-3:error(1-9 are not compleated)
block 3-1:error(1-9 are not compleated)
block 3-2:error(1-9 are not compleated)
block 3-3:error(1-9 are not compleated)
Blocks were compleated.
the solution is not correct.
```

このように、ナンプレのルールを満たしていない行・列・ブロックの番号を明示できており、最終文では“the solution is not correct.”と出力されていることから、きちんとプログラムは成立していることが分かる。

9　感想

課題が出た当初は、段階１ですらまったくプログラムを書くことができなかった。初めて書いたプログラムでは、コンパイルは通るものの、与えられたナンプレ問題の埋まっている数字の部分も書き換わってしまっていたり、埋まっている数字同士が入れ替わっていたりと、まったく使い物にならないものが出来上がっていた。しかしながら、日が経って脳をリセットした状態で改めてプログラムを見ていると、関数の置く場所がもう一つ後ろの中括弧であったり、グローバル変数とローカル変数が間違っていたりしていることに気づくことができ、直してみるときちんと動作した。段階１・２では比較的ミスをすぐに見つけることができたが、段階３ともなると、ミスを見つけるのにも何日も何時間もかかってしまった。そうした中でも、プログラミングに詳しい人に助言を求めたり、インターネット上にある情報を収集しようとしたりせず、完璧に独力で最後までプログラムを書ききることができたのは、非常に良い経験となった。また、プログラムを何度も実行して、実行結果を確認するときに、ループ処理が永遠に終わらなかったり、最後までプログラムが処理されずにスキップされてしまったりしたが、そうした時には何が間違っていたのか、原因を突き止めることができたため、今後、プログラムを書く際には参考にしたいと思う。

10　作業工程

課題を提出するまでの作業内容を日付と時間とともに列挙する。

- アルゴリズム設計 (2020 年 7 月 16 日 14:00–15:00)
- プログラム設計 (2020 年 7 月 16 日 15:00–16:00)
- コーディング (2020 年 7 月 20 日 9:00–12:00)
- デバッグ (2020 年 7 月 21 日 18:00–19:00)
- レポート作成 (2020 年 7 月 21 日 21:00–22:00，2020 年 7 月 22 日 21:00–22:00，2020 年 8 月 1 日 9:00–13:00)

参考文献

[1]　浅井 宗海, 栗原 徹,“プログラミング入門 C 言語”, 実教出版, 2005.

A　プログラムリスト

評価用に作成したプログラムを下記に示す。

```
1   //
2   // 数独プログラム
3   //
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   // 標準課題 2-1
8   int problem2_1[9][9] =
9   {{5,3,0,0,7,0,0,0,0},
10  {6,0,0,1,9,5,0,0,0},
11  {0,9,8,0,0,0,0,6,0},
12  {8,0,0,0,6,0,0,0,3},
13  {4,0,0,8,0,3,0,0,1},
14  {7,0,0,0,2,0,0,0,6},
15  {0,6,0,0,0,0,2,8,0},
16  {0,0,0,4,1,9,0,0,5},
17  {0,0,0,0,8,0,0,7,9}};
18
19  // 標準課題 2-2
20  int problem2_2[9][9] =
21  {{8,1,0,0,2,0,4,0,0},
22  {6,0,7,0,3,0,1,0,0},
23  {0,2,9,0,0,4,0,0,0},
24  {0,0,0,0,9,2,0,3,1},
25  {0,7,1,4,0,3,6,9,0},
26  {9,3,0,6,8,0,0,0,0},
27  {0,0,0,9,0,0,2,8,0},
28  {0,0,8,0,7,0,3,0,4},
29  {0,0,2,0,4,0,0,1,6}};
30
31  // 2 次元配列をコピーする関数
32  void copy_problem(int masu[9][9], int problem[9][9])
33  {
34      int i, j;
35      for(i=0;i<9;i++)
36                  for(j=0;j<9;j++)
37                          masu[i][j] = problem[i][j];
38  }
39
40  // 9x9 のマスを表示する関数
41  void display_masu(int masu[9][9])
42  {
43          int i, j;
44          for (i=0;i<9;i++) {
45                  for (j=0;j<9;j++) {
46                          if (masu[i][j] == 0)
47                          printf("- ");
48                          else
49                          printf("%d ", masu[i][j]);
50                  }
51                  printf("\n");
52          }
53  }
54
```

```
55  // メイン関数
56  int main()
57  {
58      int a, b;
59      int i, j, k, l;
60      int m, n;
61      int p;
62      int sum;
63      int find[9];
64      int answer[9]={1, 2, 3, 4, 5, 6, 7, 8, 9};
65      int masu[9][9];
66
67      printf("Select Sudoku problem 1 or 2: ");
68      scanf("%d", &n);
69
70  // 問題を選択する
71      switch(n) {
72        case 1: // 問題 2-1
73         printf("Sudoku problem 2-1 is selected:\n");
74         copy_problem(masu, problem2_1);
75         break;
76        case 2: // 問題 2-2
77         printf("Sudoku problem 2-2 is selected:\n");
78         copy_problem(masu, problem2_2);
79         break;
80        default: // エラー
81         printf("Strange number.\n");
82         exit(1);
83          }
84
85  // 問題を表示する
86      printf(" *** problem ***\n");
87      display_masu(masu);
88      printf("\n");
89
90  // 解の選定
91      do{
92      for(i=0;i<9;i++){ // 対象となる配列番号の行を指定
93                  for(j=0;j<9;j++){ // 対象となる配列番号の列を指定
94                          if(masu[i][j]==0){
95                                  a = i; b = j; // 対象となる配列番号を記憶
96                                  for (k=0;k<9;k++){
97                                       find[k]=answer[k];
98                                  }
99                                  // 行からの消去法
100                                 for(k=0;k<9;k++){
101                                         switch(masu[a][k]){
102                                                 case 1:
103                                                  find[0] = 0;
104                                                  break;
105                                                 case 2:
106                                                  find[1] = 0;
107                                                  break;
108                                                 case 3:
109                                                  find[2] = 0;
110                                                  break;
111                                                 case 4:
112                                                  find[3] = 0;
```

```
                        break;
                    case 5:
                        find[4] = 0;
                        break;
                    case 6:
                        find[5] = 0;
                        break;
                    case 7:
                        find[6] = 0;
                        break;
                    case 8:
                        find[7] = 0;
                        break;
                    case 9:
                        find[8] = 0;
                        break;
                    case 0:
                        break;
            }
        }
        // 列からの消去法
        for(k=0;k<9;k++){
            switch(masu[k][b]){
                    case 1:
                        find[0] = 0;
                        break;
                    case 2:
                        find[1] = 0;
                        break;
                    case 3:
                        find[2] = 0;
                        break;
                    case 4:
                        find[3] = 0;
                        break;
                    case 5:
                        find[4] = 0;
                        break;
                    case 6:
                        find[5] = 0;
                        break;
                    case 7:
                        find[6] = 0;
                        break;
                    case 8:
                        find[7] = 0;
                        break;
                    case 9:
                        find[8] = 0;
                        break;
                    case 0:
                        break;
            }
        }
        // ブロックからの消去法
        m=(a/3)*3; // m=0,3,6 のいずれかを得る
        n=(b/3)*3; // n=0,3,6 のいずれかを得る
        for(k=m;k<m+3;k++){
```

```
171                                             for(l=n;l<n+3;l++){
172                                                     switch(masu[k][l]){
173                                                             case 1:
174                                                              find[0] = 0;
175                                                              break;
176                                                             case 2:
177                                                              find[1] = 0;
178                                                              break;
179                                                             case 3:
180                                                              find[2] = 0;
181                                                              break;
182                                                             case 4:
183                                                              find[3] = 0;
184                                                              break;
185                                                             case 5:
186                                                              find[4] = 0;
187                                                              break;
188                                                             case 6:
189                                                              find[5] = 0;
190                                                              break;
191                                                             case 7:
192                                                              find[6] = 0;
193                                                              break;
194                                                             case 8:
195                                                              find[7] = 0;
196                                                              break;
197                                                             case 9:
198                                                              find[8] = 0;
199                                                              break;
200                                                             case 0:
201                                                              break;
202                                                     }
203                                             }
204                                     }
205                                     sum=0;
206                                     for(k=0;k<9;k++){
207                                             if(find[k]!=0){
208                                                     p = k;
209                                                     sum++;
210                                             }
211                                     }
212                                             if(sum==1) {
213                                                     masu[a][b]=find[p];
214                                             }
215                                     }
216                             }
217                     }
218             sum=0;
219             for(k=0;k<9;k++){
220                     for(l=0;l<9;l++){
221                             if(masu[k][l]==0) sum++;
222                     }
223             }
224             }while(sum>0);
225
226     // 解の表示
227         printf(" *** answer ***\n");
228         display_masu(masu);
```

```
229
230 // 解の確認
231     sum=0;
232     // 列の確認
233     printf("\nCheck the solution...\n");
234         for(i=0;i<9;i++){
235                 for (j=0;j<9;j++){
236                 find[j]=answer[j];
237                 }
238         for(j=0;j<9;j++){
239                         switch(masu[i][j]){
240                                 case 1:
241                                  find[0] = 0;
242                                  break;
243                                 case 2:
244                                  find[1] = 0;
245                                  break;
246                                 case 3:
247                                  find[2] = 0;
248                                  break;
249                                 case 4:
250                                  find[3] = 0;
251                                  break;
252                                 case 5:
253                                  find[4] = 0;
254                                  break;
255                                 case 6:
256                                  find[5] = 0;
257                                  break;
258                                 case 7:
259                                  find[6] = 0;
260                                  break;
261                                 case 8:
262                                  find[7] = 0;
263                                  break;
264                                 case 9:
265                                  find[8] = 0;
266                                  break;
267                                 case 0:
268                                  break;
269                         }
270                 }
271                 for(j=0;j<9;j++){
272                         if(find[j]!=0){
273                                 printf("Row %d:error(1-9 are not compleated)\n",i+1);
274                                 sum++;
275                         }
276                 }
277         }
278         printf("Rows were compleated.\n");
279
280         // 行の確認
281         for(j=0;j<9;j++){
282                 for(i=0;i<9;i++){
283                 find[i]=answer[i];
284                 }
285         for(i=0;i<9;i++){
286                         switch(masu[i][j]){
```

```
                                case 1:
                                 find[0] = 0;
                                 break;
                                case 2:
                                 find[1] = 0;
                                 break;
                                case 3:
                                 find[2] = 0;
                                 break;
                                case 4:
                                 find[3] = 0;
                                 break;
                                case 5:
                                 find[4] = 0;
                                 break;
                                case 6:
                                 find[5] = 0;
                                 break;
                                case 7:
                                 find[6] = 0;
                                 break;
                                case 8:
                                 find[7] = 0;
                                 break;
                                case 9:
                                 find[8] = 0;
                                 break;
                                case 0:
                                 break;
                        }
                }
                for(i=0;i<9;i++){
                        if(find[i]!=0){
                                printf("column %d:error(1-9 are not compleated)\n",j+1);
                                sum++;
                        }
                }
        }
        printf("Columns were compleated.\n");

        // ブロックの確認
        for(a=0;a<9;a=a+3){
                for(b=0;b<9;b=b+3){
                        for(k=0;k<9;k++){
                                find[k]=answer[k];
                        }
                        for(i=a;i<a+3;i++){
                                for(j=b;j<b+3;j++){
                                        switch(masu[i][j]){
                                                case 1:
                                                 find[0] = 0;
                                                 break;
                                                case 2:
                                                 find[1] = 0;
                                                 break;
                                                case 3:
                                                 find[2] = 0;
                                                 break;
```

```
345                                                 case 4:
346                                                  find[3] = 0;
347                                                  break;
348                                                 case 5:
349                                                  find[4] = 0;
350                                                  break;
351                                                 case 6:
352                                                  find[5] = 0;
353                                                  break;
354                                                 case 7:
355                                                  find[6] = 0;
356                                                  break;
357                                                 case 8:
358                                                  find[7] = 0;
359                                                  break;
360                                                 case 9:
361                                                  find[8] = 0;
362                                                  break;
363                                                 case 0:
364                                                  break;
365                                          }
366                                     }
367                            }
368                            for(k=0;k<9;k++){
369                                 if(find[k]!=0){
370                                    printf("block %d-%d:error(1-9 are not compleated)\n",(a/3)+1,(b/3)+1);
371                                    sum++;
372                       }
373                            }
374                   }
375          }
376          printf("Blocks were compleated.\n");
377          if(sum==0) printf("finished.\n");
378          else printf("the solution is not correct.\n");
379   }
```