第 2 回 C++プログラミングⅡ課題レポート

出題日：2021 年 6 月 18 日（金）
提出期限：2021 年 7 月 2 日（金）23:59
提出方法：CoursePower の指定された箇所に提出

課題 2

複数の壁や箱を利用可能な倉庫番の作成と、倉庫番の解の自動検索のプログラミング課題を通して、C++の機能について学びます。以下の注意をよく読み、課題に取り組んでください。

- レポートは Word で作成し、docx ファイルを提出してください。必要なソースコードはレポート内に記載してください。
- 第 11 回実験課題を完成させていることが前提の課題です。先に第 11 回実験課題に取り組んでください。
- 課題は、基礎編(2A)、応用編(2B)、発展編(2C)に分かれています。
- 基礎編の課題は必ず完成させてください。
- 応用編の課題はできるだけ完成させてください。
- 発展編の課題は、基礎編・応用編をすべて完成させた上で、余裕がある場合は取り組んでみてください。
- 各課題(2A-2C)には、それぞれレポートに記載すべきコードの範囲、実行結果、考察内容が明記されています。必要な項目がレポートに記載されていない場合は、その課題は採点の対象としない場合がありますので注意してください。
- 質問・問い合わせは、s02767@cc.seikei.ac.jp(松田宛)もしくは授業の Teams のレポート課題チャネルまでお願いします。

基礎編（課題 2A）

第 11 回実験課題で作成したプログラムでは、壁、箱、ゴール全てが 1 個だけ存在すると仮定しているが、通常の倉庫番では複数存在する。そこで、下記の方針に従って wall, box, goal を vector に変更し、複数もしくは 0 個でも機能するようにコードを変更すること。

1. #include<vector>を追加し「ゲームフィールドの初期設定」部分を以下の例のように変更する。

```
const int row{3};
const int column{3};
const std::vector<Pos> wall{}; //壁 0 個
const std::vector<Pos> goal{{0,0},{2,1}}; //ゴール 2 個
Pos player{2,0};
std::vector<Pos> box{{0,1},{1,1}}; //箱 2 個
```

2. #include<algorithm>を追加し、wall, box, goal と同一座標にあるという判定をしていた部分を「vector の要素に含まれているかどうかを判定する」ように変更する。
   ※vector の algorithm の機能を利用する（方法は色々とあるので自分で選択すること）。
3. completed 関数を「順番に関係なく goal の全ての要素が box の全ての要素と一致しているかを判定する」ものに変更する。
4. box の移動時に、他の box に重なるかどうかの判定を追加し、重なっていない場合のみ移動するように変更する。
5. レポートには、完成させたソースコード全体、自分で考えた異なる初期配置での実行結果の画像、vector の要素に含まれているかを判定するのに利用した algorithm 機能の説明、を記載すること。

課題 2A 完成版実行例（初期配置は上記の例、下線はキーボードからの入力）

```
./sokoban_report
XXXXX
X+$ X
X $ X
X@+ X
XXXXX
r
XXXXX
X+$ X
X $ X
X @ X
XXXXX
r
XXXXX
X+$ X
X $ X
X +@X
XXXXX
u
XXXXX
X+$ X
X $@X
X + X
XXXXX
u
XXXXX
X+$@X
X $ X
X + X
XXXXX
l
XXXXX
XO@ X
X $ X
X + X
XXXXX
d
XXXXX
XO  X
X @ X
X O X
XXXXX
The work is completed!!
```

応用編（課題 2B）

下記の方針をもとに、与えられた倉庫番の問題の解を自動で探索する関数 search を作成し、キーボードから's' を入力することで呼び出せるようにすること。

1. #include<string>を追加して、更に Field クラスに「void moves(std::string s)」というメンバ関数を追加し、"uur"（上上右）のように文字列で複数の移動方向を一度に入力できるようにする
2. search 関数を作成し、n 回の全ての移動方向の組み合わせを用意し（例えば n=2 の場合は、{"uu", "ud", "ur", "ul", "du", "dd", "dr", "dl", "ru", "rd", "rr", "rl", "lu", "ld", "lr", "ll"}の 16 通り）、ゲームクリアとなる組合せが存在するかを調査する。
3. search 関数は n を徐々に増やして解が見つかるまで調査を続行し、n が上限に達した場合（PC のメモリ環境にもよるが n=6 程度にしておいた方が安全）は「unsolvable」（解無し）として調査を終了する。
4. main 関数で、キーボードから's'の入力があった場合は、search 関数を呼び出して探索結果を表示して終了する。
5. レポートには、作成した moves 関数と search 関数のソースコード、自分で考えた異なる初期配置での実行結果の画像、search 関数で n 回の移動方向の組み合わせを用意した方法に関する説明、を記載すること。

課題 2B 完成版実行例（初期配置は 2A の例、下線はキーボードからの入力）

```
./sokoban_report
XXXXX
X+$ X
X $ X
X@+ X
XXXXX
s
rruuld
```

発展編（課題 2C）

※応用編が完成し、更に余裕がある場合は取り組むと良い。

※発展編では必要な記載事項の指定はない。必要と思われるコード、結果、考察を自分で整理してレポートに記載すること。未完成の場合も取り組んだところまで記載すると良い。

課題 2B の方法では解を探索するのに全ての移動方向の組み合わせを用意しており、解までの移動回数が大きくなると、メモリの使用量が膨大となり解けなくなる。一方、倉庫番では、実際には動けない方向やフィールドの状態の重複が多く、移動方向の全ての組み合わせを探索するのは無駄である。そこで、移動方向ではなく、フィールドの状態を基本に探索し、より複雑な問題でも解を探索可能な関数 fast_search を作成すること

ヒント 1：「n 回の移動方向の組み合わせ」ではなく「n 回の移動で到達可能な全ての Field の状態」を記録するとよい。

ヒント 2：Field の状態だけでは移動手順が失われるので、「その状態への最短移動手順」も文字列等で併せて記録するとよい。

以上