

NCU 2022

Data Structure

Assignment #3

## **Shogi Game (Linked List)**

**Cheng Yu Hin**

110503534

## **Abstract**

This is a game project which simulates the gameplay of Shogi. It is written in C and uses the linked list data structure. It supports real-time matches and replay. Two users can play in the same match. All the steps will be recorded and can be outputted to a file. The save file, then, can be loaded for later replay.

## **Table of Contents**

<b>Features</b>	<b>2</b>
<b>Kifu Scope</b>	<b>2</b>
<b>Program Structure</b>	<b>2</b>
<b>Modules</b>	<b>3</b>
main.c	3
shogi.c	4
replay.c	5
shogilib.c	6
kifu.c	6
<b>Data Structure</b>	<b>7</b>
<b>Methods</b>	<b>10</b>
shogi.c	10
shogilib.c	11
kifu.c	14
replay.c	16
<b>Compiling</b>	<b>17</b>
<b>Running the Application</b>	<b>17</b>
<b>Demo Screenshots</b>	<b>18</b>

## Features

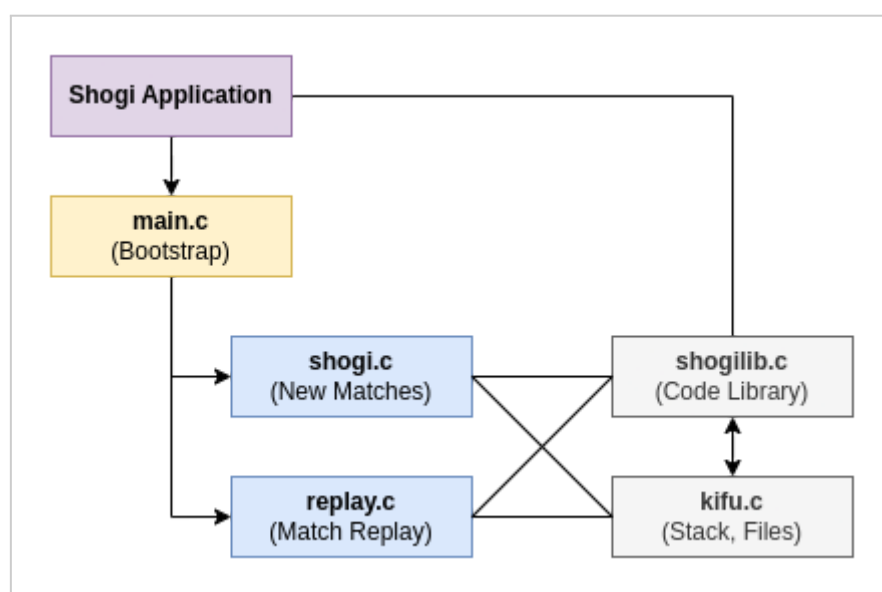
- Shogi game simulation
  - Different shogi pieces
  - Rendering of the kifu
  - Movement of pieces
  - Movement validation
  - “Eating” of pieces
- Reversion of steps
- Kifu record
  - File output of kifu
- Replay of a match (w/ timeline controls)
- Timer between moves in matches

## Kifu Scope

Rather than recording everything on the entire board, this project records the actions of individual pieces. For reversion/replay, the system moves the pieces according to the action records.

There are two types of actions — moving and “eating”. In the case of “eating”, the eaten piece (被吃棋子) will be recorded first, then the eater piece’s (吃棋方) movement will be recorded.

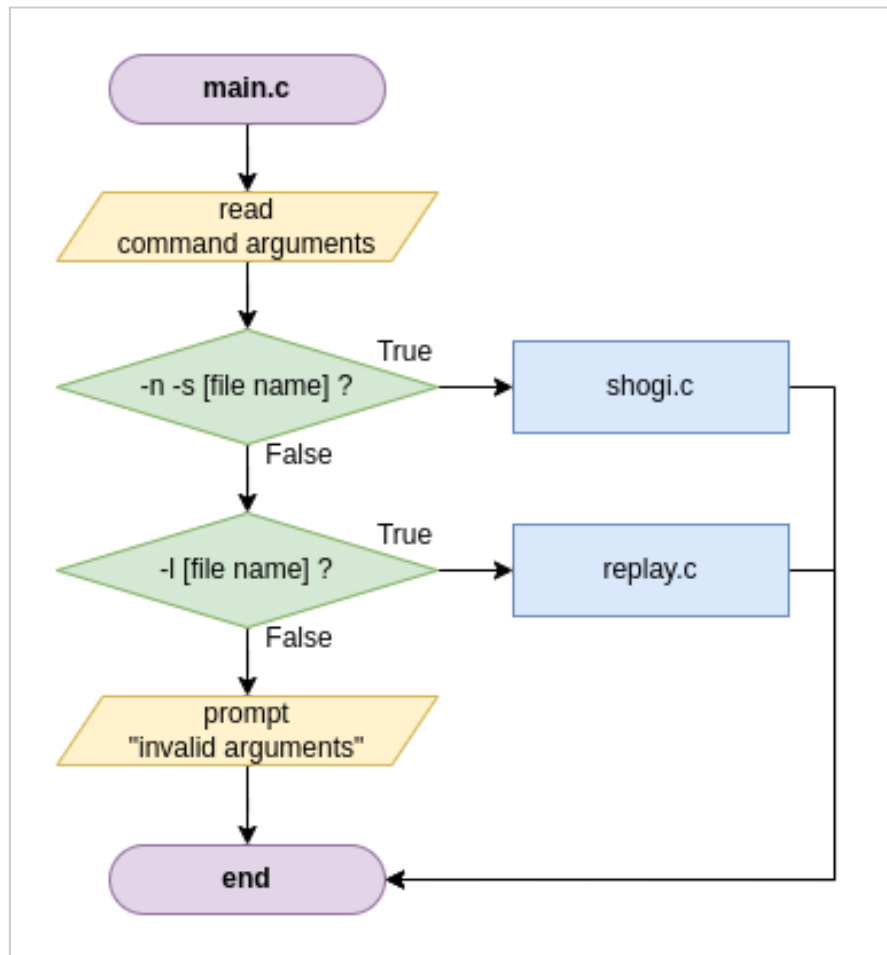
## Program Structure



## Modules

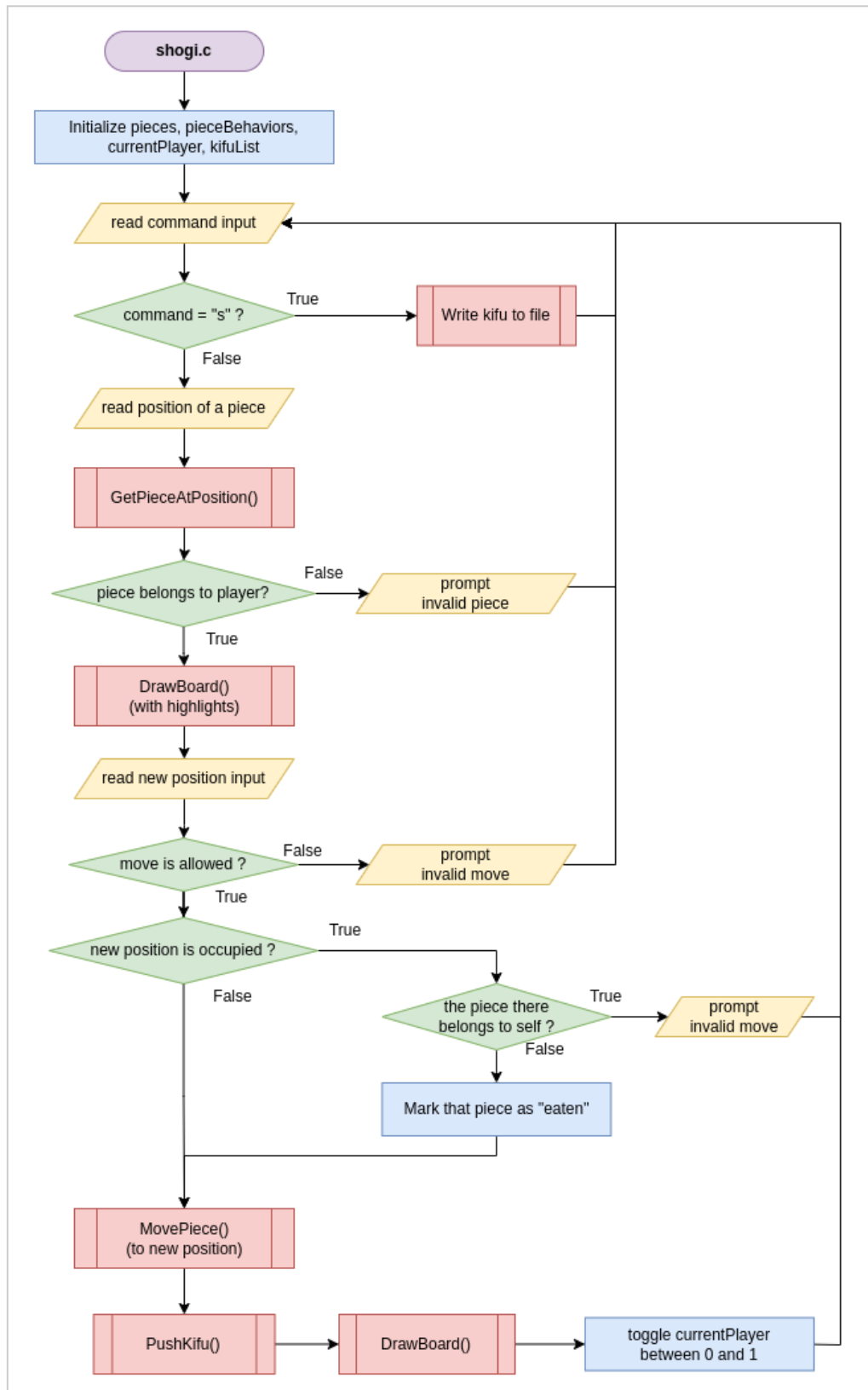
### main.c

This is the entry point of the program. It will extract and compare the command arguments and launch the appropriate modules.



## shogi.c

This module handles the real-time match of a shogi game. Two players can move the pieces in turns. It is responsible for parsing the user inputs and performing match actions (such as checking the moves, moving pieces, recording actions).

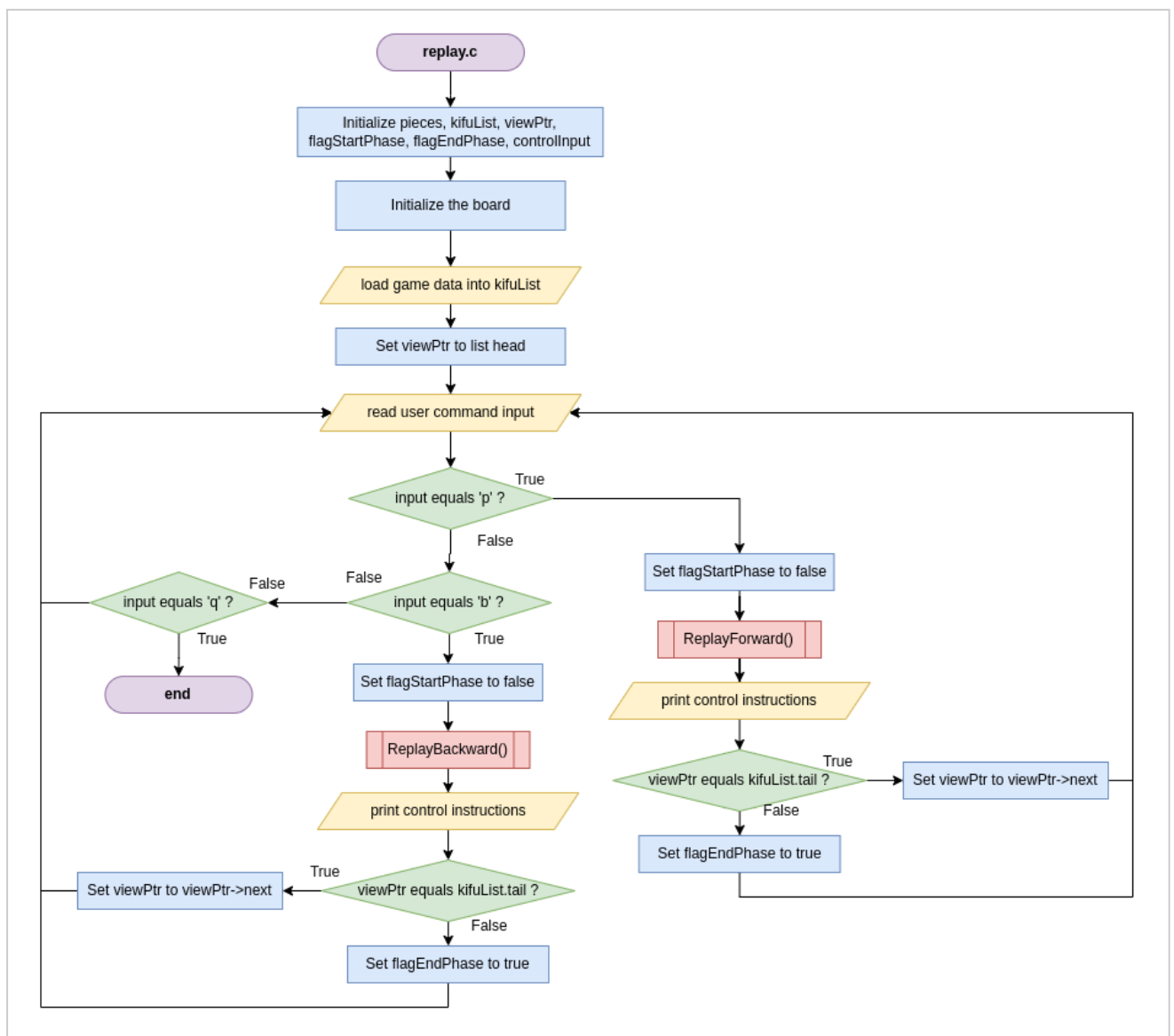


## replay.c

This module handles the replay of a specific match. It will load a saved match from a file. The user can switch back and forth to see each step.

For “eating” actions, the eat and the move actions are consecutive. Therefore, the system will play one more step if the “eating” actions are involved.

*viewPtr:* A dynamic pointer which moves along the kifur list  
*flagStartPhase:* Is viewPtr at the head of kifurList?  
*flagEndPhase:* Is viewPtr at the tail of kifurList?



### **shogilib.c**

This module includes the fundamental data types and methods required. It provides low-level functions like drawing the board and algorithms for validating the moves.

### **kifu.c**

This module provides functions and data types for the kifu (match record). It includes a stack for recording the steps, as well as methods to read/write a kifu stack from/to a file.

The format of saving the kifu stack is as follows (headers are not included):

<b>oldX</b>	<b>oldY</b>	<b>newX</b>	<b>newY</b>	<b>type</b>	<b>isEaten</b>	<b>Player</b>
2	7	2	6	8	0	0
2	3	2	4	8	0	1
2	6	2	5	8	0	0
...						

## Data Structure

### **enum** PieceType

An enum containing all available piece types.

name	label
King	王
King2	玉
Rook	飛
Bishop	角
Gold	金
Silver	銀
Knight	桂
Lance	香
Pawn	歩

### **struct** PieceBehavior

Contains the allowed behavior of a piece. Used for move validation.

name	type	description
type	PieceType	The type of the piece
n	int	Is the piece allowed to move in the specified direction?
ne	int	
e	int	
se	int	
s	int	
sw	int	
w	int	
nw	int	
allowMulti	int	Can the piece move multiple units at a time?
allowJump	int	Can the piece jump over other pieces?



### **struct Piece**

Contains properties of a piece. It is dynamic.

name	type	description
type	PieceType	The type of the piece
player	int	Which player does this piece belong to (0 or 1)
x	int	The x-coordinate of this piece
y	int	The y-coordinate of this piece
isEaten	int	Is this piece being eaten?

### **struct KifuRecord**

Records data of the movement of a piece.

name	type	description
type	PieceType	The type of the piece
oldX	int	The x-coordinate before moving
oldY	int	The y-coordinate before moving
newX	int	The x-coordinate after moving
newY	int	The y-coordinate after moving
player	int	Which player does this piece belong to (0 or 1)
isEaten	int	Is this piece being eaten?

### **struct Node**

A linked list node.

name	type	description
record	KifuRecord	The type of the piece
prev	struct node*	Pointer to the previous item
next	struct node*	Pointer to the next item

**struct** NodePtr

A pointer to a link list node. Equivalent to **struct node\*** .

**struct** KifuList

A linked list containing kifu data.

name	type	description
head	NodePtr	Pointer to the first item
tail	NodePtr	Pointer to the last item

## Methods

### shogi.c

```
void RequestInput(Pieces* pieces, PieceBehavior* pieceBehaviors, int  
currentPlayer, KifuList* kifuList, const char* kifuFileName, unsigned  
restartTimer)
```

Request game input from the user, and process the move.

Parameter	Description
<b>Pieces*</b> pieces	The array containing all 40 pieces
<b>PieceBehavior*</b> pieceBehaviors	The array containing behavior of all the pieces
<b>int</b> currentPlayer	Which player's turn is this (0 or 1)
<b>KifuList*</b> kifuList	The pointer to the kifu list
<b>const char*</b> kifuFileName	The name of the save file
<b>unsigned</b> restartTimer	Should the timer be restarted? (Used as boolean)

---

```
void ShogiGame(const char* kifuFileName)
```

The entry point of a new shogi match.

Parameter	Description
<b>const char*</b> kifuFileName	The name of the save file

## shogilib.c

**char\* GetHans(PieceType type)**

Get the Hans representation of a piece type.

Parameter	Description
PieceType type	The type of the piece

**Returns:** A Hans character representing the type of a piece

---

**void DrawBoard(Piece\* const pieces, int highlightX, int highlightY)**

Render all the pieces and the board. Highlights a piece when applicable.

Parameter	Description
Pieces* pieces	The array containing all 40 pieces
int highlightX	The x-coordinate of the piece to highlight
int highlightY	The y-coordinate of the piece to highlight

---

**void AssignPiece(Piece\* piece, int player, PieceType piece, int x, int y, int isEaten)**

Directly assign data to a piece. Used for initialization and replay.

Parameter	Description
Pieces* piece	Pointer to the piece to assign
int player	Which player does this piece belong to
PieceType type	The type of the piece
int x	The x-coordinate
int y	The y-coordinate
int isEaten	Is this piece being “eaten”?

**void InitializeBehaviors(PieceBehavior\* pieceBehaviors)**

Builds an array of piece behavior. Contains predefined rules for all the pieces.

Parameter	Description
<b>PieceBehavior*</b> pieceBehaviors	The array of piece behaviors

---

**void InitializeBoard(Piece\* const pieces)**

Initializes the starting position for all the pieces.

Parameter	Description
<b>Piece* const</b> pieces	The array of all the pieces

---

**Piece\* GetPieceAtPosition(Piece\* pieces, int x, int y)**

Get the pointer to a piece at the given position. Returns NULL if not found.

Parameter	Description
<b>Piece* const</b> pieces	The array of all the pieces
<b>int</b> x	The x-coordinate of the piece
<b>int</b> y	The y-coordinate of the piece

**Returns:** A pointer to the found piece

---

**PieceBehavior\* GetPieceBehavior(PieceBehavior\* pieceBehaviors, PieceType type)**

Get the behavior object for a specific piece type. Returns NULL if not found.

Parameter	Description
<b>PieceBehavior*</b> pieceBehaviors	The array of all the piece behaviors
<b>PieceType</b> type	The target piece type

**Returns:** A pointer to the found piece behavior

```
int CheckMoveRule(Piece* pieces, PieceBehavior* pieceBehaviors, PieceType
type, int oldX, int oldY, int newX, int newY, int currentPlayer)
```

Check whether a move is valid.

Parameter	Description
<b>Pieces</b> * piece	Pointer to the piece to assign
<b>PieceBehavior</b> * pieceBehaviors	The array of all the piece behaviors
<b>PieceType</b> type	The type of the piece
<b>int</b> oldX	The x-coordinate before moving
<b>int</b> oldY	The y-coordinate before moving
<b>int</b> newX	The x-coordinate after moving
<b>int</b> newY	The y-coordinate after moving
<b>int</b> currentPlayer	Which player does this piece belong to

**Returns:** A boolean (**int**) indicating whether the move is valid

---

```
void MovePiece(Piece* pieces, int x, int y)
```

Move a piece to a new position.

Parameter	Description
<b>Piece</b> * <b>const</b> pieces	The array of all the pieces
<b>int</b> x	The target x-coordinate
<b>int</b> y	The target y-coordinate

## kifu.c

```
int IsListEmpty(KifuList* list)
```

Check if a kifu stack is empty.

Parameter	Description
KifuList* list	Pointer to the kifu list

**Returns:** A boolean (int) indicating whether the list is empty

---

```
void PushKifu(KifuList* list, const KifuRecord record)
```

Push a kifu record to the kifu stack, if the stack is not full.

Parameter	Description
KifuList* list	Pointer to the kifu list
const KifuRecord record	The record to be pushed

```
KifuStack* PopKifu(KifuList* const list)
```

Get the last kifu record from a kifu list, if the list is not empty.

Parameter	Description
KifuList* const list	Pointer to the kifu list

**Returns:** A pointer to the kifu record

**void WriteKifuToFile(const char\* fileName, KifuList\* const list)**

Write a kifu list to a text file.

Parameter	Description
<b>const char* fileName</b>	The target file name
<b>KifuList* const list</b>	Pointer to the kifu list

---

**void LoadKifuFromFile(const char\* fileName, KifuList\* const list)**

Load a kifu stack from a text file.

Parameter	Description
<b>const char* fileName</b>	The target file name
<b>KifuList* const list</b>	Pointer to the kifu list

---

**int CheckHasRevertibleSteps(KifuList\* const list, const int currentPlayer)**

Check if there are revertible steps for a specific player. Used for step reversion and replay.

Parameter	Description
<b>KifuList* const list</b>	Pointer to the kifu list
<b>const int currentPlayer</b>	The index of the player

**Returns:** A boolean (int) indicating if there are revertible steps

---

**void PrintNode(NodePtr const node)**

(Debug) prints the content of a list node.

Parameter	Description
<b>NodePtr const node</b>	Pointer to the target node



**void PrintList(KifuList\* const list)**

(Debug) prints the content of a kifu list.

Parameter	Description
<b>KifuList* const list</b>	Pointer to the target kifu list

---

## **replay.c**

**void ShogiReplay(char\* kifuFileName)**

Load a kifu stack from a text file.

Parameter	Description
<b>char* fileName</b>	The target file name

## Compiling

First, make sure the libev library is installed.

To compile the program, switch to the project directory. Use the following commands with gcc:

```
> gcc main.c -lm shogi.c -lm -lev shogilib.c -lm kifu.c -lm replay.c -lm  
-o shogi.out
```

A new file `shogi.out` will be generated.

## Running the Application

First, switch to the directory where the `shogi.out` file is located.

To start a new game:

```
> ./shogi.out -n -s {SAVEFILE NAME}
```

To replay an existing game:

```
> ./shogi.out -l {SAVEFILE NAME}
```

The arguments must be present, or the program will not run.

Demo Screenshots

New Game

歩

987654321

+

+

+

+

+

+

+

+

+

+

|

香

|

桂

|

銀

|

金

|

玉

|

金

|

銀

|

桂

|

香

|

1

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

角

|

|

|

|

|

|

飛

|

2

+

+

+

+

+

+

+

+

+

+

|

歩

|

歩

|

歩

|

歩

|

歩

|

歩

|

3

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

歩

|

5

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

歩

|

歩

|

歩

|

歩

|

歩

|

歩

|

7

+

+

+

+

+

+

+

+

+

+

|

|

角

|

|

|

|

|

飛

|

8

+

+

+

+

+

+

+

+

+

+

|

香

|

桂

|

銀

|

金

|

王

|

金

|

銀

|

桂

|

香

|

9

+

+

+

+

+

+

+

+

+

+

Moved 歩 to (2, 5)  
Time used: 2s  
  
Player 0:  
Select piece (x, y) >>

歩

987654321

+

+

+

+

+

+

+

+

+

+

|

香

|

桂

|

銀

|

金

|

玉

|

金

|

銀

|

桂

|

香

|

1

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

角

|

|

|

|

|

|

飛

|

2

+

+

+

+

+

+

+

+

+

+

|

歩

|

歩

|

歩

|

歩

|

歩

|

歩

|

3

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

歩

|

5

+

+

+

+

+

+

+

+

+

+

|

|

|

|

|

|

|

|

|

|

+

+

+

+

+

+

+

+

+

+

|

歩

|

歩

|

歩

|

歩

|

歩

|

歩

|

7

+

+

+

+

+

+

+

+

+

+

|

|

角

|

|

|

|

|

飛

|

8

+

+

+

+

+

+

+

+

+

+

|

香

|

桂

|

銀

|

金

|

王

|

金

|

銀

|

桂

|

香

|

9

+

+

+

+

+

+

+

+

+

+

[歩] -> Move piece (x, y) >> 1 1  
This move is illegal!

## Game Replay

9 8 7 6 5 4 3 2 1

香	桂	銀	金	玉	金	銀	桂	香	1
	角							飛	2
步	步	步	步	步	步	步	步	步	3
									4
									5
							步		6
步	步	步	步	步	步	步		步	7
	角							飛	8
香	桂	銀	金	王	金	銀	桂	香	9

REPLAY FORWARD

Options:  
Forward (p)  
Backward (b)

步

9 8 7 6 5 4 3 2 1

香	桂	銀	金	玉	金	銀	桂	香	1
	角							飛	2
步	步	步	步	步	步	步		步	3
									4
							步		5
									6
步	步	步	步	步	步	步		步	7
	角							飛	8
香	桂	銀	金	王	金	銀	桂	香	9

REPLAY FORWARD

Options:  
Backward (b)

## Save File

