スタックの実装と応用

22060 211 古城隆人

2024 年 6 月 25 日

目次

1　目的

ススタックを用いたデータ構造を理解し、スタックを用いたプログラムを作成することで、スタックの基本的な操作を理解する。また、スタックを用いたハノイの塔のプログラムを作成し、理解を深める。

2　原理

2.1　スタック

スタックとはデータ構造の一種であり、データの追加や削除が限定的に行うことが出来る構造である。限定的にすることによりデータの数が増えてもデータの挿入や削除にかかる時間が一定になるという利点がある。スタックはデータが出入りする順番により、LIFO(Last In First Out)、FILO(First in Last out) と呼ばれる。スタックには以下の操作がある。

- push: スタックにデータを追加する
- pop: スタックからデータを取り出す

push を行うと、スタックの一番上にデータが追加される。pop を行うと、スタックの一番上のデータが取り出され、取り出したデータを削除する。

2.2　ハノイの塔

ハノイの塔は、3 本の杭とその上に積まれた円盤からなるパズルである。移動のルールは以下の通りである。

- 1 回の移動で 1 枚の円盤しか動かせない
- 小さい円盤の上に大きい円盤を乗せることはできない
- すべての円盤を移動させるとき、最初の杭から最後の杭に移動させる

杭を stack とみなすことが出来るため 3 個の stack を用いて実装を行う。

3　実験環境

実験環境を表 1 に示す。

項目	値
OS	windows10 上の wsl2(Ubuntu)
CPU	Intel Core i7
メモリ	8GB
コンパイラ	gcc 11.4.0

表 1　実験環境

4　プログラムの設計

4.1　スタックの実装

スタックの実装をするために最初にソースコード 1 に示す構造体を作成する。また、次に作成する関数の引数としてこの構造体のポインタを渡すことで、スタックのデータを操作することが出来る。作成する関数は表 2 の通りである。

関数名	説明
init	スタックの構造体を初期化する
pop	スタックからデータを取り出す
push	スタックに値を代入する
printStack	スタックの中身を表示する
pushTest	push 関数のテストを行う
popTest	pop 関数のテストを行う

表 2　作成する関数のリスト

4.1.1　構造体

この構造体はスタックのデータを格納するためのものであり、データの格納数を HEIGHT で定義している。

ソースコード 1　stack.h

```
1  #define HEIGHT 5
2  struct Stack
3  {
4    int data[HEIGHT];
5    int volume;
6  };
```

4.1.2　作成する関数

- init: スタックの構造体を初期化する、関数の仕様は 表 3 に示す。
- push: スタックに値を代入する、関数の仕様は 表 4 に示す。
- pop: スタックからデータを取り出す、関数の仕様は 表 5 に示す。
- printStack: スタックの中身を表示する、関数の仕様は 表 6 に示す。
- pushTest: push 関数のテストを行う、関数の仕様は 表 7 に示す。
- popTest: pop 関数のテストを行う、関数の仕様は 表 8 に示す。

機能	スタック構造体の中にある配列を全部 0 で初期化する。
引数	struct Stack *stack：初期化するスタックのポインタ
戻り値	なし

表 3　init 関数

機能	スタックにデータを追加させる。
引数	struct Stack *stack：push したい stack のポインタ
戻り値	処理が正常に終了したら 1、エラーが出たら-1

表 4　push 関数

機能	スタックのデータを一番上から取り出して、そのデータを削除する。
引数	struct Stack *stack：pop したい stack のポインタ
戻り値	削除したデータの値 (int) を返すが、エラーが出たときは-1(int) を返す

表 5　pop 関数

機能	スタックの中身を表示する。
引数	struct Stack *stack：表示したい stack のポインタ
戻り値	なし

表 6　printStack 関数

機能	push 関数のテストを行う。成功したら push 関数の戻り値が 0 になるため、 if 文を使い結果を出力する
引数	struct Stack *stack：push したい stack のポインタ num: push する値
戻り値	なし

表 7　pushTest 関数

機能	pop 関数のテストを行う。pop に失敗したら pop 関数の戻り値が-1 になるため、 if 文を使い結果を出力する
引数	struct Stack *stack：pop したい stack のポインタ
戻り値	なし

表 8　popTest 関数

4.2　ハノイの塔

2.2 でも示した通り、ハノイの塔は 3 本の杭とその上に積まれた円盤からなるパズルである。そのため、3 個の stack を用いて実装を行う。ハノイの塔のために作成する関数は表 9 の通りである。

関数名	説明
enableStack	移動可能かどうかを判別する
checkFinish	終了したかどうかを判別する

表 9　作成する関数のリスト

4.2.1　作成する関数

- enableStack: 移動可能かどうかを判別する、関数の仕様は 表 10 に示す。
- checkFinish: 終了したかどうかを判別する、関数の仕様は 表 11 に示す。

機能	移動可能かどうかを判別する。移動可能なら 1 を返し、不可能なら-1 を返す。
引数	struct Stack stack1：移動元の stack のポインタ struct Stack stack2：移動先の stack のポインタ
戻り値	移動可能なら 1、不可能なら 0 を返す。

表 10　enableStack 関数

機能	終了したかどうかを判別する。終了したら 1 を返し、終了していないなら-1 を返す。
引数	struct Stack *stack1：終了判定する stack のポインタ struct Stack *stack2：終了判定する stack のポインタ
戻り値	終了したら 1、終了していないなら 0 を返す。

表 11　checkFinish 関数

5　プログラムの説明

前項で示した関数を用いて、スタックの実装とハノイの塔のプログラムを作成する。

5.1　スタックの実装

以下にスタックの実装に使用した関数に関するプログラムの抜粋を示す。仕様は前項で示した通りである。

- init 関数 – ソースコード 3
- push 関数 – ソースコード 4
- pop 関数 – ソースコード 5
- printStack 関数 – ソースコード 6
- pushTest 関数 – ソースコード 7
- popTest 関数 – ソースコード 8

また、これらの関数を用いて stack が実際に動作するかを確認するために、main 関数を作成し、pushTest 関数と popTest 関数を呼び出す。main 関数のソースコードをソースコード 2 に示す。スタックの範囲外の動作

をさせるために 6 回 push/pop している。

ソースコード 2　main 関数

```
1   int main()
2   {
3       struct Stack stack;
4       init(&stack);
5       pushTest(&stack, 10);
6       pushTest(&stack, 20);
7       pushTest(&stack, 30);
8       pushTest(&stack, 40);
9       pushTest(&stack, 50);
10      pushTest(&stack, 60);
11      popTest(&stack);
12      popTest(&stack);
13      popTest(&stack);
14      popTest(&stack);
15      popTest(&stack);
16      popTest(&stack);
17      return 0;
18  }
```

ソースコード 3　init 関数

```
1   void init(struct Stack *stack)
2   {
3       // スタックのすべての要素の値を 0 にする
4       // スタックに格納されているデータ数を 0 にする
5       for (int i = 0; i < HEIGHT; i++)
6       {
7           stack->data[i] = 0;
8       }
9       stack->volume = 0;
10  }
```

ソースコード 4　push 関数

```
1   int push(struct Stack *stack, int number)
2   {
3       // データを最上位に積み込む
4       // データの個数を増やす
5       if (stack->volume >= HEIGHT)
6       {
7           return -1;
8       }
9       stack->data[stack->volume] = number;
10      stack->volume++;
11      return 0;
12  }
```

ソースコード 5　pop 関数

```
1   int pop(struct Stack *stack)
2   {
3       // 格納されているデータ個数のカウントを減らす
4       // 取り出すデータを取り出す
5       // 取り出した場所を初期化する
6       if (stack->volume == 0)
7       {
8           return -1;
9       }
10      stack->volume--;
11      int result = stack->data[stack->volume];
12      stack->data[stack->volume] = 0;
13      return result;
14  }
```

ソースコード 6　printStack 関数

```
1   void printStack(struct Stack stack)
2   {
3       // スタックに格納されている値をスタックされている順番に 1 行に表示
4       for (int i = stack.volume - 1; i >= 0; i--)
5       {
6           printf("%d ", stack.data[i]);
7       }
8       printf("\n");
9   }
```

ソースコード 7　pushTest 関数

```
1   void pushTest( struct Stack *stack,int num)
2   {
3       printf("push (%d) ",num);
4       if(push(stack, num) == 0){
5           printf("SUCCESS\n");
6       }else{
7           printf("FAILURE\n");
8       }
9       printf("data : ");
10      printStack(*stack);
11  }
```

ソースコード 8　popTest 関数

```
1   void popTest( struct Stack *stack)
2   {
3       printf("pop ");
4       int result = pop(stack);
5       printf("(%d) ",result);
6       if(result == -1){
```

```
 7            printf("FAILURE\n");
 8        }else{
 9            printf("SUCCESS\n");
10        }
11        printf("data : ");
12        printStack(*stack);
13    }
```

5.2 ハノイの塔

以下にハノイの塔の実装に使用した関数に関するプログラムの抜粋を示す。仕様は前項で示した通りである。

- enableStack 関数 – ソースコード 10
- checkFinish 関数 – ソースコード 11

また、これらの関数を用いてハノイの塔のプログラムを作成する。ハノイの塔のプログラムの main 関数をソースコード 9 に示す。
main 関数に実装した移動禁止の条件とその時の動作は以下の通りである。もし移動禁止になった場合は移動できませんと表示されて今の盤面とともにもう一度入力を促す文字が表示されるようになっている。

- 入力された数字が 1 以上 3 以下の場合、移動元の塔と移動先の塔を関数に入力する
- 移動元の塔が空の場合、移動禁止
- 移動先の塔が空の場合、移動可能
- 移動元の塔の一番上のブロックが移動先の塔の一番上のブロックより小さい場合、移動可能
- それ以外の場合、移動禁止

ソースコード 9　main 関数

```
 1  int main()
 2  {
 3      int i;
 4      int count = 1;
 5      int fromNumber, toNumber;
 6      int tempNumber;
 7      int blocks;
 8      struct Stack tower[TOWERS];
 9
10      printf("段数を選んでください : 3,4,5:");
11      scanf("%d", &blocks);
12      /*3 塔を初期化する*/
13      init(&tower[0]);
14      init(&tower[1]);
15      init(&tower[2]);
16      /*第１塔に決められた個数をスタックする*/
17      for (i = 0; i < blocks; i++)
18      {
```

```
19          push(&tower[0], blocks - i);
20      }
21      /*塔の初期状態を表示する*/
22      for (i = 0; i < TOWERS; i++)
23      {
24          printf("%d : ", i + 1);
25          printStack(tower[i]);
26      }
27      while (1)
28      {
29          // 今，何回目の移動であるかを数える．
30          printf("count : %d\n", count);
31
32          // 移動元と移動先を受け取る
33          printf("移動元塔と移動先塔を入力してください。[? ?]:");
34          scanf("%d %d", &fromNumber, &toNumber);
35
36          if (fromNumber >= 0 && toNumber >= 0)
37          {
38              // 移動元の塔から移動先の塔にデータを移動させる
39              if (enableStack(tower[fromNumber - 1], tower[toNumber - 1]))
40              {
41                  tempNumber = pop(&tower[fromNumber - 1]);
42                  push(&tower[toNumber - 1], tempNumber);
43                  count++;
44              }
45              else
46              {
47                  printf("移動できません\n");
48              }
49          }
50          else
51          {
52              printf("移動できません\n");
53          }
54
55          // 現在の塔の状態を表示する
56          for (i = 0; i < TOWERS; i++)
57          {
58              printf("%d : ", i + 1);
59              printStack(tower[i]);
60          }
61          // クリア判定をする
62          if (checkFinish(tower[2], blocks))
63          {
64              printf("クリア\n");
65              break;
66          }
67      }
```

```
68  }
```

ソースコード 10　enableStack 関数

```
1   int enableStack(struct Stack fromTower, struct Stack toTower)
2   {
3       /* 移動可能である条件に応じて返り値を返す */
4       if (fromTower.volume == 0)
5       {
6           return 0;
7       }
8       else if (toTower.volume == 0)
9       {
10          return 1;
11      }
12      else if (top(fromTower) < top(toTower))
13      {
14          return 1;
15      }
16      else
17      {
18          return 0;
19      }
20  }
```

ソースコード 11　checkFinish 関数

```
1   int checkFinish(struct Stack tower, int blocks)
2   {
3       // ブロックが初期状態と同じ状態かチェックする
4       for (int i = 0; i < blocks; i++)
5       {
6           if (tower.data[i] != blocks - i)
7           {
8               return 0;
9           }
10      }
11      return 1;
12  }
```

6　実行結果

6.1　スタックの実装

main 関数を実行した結果を以下に示す。

```
push (10) SUCCESS
data : 10
```

```
push (20) SUCCESS
data : 20 10
push (30) SUCCESS
data : 30 20 10
push (40) SUCCESS
data : 40 30 20 10
push (50) SUCCESS
data : 50 40 30 20 10
push (60) FAILURE
data : 50 40 30 20 10
pop (50) SUCCESS
data : 40 30 20 10
pop (40) SUCCESS
data : 30 20 10
pop (30) SUCCESS
data : 20 10
pop (20) SUCCESS
data : 10
pop (10) SUCCESS
data :
pop (-1) FAILURE
data :
```

6.2 ハノイの塔

ハノイの塔のプログラムを実行した結果を以下に示す。また、入力エラーをわざと起こしたときの実行結果を以下に示す。

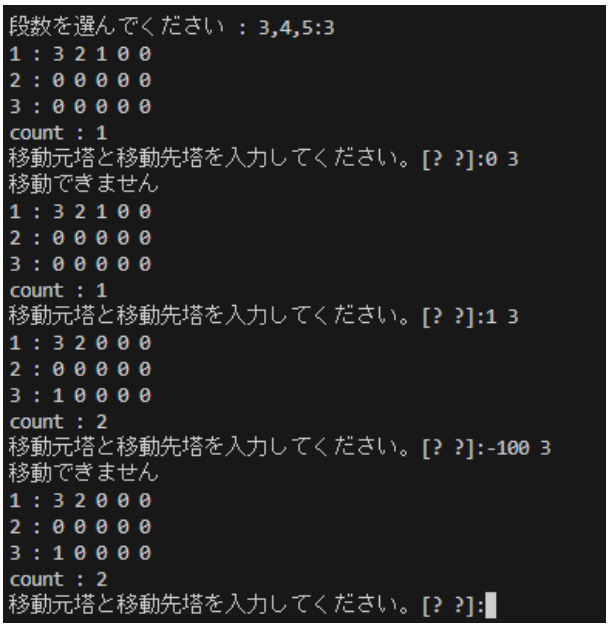

図１　ファイルが存在するときの実行結果

段数を選んでください：3,4,5:5

1：5 4 3 2 1

2：0 0 0 0 0

3：0 0 0 0 0

count：1

移動元塔と移動先塔を入力してください。[? ?]:1 2

1：5 4 3 2 0

2：1 0 0 0 0

3：0 0 0 0 0

count：2

移動元塔と移動先塔を入力してください。[? ?]:1 3

1：5 4 3 0 0

2：1 0 0 0 0

3：2 0 0 0 0

count：3

移動元塔と移動先塔を入力してください。[? ?]:2 3

1：5 4 3 0 0

2：0 0 0 0 0

3：2 1 0 0 0

count：4

移動元塔と移動先塔を入力してください。[? ?]:1 2

1：5 4 0 0 0

2：3 0 0 0 0
3：2 1 0 0 0
count：5
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：5 4 1 0 0
2：3 0 0 0 0
3：2 0 0 0 0
count：6
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：5 4 0 0 0
2：3 1 0 0 0
3：2 0 0 0 0
count：7
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：5 4 2 0 0
2：3 1 0 0 0
3：0 0 0 0 0
count：8
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：5 4 2 1 0
2：3 0 0 0 0
3：0 0 0 0 0
count：9
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：5 4 2 1 0
2：0 0 0 0 0
3：3 0 0 0 0
count：10
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：5 4 2 0 0
2：1 0 0 0 0
3：3 0 0 0 0
count：11
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：5 4 0 0 0
2：1 0 0 0 0
3：3 2 0 0 0
count：12
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：5 4 0 0 0

2：0 0 0 0 0
3：3 2 1 0 0
count：13
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：5 0 0 0 0
2：4 0 0 0 0
3：3 2 1 0 0
count：14
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：5 1 0 0 0
2：4 0 0 0 0
3：3 2 0 0 0
count：15
移動元塔と移動先塔を入力してください。[? ?]:3 2
1：5 1 0 0 0
2：4 2 0 0 0
3：3 0 0 0 0
count：16
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：5 0 0 0 0
2：4 2 0 0 0
3：3 1 0 0 0
count：17
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：5 2 0 0 0
2：4 0 0 0 0
3：3 1 0 0 0
count：18
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：5 2 1 0 0
2：4 0 0 0 0
3：3 0 0 0 0
count：19
移動元塔と移動先塔を入力してください。[? ?]:3 2
1：5 2 1 0 0
2：4 3 0 0 0
3：0 0 0 0 0
count：20
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：5 2 0 0 0

2：4 3 0 0 0
3：1 0 0 0 0
count：21
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：5 0 0 0 0
2：4 3 2 0 0
3：1 0 0 0 0
count：22
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：5 1 0 0 0
2：4 3 2 0 0
3：0 0 0 0 0
count：23
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：5 0 0 0 0
2：4 3 2 1 0
3：0 0 0 0 0
count：24
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：0 0 0 0 0
2：4 3 2 1 0
3：5 0 0 0 0
count：25
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：1 0 0 0 0
2：4 3 2 0 0
3：5 0 0 0 0
count：26
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：1 0 0 0 0
2：4 3 0 0 0
3：5 2 0 0 0
count：27
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：0 0 0 0 0
2：4 3 0 0 0
3：5 2 1 0 0
count：28
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：3 0 0 0 0

2：4 0 0 0 0
3：5 2 1 0 0
count：29
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：3 1 0 0 0
2：4 0 0 0 0
3：5 2 0 0 0
count：30
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：3 0 0 0 0
2：4 1 0 0 0
3：5 2 0 0 0
count：31
移動元塔と移動先塔を入力してください。[? ?]:3 1
1：3 2 0 0 0
2：4 1 0 0 0
3：5 0 0 0 0
count：32
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：3 2 1 0 0
2：4 0 0 0 0
3：5 0 0 0 0
count：33
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：3 2 1 0 0
2：0 0 0 0 0
3：5 4 0 0 0
count：34
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：3 2 0 0 0
2：1 0 0 0 0
3：5 4 0 0 0
count：35
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：3 2 0 0 0
2：0 0 0 0 0
3：5 4 1 0 0
count：36
移動元塔と移動先塔を入力してください。[? ?]:1 2
1：3 0 0 0 0

2：2 0 0 0 0
3：5 4 1 0 0
count：37
移動元塔と移動先塔を入力してください。[? ?]:3 2
1：3 0 0 0 0
2：2 1 0 0 0
3：5 4 0 0 0
count：38
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：0 0 0 0 0
2：2 1 0 0 0
3：5 4 3 0 0
count：39
移動元塔と移動先塔を入力してください。[? ?]:2 1
1：1 0 0 0 0
2：2 0 0 0 0
3：5 4 3 0 0
count：40
移動元塔と移動先塔を入力してください。[? ?]:2 3
1：1 0 0 0 0
2：0 0 0 0 0
3：5 4 3 2 0
count：41
移動元塔と移動先塔を入力してください。[? ?]:1 3
1：0 0 0 0 0
2：0 0 0 0 0
3：5 4 3 2 1
クリア

7　考察

7.1　スタックの実装

スタックの実験結果より、満杯のスタックにさらに push しようとしたときにエラー表示となり中身を壊すことなく処理を終えることが出来ている。また、中身がないときに pop しようとしてもエラーが出力されている。このことからスタックの基本的な動作が期待通りに行われていることがわかる。

7.2 ハノイの塔

8 付録: 今回使用したプログラム

ソースコード 12 stack.c

```c
#include <stdio.h>
#define HEIGHT 5
struct Stack
{
    int data[HEIGHT];
    int volume;
};

void init(struct Stack *stack)
{
    // スタックのすべての要素の値を 0 にする
    // スタックに格納されているデータ数を 0 にする
    for (int i = 0; i < HEIGHT; i++)
    {
        stack->data[i] = 0;
    }
    stack->volume = 0;
}


int push(struct Stack *stack, int number)
{
    // データを最上位に積み込む
    // データの個数を増やす
    if (stack->volume >= HEIGHT)
    {
        return -1;
    }
    stack->data[stack->volume] = number;
    stack->volume++;
    return 0;
}

int pop(struct Stack *stack)
{
    // 格納されているデータ個数のカウントを減らす
    // 取り出すデータを取り出す
    // 取り出した場所を初期化する
    if (stack->volume == 0)
    {
        return -1;
    }
```

```
43      stack->volume--;
44      int result = stack->data[stack->volume];
45      stack->data[stack->volume] = 0;
46      return result;
47  }
48
49  void printStack(struct Stack stack)
50  {
51      // スタックに格納されている値をスタックされている順番に 1 行に表示
52      for (int i = stack.volume - 1; i >= 0; i--)
53      {
54          printf("%d ", stack.data[i]);
55      }
56      printf("\n");
57  }
58
59  void pushTest( struct Stack *stack,int num)
60  {
61      printf("push (%d) ",num);
62      if(push(stack, num) == 0){
63          printf("SUCCESS\n");
64      }else{
65          printf("FAILURE\n");
66      }
67      printf("data : ");
68      printStack(*stack);
69  }
70
71  void popTest( struct Stack *stack)
72  {
73      printf("pop ");
74      int result = pop(stack);
75      printf("(%d) ",result);
76      if(result == -1){
77          printf("FAILURE\n");
78      }else{
79          printf("SUCCESS\n");
80      }
81      printf("data : ");
82      printStack(*stack);
83  }
84
85  int main()
86  {
87      struct Stack stack;
88      init(&stack);
89      pushTest(&stack,10);
90      pushTest(&stack,20);
91      pushTest(&stack,30);
```

```
92      pushTest(&stack,40);
93      pushTest(&stack,50);
94      pushTest(&stack,60);
95      popTest(&stack);
96      popTest(&stack);
97      popTest(&stack);
98      popTest(&stack);
99      popTest(&stack);
100     popTest(&stack);
101     return 0;
102 }
```

ソースコード 13　tower.c

```
1   #include <stdio.h>
2   #define HEIGHT 5
3   #define TOWERS 3
4
5   struct Stack
6   {
7       int data[HEIGHT];
8       int volume;
9   };
10
11  void init(struct Stack *stack)
12  {
13      // スタックのすべての要素の値を 0 にする
14      // スタックに格納されているデータ数を 0 にする
15      for (int i = 0; i < HEIGHT; i++)
16      {
17          stack->data[i] = 0;
18      }
19      stack->volume = 0;
20  }
21
22  int push(struct Stack *stack, int number)
23  {
24      // データを最上位に積み込む
25      // データの個数を増やす
26      if (stack->volume >= HEIGHT)
27      {
28          return -1;
29      }
30      stack->data[stack->volume] = number;
31      stack->volume++;
32      return 0;
33  }
34
35  int pop(struct Stack *stack)
```

```
36  {
37      // 格納されているデータ個数のカウントを減らす
38      // 取り出すデータを取り出す
39      // 取り出した場所を初期化する
40      if (stack->volume == 0)
41      {
42          return -1;
43      }
44      stack->volume--;
45      int result = stack->data[stack->volume];
46      stack->data[stack->volume] = 0;
47      return result;
48  }
49
50  void printStack(struct Stack stack)
51  {
52      // スタックに格納されている値をスタックされている順番に 1 行に表示
53      for (int i = 0; i < HEIGHT; i++)
54      {
55          printf("%d ", stack.data[i]);
56      }
57      printf("\n");
58  }
59
60  int top(struct Stack tower)
61  {
62      return tower.data[tower.volume - 1];
63  }
64  int enableStack(struct Stack fromTower, struct Stack toTower)
65  {
66      /* 移動可能である条件に応じて返り値を返す */
67      if (fromTower.volume == 0)
68      {
69          return 0;
70      }
71      else if (toTower.volume == 0)
72      {
73          return 1;
74      }
75      else if (top(fromTower) < top(toTower))
76      {
77          return 1;
78      }
79      else
80      {
81          return 0;
82      }
83  }
84
```

```
85  int checkFinish(struct Stack tower, int blocks)
86  {
87      // ブロックが初期状態と同じ状態かチェックする
88      for (int i = 0; i < blocks; i++)
89      {
90          if (tower.data[i] != blocks - i)
91          {
92              return 0;
93          }
94      }
95      return 1;
96  }
97
98  int main()
99  {
100     int i;
101     int count = 1;
102     int fromNumber, toNumber;
103     int tempNumber;
104     int blocks;
105     struct Stack tower[TOWERS];
106
107     printf("段数を選んでください : 3,4,5:");
108     scanf("%d", &blocks);
109     /*3 塔を初期化する*/
110     init(&tower[0]);
111     init(&tower[1]);
112     init(&tower[2]);
113     /*第１塔に決められた個数をスタックする*/
114     for (i = 0; i < blocks; i++)
115     {
116         push(&tower[0], blocks - i);
117     }
118     /*塔の初期状態を表示する*/
119     for (i = 0; i < TOWERS; i++)
120     {
121         printf("%d : ", i + 1);
122         printStack(tower[i]);
123     }
124     while (1)
125     {
126         // 今，何回目の移動であるかを数える．
127         printf("count : %d\n", count);
128
129         // 移動元と移動先を受け取る
130         printf("移動元塔と移動先塔を入力してください。[? ?]:");
131         scanf("%d %d", &fromNumber, &toNumber);
132
133         if (fromNumber >= 1 && fromNumber <= 3 && toNumber >= 1 && toNumber <= 3)
```

```
134         {
135             // 移動元の塔から移動先の塔にデータを移動させる
136             if (enableStack(tower[fromNumber - 1], tower[toNumber - 1]))
137             {
138                 tempNumber = pop(&tower[fromNumber - 1]);
139                 push(&tower[toNumber - 1], tempNumber);
140                 count++;
141             }
142             else
143             {
144                 printf("移動できません\n");
145             }
146         }
147         else
148         {
149             printf("移動できません\n");
150         }
151 
152         // 現在の塔の状態を表示する
153         for (i = 0; i < TOWERS; i++)
154         {
155             printf("%d : ", i + 1);
156             printStack(tower[i]);
157         }
158         // クリア判定をする
159         if (checkFinish(tower[2], blocks))
160         {
161             printf("クリア\n");
162             break;
163         }
164     }
165 }
```