



# 情報実験第一

2012年6月27日(E) 2012年6月28日(O)

# 大課題について

- 課題1と課題2それぞれについてプログラムとレポートを提出
- プログラムの先頭にどのように実行するか簡単な説明を加えること
- 複数のファイルを提出したい場合, zipもしくはtar.gz形式で圧縮して提出すること
  - zipでの圧縮: `%zip compress.zip file1 file2 ...`
  - Tarでの圧縮: `%tar zcvf compress.tar.gz file1 file2 ...`

# 大課題問題

『ぬりかべ』パズルに関するプログラムを作成せよ

- 課題1:

ユーザがパズルを遊ぶためのプログラムを作成せよ

- 課題2:

パズルの正解を自動的に解くプログラムを作成せよ

# ぬりかべ

## 『ぬりかべパズル』のルール

1. 数字のあるマスは壁(黒マス)にならない
2. 数字は壁に囲まれた壁以外の領域(シマと呼ぶ)の面積であり、各シマには数字が一つずつ入る
3. すべての壁は縦横に一つながりになる
4. 壁が $2 \times 2$ 以上の固まりになってはいけない

# パズルの例

	1		3
			3

問題

	1		3
			3

正解

	1		3
			3

不正解

- シマに2つ以上の数字が入っている
- 数字とシマの面積が不一致

	1		3
			3

不正解

- シマに2つ以上の数字が入っている
- 壁が2x2以上の固まりになっている

# パズルの例2

			1
3			
		3	
1			

問題

			1
3			
		3	
1			

正解

			1
3			
		3	
1			

不正解

•すべての壁が縦横に一つながりになっていない

# 課題1

- ユーザがパズルを遊ぶためのプログラムを作成せよ
  - 問題はファイルから読み込むこと
  - 入力毎にルール違反を確認し、ルールに違反していた場合は再入力を促すこと
  - パズルが解けたかどうかを判定し、解けていた場合はその旨を出力すること
  - パズルのサイズは最大で36x36とする
  - マスの数字は最大で9とする

# 問題のフォーマット

- 一行目はパズルのサイズ  
#SizeX SizeY
- 2行目以降はパズル問題の情報
  - 0: 白マス

	1		
			2
	1		
1			1



```
#4 4  
0100  
0002  
0100  
1001
```



# プログラムの実行例

```
-1--  
---2  
-1--  
1--1
```

最初に問題を出力

```
Input wall position (x,y): 1,1
```

壁の座標を入力

```
-1--  
-*-2  
-1--  
1--1
```

```
Input wall position (x,y): 1,1
```

すでに壁である座標が  
入力された場合は壁か  
ら白マスに戻す

```
-1--  
---2  
-1--  
1--1
```

# プログラムの実行例2

```
*1*-  
***2  
*1**  
1*-1
```

```
Input wall position (x,y): 0,1  
[Against the Rule] You cannot put a  
block at a cell with a number.
```

ルールに違反した入  
力はルール違反の出  
力後、再入力

```
*1*-  
***2  
*1**  
1*-1
```

```
Input wall position (x,y): 3,2
```

```
=====  
### You solve the puzzule! ###  
=====
```

パズルが解けたらそ  
の旨を出力

```
*1*-  
***2  
*1**  
1**1
```

# 課題2

- パズルの正解を自動的に解くプログラムを作成せよ
  - 問題はファイルから読み込むこと
  - パズルのサイズは最大で36x36とする
  - マスの数字は最大で9とする
  - 採点時には実行速度、メモリ消費量を考慮する
    - 単なる全探索のプログラムでは実行時間が遅くなる(36x36マスのパズルでは $2^{(36 \times 36)}$ の探索が必要)ので、出来るだけ工夫をすること

# プログラムを書く上での注意


- ソースファイルの冒頭のコメント部分には、どの課題に対するプログラムなのか必ず明記する
- インデント(字下げ)をきちんとつける
  - インデントのないソースは、採点の対象外とするので注意すること
- 関数定義や構造体の定義などには、それらの簡単な説明となるコメントをつける
  - コメントのないソースファイルは採点の対象外とする場合もあるので注意すること。

# 締切

■ Eクラス7月18日 17:00

■ Oクラス7月19日 17:00

※レポートの書き方については次週説明



# 参考：課題 1 Step by Step

- 課題1について何から初めて良いのかわからない人は、以下の順番で各機能を実装していくこと
  1. パズルの読み込みと表示
  2. 配置する壁の座標の入力処理
  3. 配置された壁に対するルール違反の確認
  4. パズル終了判定

# 参考：パズルの読み込みと表示

- パズルのファイル名は実行時に引数として読み込めた方がよい  
→ コマンドライン引数を使う

コマンドライン  
引数の数

コマンドライン  
引数の内容

```
int main(int argc, char **argv) {  
    char *fname;  
  
    if (argc != 2) {  
        fprintf(stderr, "Usage: %s puzzle_file\n", argv[0])  
    }  
    fname = argv[1];  
    load_puzzle(fname, ...); //パズルの読み込み  
    print_board(...);       //盤面の表示  
    ...  
}
```

```
% ./a.out sample4x4.pzl
```

## 参考：配置する壁の座標の入力処理

- scanf関数では適切なエラー処理が行われない  
→ 想定外の入力があると暴走する
- scanf関数は返値として読み込めた変数の数を返す
- 異常な値が入力であった場合はscanf ("%\*s") によってバッファをクリアする

```
while(1){  
    printf("Input wall position (x,y): ");  
    if(scanf("%d,%d", &x, &y)!=2){  
        scanf("%*s");  
        printf("Illegal inputs\n");  
        continue;  
    }  
    ...  
}
```



## 参考：配置された壁に対するルール違反の確認

ぬりかべパズルは以下の2つの規則に関しては毎回の座標入力について判定可能

1. 数字のあるマスは壁（黒マス）にならない
2. 壁が $2 \times 2$ 以上の固まりになってはいけない



規則違反の入力で合った場合は再入力を促す

## 参考：パズル終了判定

毎座標入力時に2つの規則違反の判定をしていれば、パズルが解けたかどうかは以下の2つの条件が満たされているかどうかで判定可能

1. 数字は壁に囲まれた壁以外の領域(シマと呼ぶ)の面積であり、各シマには数字が一つずつ入る
2. すべての壁は縦横に一つながりになる