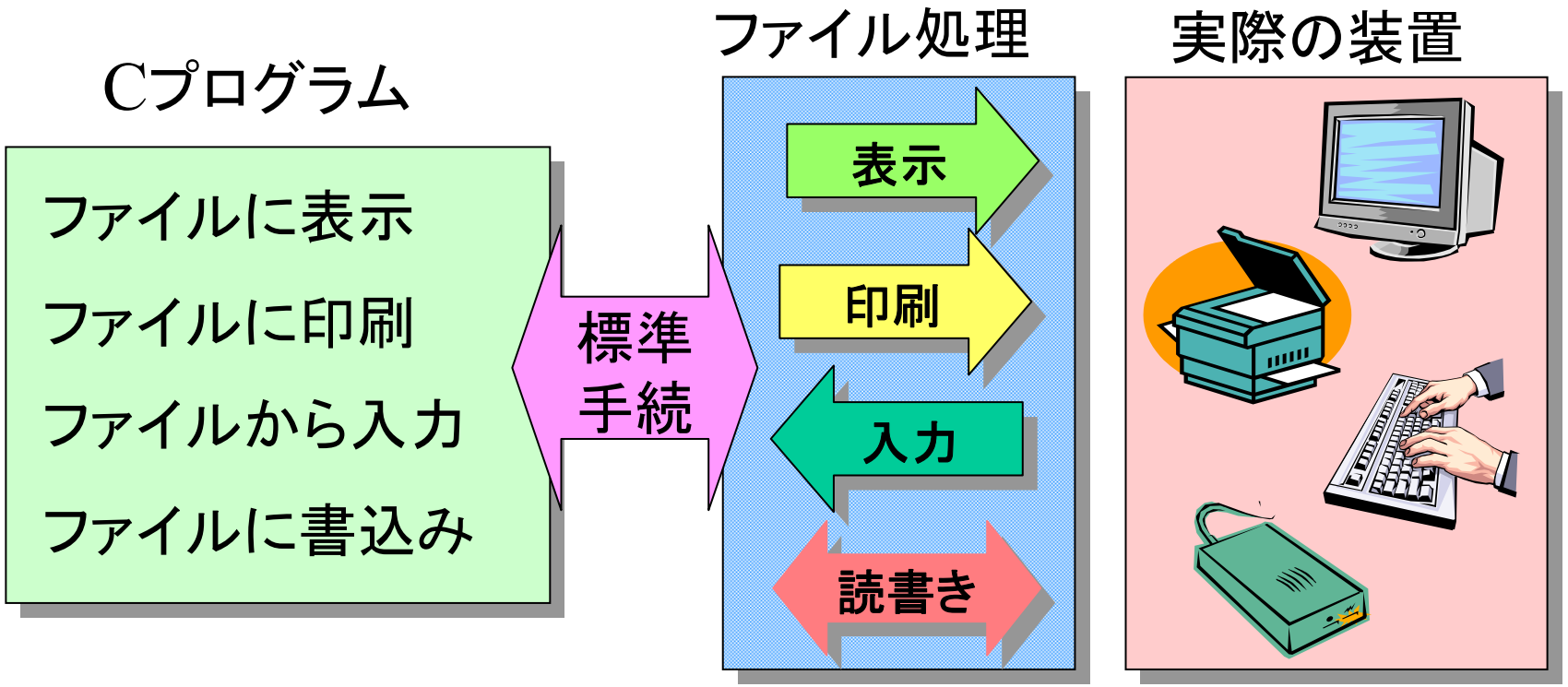
プログラミング入門
第12回講義

- リダイレクトとパイプ
  - ファイル
  - リダイレクション
  - パイプ
  - 応用例

 マークのあるサンプルプログラムは
`/home/course/prog0/public_html/2006/lec/source/`
下に置いてありますから、各自自分のディレクトリに
コピーして、コンパイル・実行してみてください

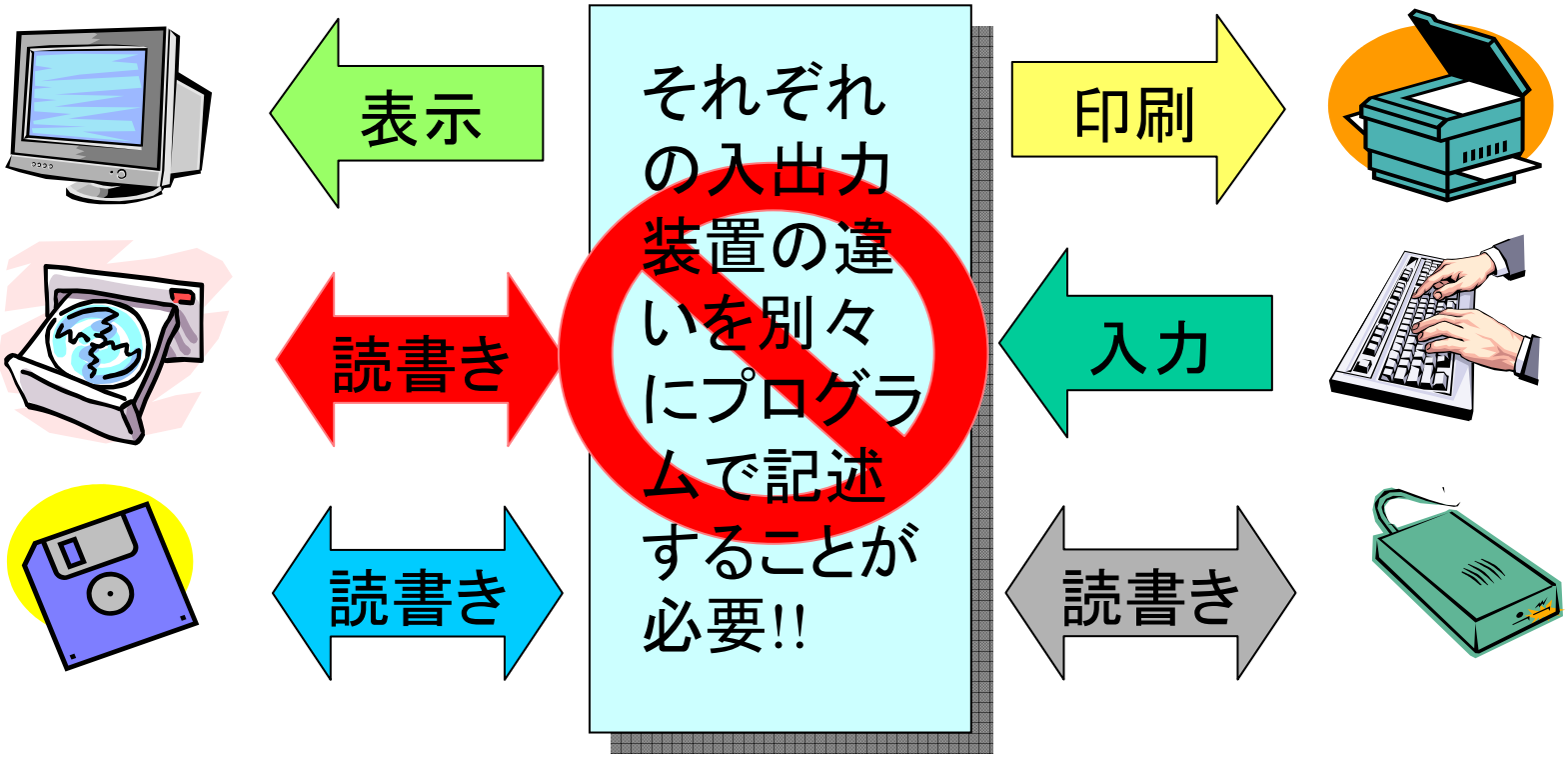
ファイル(p.326)

ファイル：あらゆる入出力データ・装置を論理化する概念
（実際のデータ記録装置，記録方式，記録形式等に依存しない処理環境の実現）



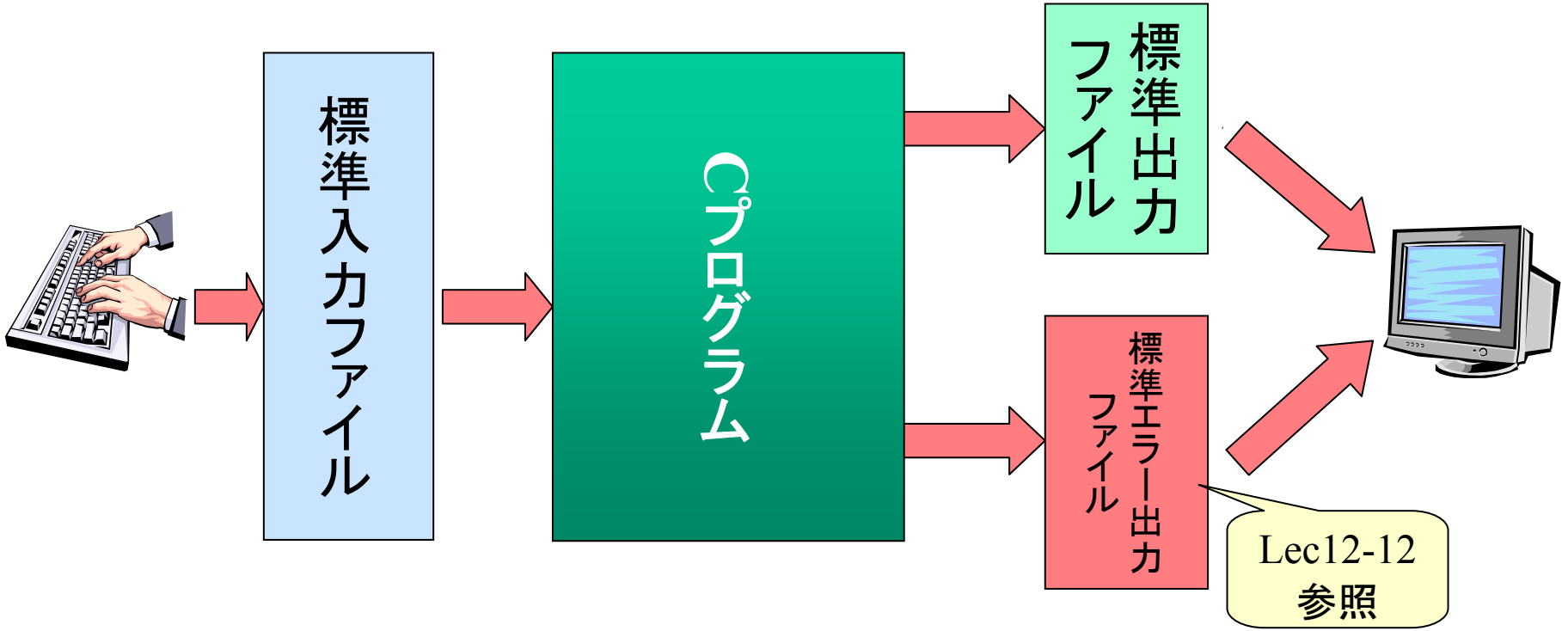
ファイル

もし、ファイルの概念が無かったら!?



Cプログラムの流れ

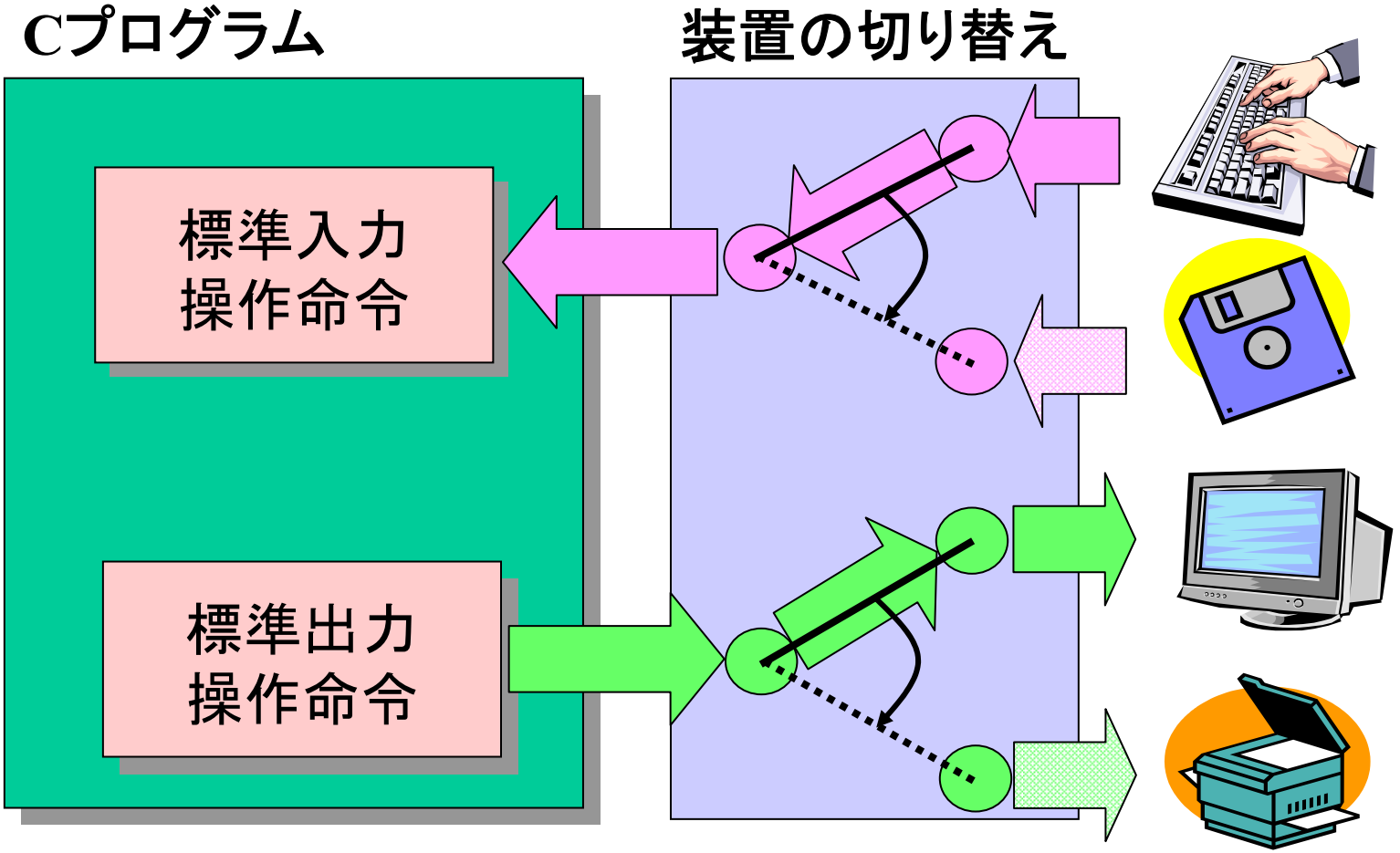
scanf(),printf()はそれぞれ標準入力ファイル（キーボード）、標準出力ファイル（ディスプレイ）に対応



標準入出力とリダイレクション (p.365)

- リダイレクションとは
  - 標準入出力ファイルの切り替えをプログラムの外で行う機能
- リダイレクションの例
  - キーボード入力 → ディスク入力に変更
  - 画面表示 → ディスク出力に変更

リダイレクション(p.20)



標準入出力のリダイレクションの書式

実行ファイル名　＜　入力ファイル名

例: プログラム ./a.out の標準入力を test.dat から読み込む

```
./a.out  <  test.dat
```

実行ファイル名　＞　出力ファイル名
実行ファイル名　＞＞　出力ファイル名

例: ./a.out の標準出力を test.dat に書き込む

```
./a.out  > test.dat
./a.out  >> test.dat
```

先頭から書込み

末尾に追加書き

リダイレクションの例(1)

例１. ファイル一覧をファイルlist.txtに書込む

```
ls > list.txt
```

例２. list.txtの行,単語,文字数を表示する

```
wc < list.txt
```

例３. list.txtから文字列「file」を含む行だけfile.txtに書き込む

```
grep file < list.txt > file.txt
```

リダイレクションの例(2)

リダイレクションでファイル中のデータの平均を計算する

```
#include <stdio.h>
#include <stdlib.h>

main()
{
  int i, data, result, sum = 0;
  for(i = 0;; i++){
    result = scanf("%d",&data);
    if (result == EOF) break;
    if (result != 1) exit(1);
    sum += data;
  }
  printf("Average = %f\n",(float)sum/i);
}
```

```
std1dc1{s1000000}1: ./a.out
1 2 3 4 5
2 3 4 5 6
Control+D
Average = 3.500000
std1dc1{s1000000}2: ./a.out < lec12-1.data
Average = 3.500000
std1dc1{s1000000}3: cat lec12-1.data
1 2 3 4 5
2 3 4 5 6
std1dc1{s1000000}4:
```

Control+Dは入力の終わりを示す

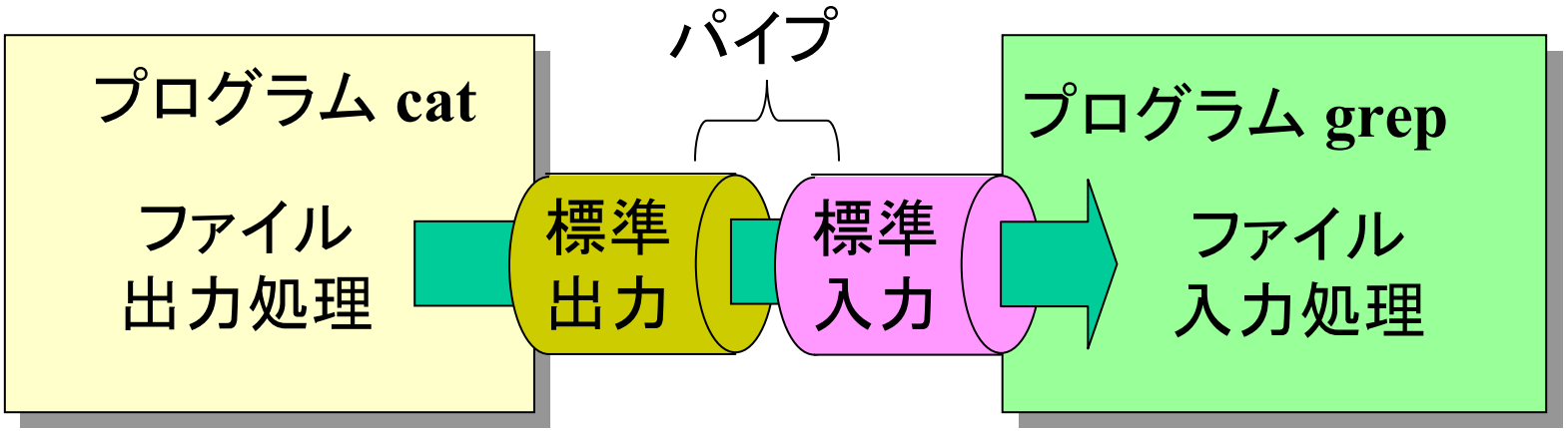
/home/course/prog0/public_html/2006/lec/source/lec12-1.c

標準入出力とパイプ(p.21)

- パイプとは
  - 二つのプログラムの標準出力と標準入力を結合する機能
    コマンド間に「|」を挟む
- パイプの例

```
cat ex1.c | grep 'printf'
```

  - catの出力が、grepの入力となる



パイプの例

例１. ディレクトリ内のファイルをファイル容量の大きい順に表示

```
ls -l | sort -k 4,4 -nr
```

例２. そのディレクトリ内のファイルの数を数える

```
ls | wc -l
```

例３. result.txt(IDと点数のペアデータ) から得点の高い順に１０人表示する

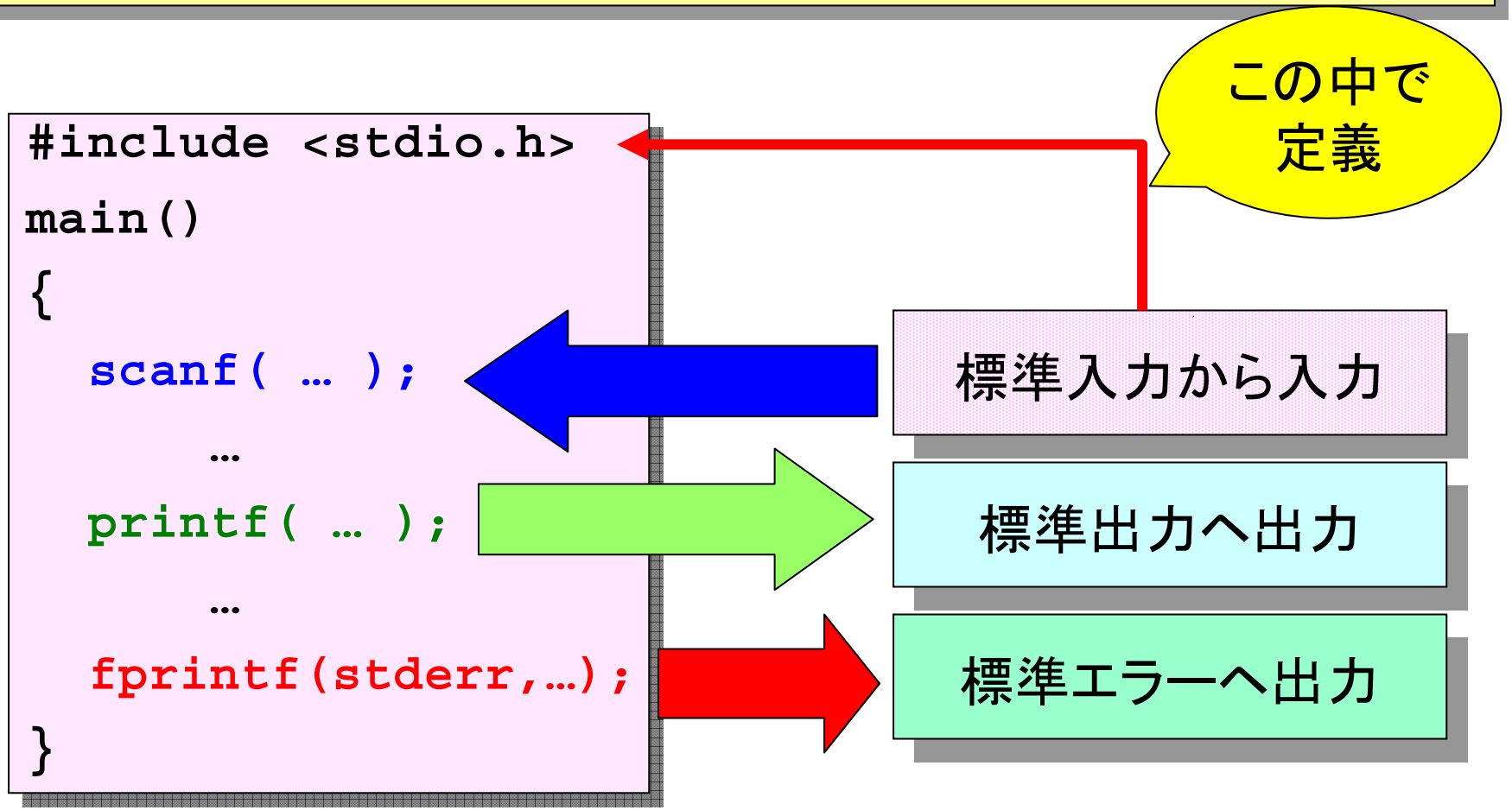
```
cat result.txt | sort -k 2,2 -nr | head -10
```

標準エラー出力(p.47,349)

- プログラムの出力をリダイレクトやパイプする場合に、エラーメッセージを標準出力に出力していると、エラーメッセージも画面に表示されず、リダイレクトやパイプの対象になってしまう。
- このためエラー出力用の標準ファイルを別に設け(標準エラー出力と呼ぶ)、リダイレクト・パイプの場合も画面にエラーメッセージを表示するようにしている。標準エラー出力への出力は以下のようにして行う

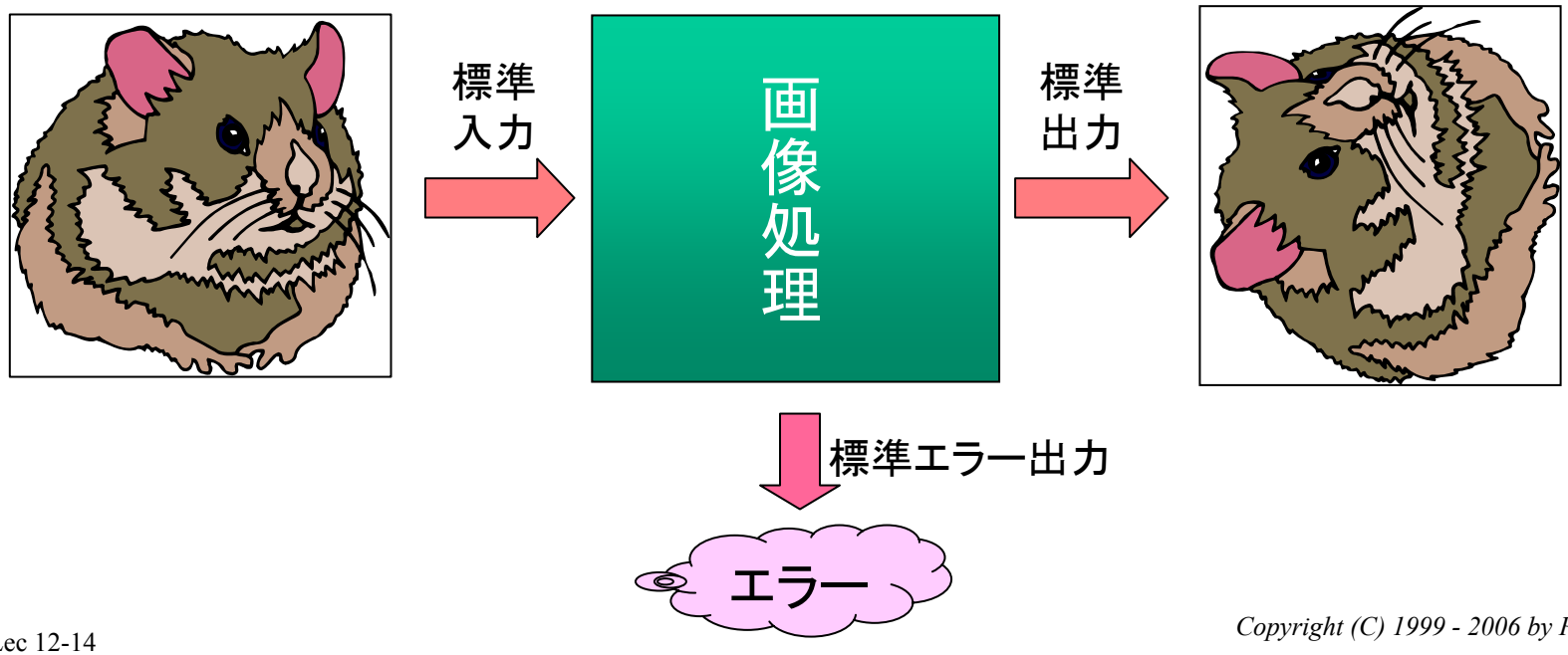
```
fprintf(stderr,"書式",変数リスト);

例:fprintf(stderr,"入力した値がおかしい¥n");
```

プログラム内の標準入出力と標準エラー (p.47)



リダイレクト・パイプの例
簡単な画像処理

- 画像を標準入力から入力
- 入力された画像を処理（例えば左に90度回転）
- 画像を標準出力から出力
- エラー情報は標準エラー出力により表示

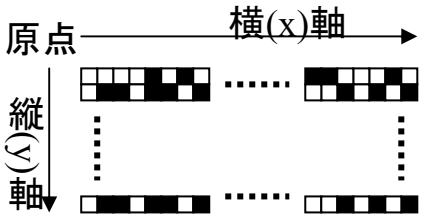


簡単な画像形式
Plain PBM(Portable Bitmap)形式

必ず「P1」(magic number)

```
P1
320 160
0 0 0 0 1 0 1 0 ..... 1 1 0 0 0 1 0
0 1 1 0 1 1 0 1 ..... 0 0 1 0 1 0 1
0 0 0 0 1 0 1 0 ..... 1 1 0 0 0 1 0
.....
.....
0 1 1 0 1 1 0 1 ..... 0 0 1 0 1 0 1
0 0 0 0 1 0 1 0 ..... 1 1 0 0 0 1 0
0 1 1 0 1 1 0 1 ..... 0 0 1 0 1 0 1
```

画像の大きさ:横　縦(画素数)



横×縦の数の
画素データ
(左上から横に)
1:黒
0:白

データ間は
空白で区切られている
(本当は無くても良い)

1行は70文字以下

今回はあまり気にしない

データ入力部分（共通部分）

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_X 800
#define MAX_Y 800
#define BLACK 1
#define WHITE 0
```

取り扱える画像の最大の大きさ（縦、横）

```
main(){
  int img_data[MAX_Y][MAX_X];
  int i, j, x_size, y_size;

  if (getchar() != 'P' || getchar() != '1'){
    fprintf(stderr, "データの形式が違います\n");
    exit(1);
  }
  scanf("%d", &x_size);
  scanf("%d", &y_size);
  if (x_size > MAX_X || y_size > MAX_Y){
    fprintf(stderr, "データが大きすぎます\n");
    exit(2);
  }
  for (i = 0; i < y_size; i++){
    for (j = 0; j < x_size; j++){
      if(scanf("%d",&img_data[i][j]) != 1){
        fprintf(stderr, "データ入力に異常があります\n");
        exit(3);
      }
      if(img_data[i][j] != WHITE && img_data[i][j] != BLACK){
        fprintf(stderr, "データが異常でした\n");
        exit(4);
      }
    }
  }
```

画素データ読み込み

画像用二次元配列

最初に「P1」と書いていないものは、データ形式が違う
この部分はプログラミング入門では扱わない文字型を使用しているので、今のところは呪文だと思っておいてください
（詳しくはP67,p139参照）

x,yそれぞれの画素数を得る

画素数が多すぎる場合

エラーメッセージは全て標準エラー出力へ

scanf入力データがおかしいか個数より早くEOFになった場合

データが白黒ではない場合

各エラーコードに対応したサンプルデータが
/home/course/prog0/public_html/2006/lec/source/lec12-err{1,2,3a,3b,4}.pbm
にあるので試してみるとよい

左90度回転出力

```
printf("P1\n");
printf("%d %d\n", y_size, x_size);
for (i = 0; i < x_size; i++){
  for (j = 0; j < y_size; j++){
    printf("%d ",img_data[j][x_size-1-i]);
  }
  printf("\n");
}
```

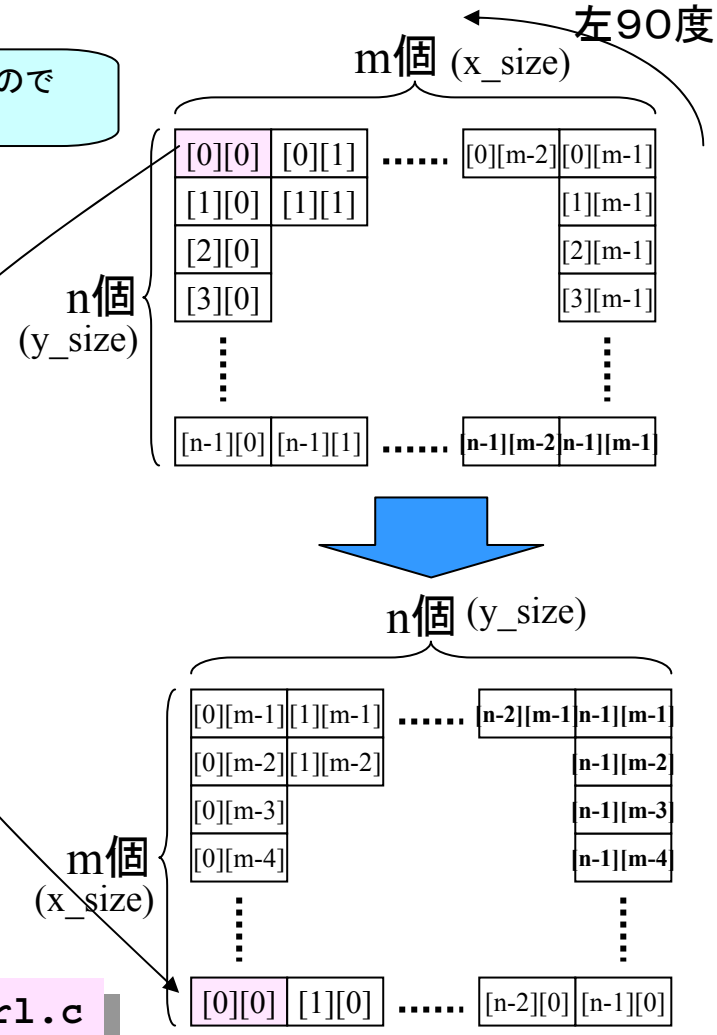
最初にP1と画素数を出力

縦横反転するのでy,xの順となる

1行分終了で改行

左90度回転

データ出力



/home/course/prog0/public_html/2006/lec/source/lec12-rl.c

実行結果

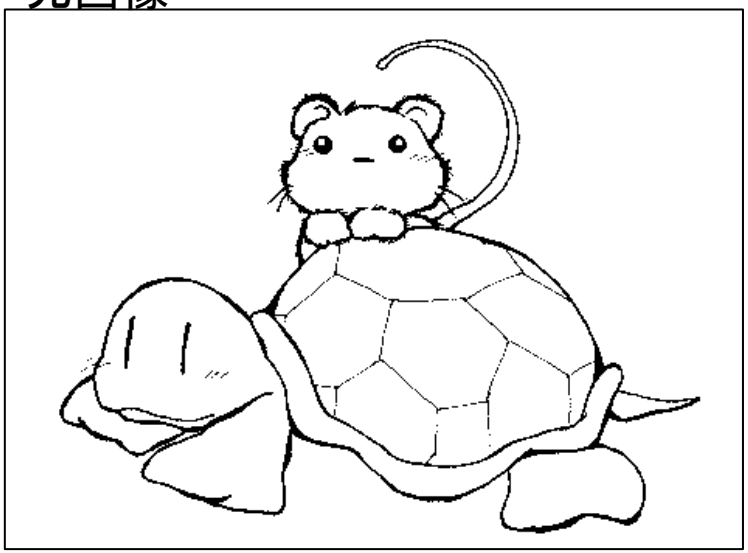
元データの表示

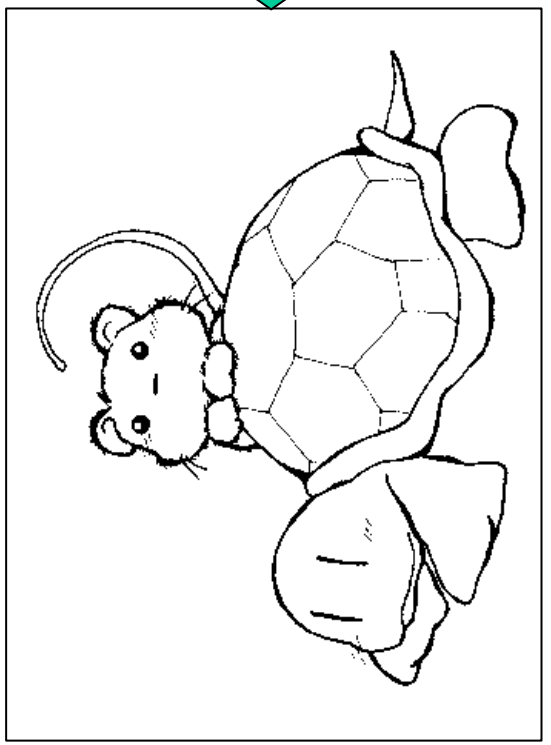
結果をdisplayコマンドにパイプ

```
std1dc1{s1000000}1: display lec12-1.pbm &
std1dc1{s1000000}2: gcc lec12-rl.c -o rotl
std1dc1{s1000000}3: ./rotl < lec12-1.pbm | display &
std1dc1{s1000000}4:
```

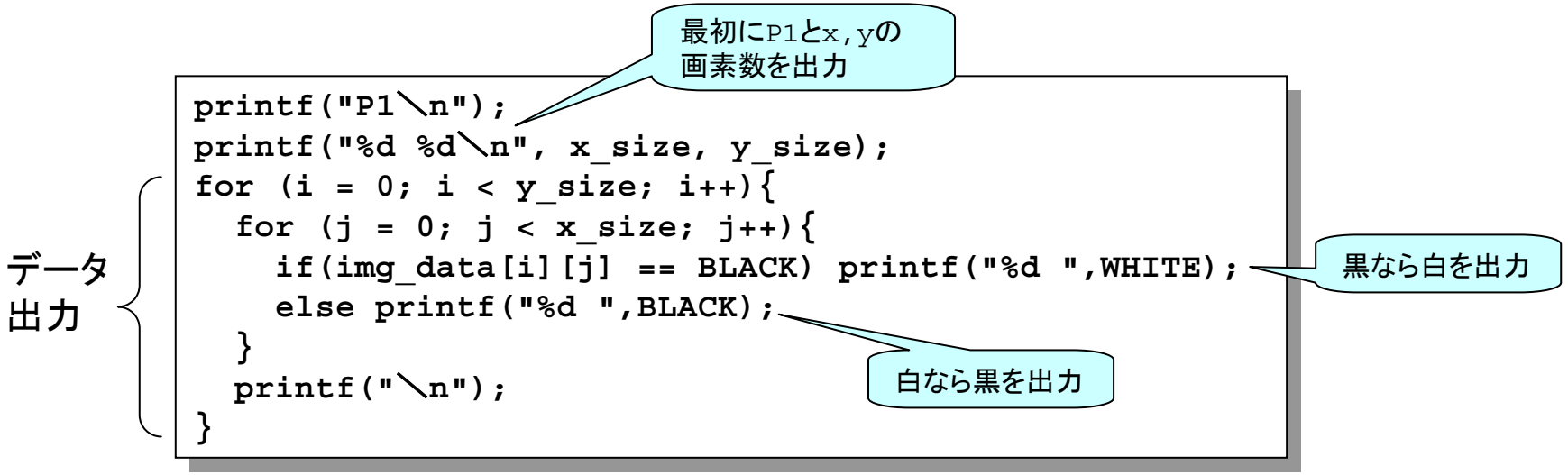
元画像ファイルをリダイレクトで入力

元画像





◆白黒反転プログラム




```
/home/course/prog0/public_html/2006/lec/source/lec12-iv.c
```

その他以下のプログラムソースがある（演習問題の関係で、* がついているものは公開していない）

右90度回転*	/home/course/prog0/public_html/2006/lec/source/lec12-rr.c
左右反転	/home/course/prog0/public_html/2006/lec/source/lec12-lr.c
上下反転*	/home/course/prog0/public_html/2006/lec/source/lec12-ud.c
辺縁検出	/home/course/prog0/public_html/2006/lec/source/lec12-eg.c

実行結果

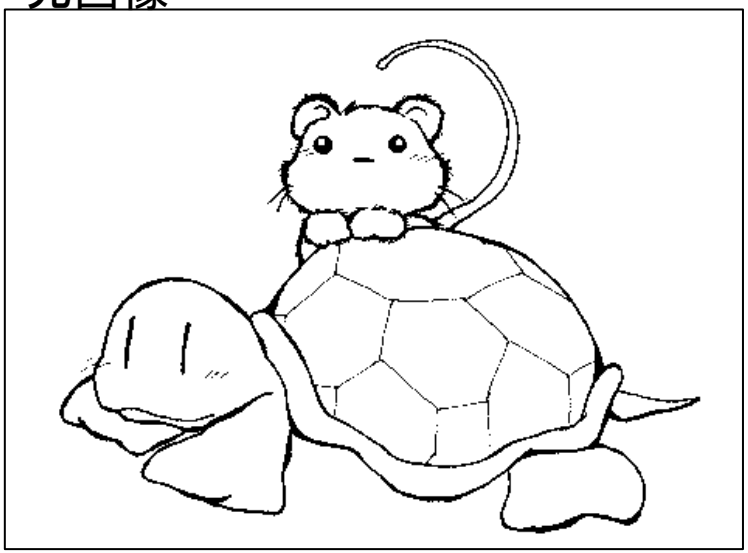
```
std1dc1{s1000000}1: gcc lec12-iv.c -o invrt
std1dc1{s1000000}2: ./rotl < lec12-1.pbm | ./invrt | display &
std1dc1{s1000000}3:
```

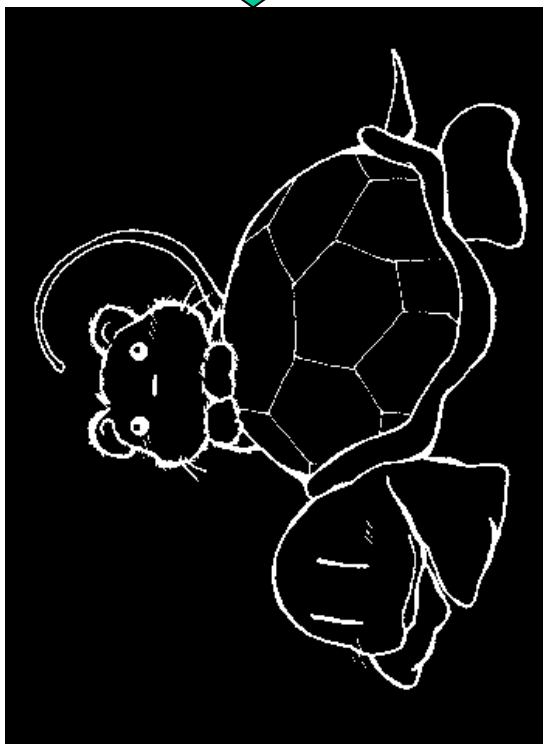
左90度回転の後
白黒反転

ファイルからリダイレクト

このようにパイプで処理をどんどん繋いで行く事が出来る

元画像



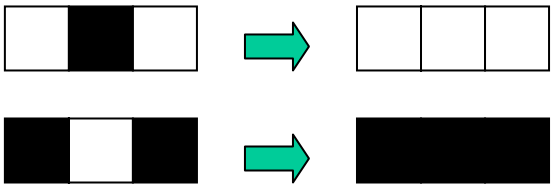


ノイズ除去

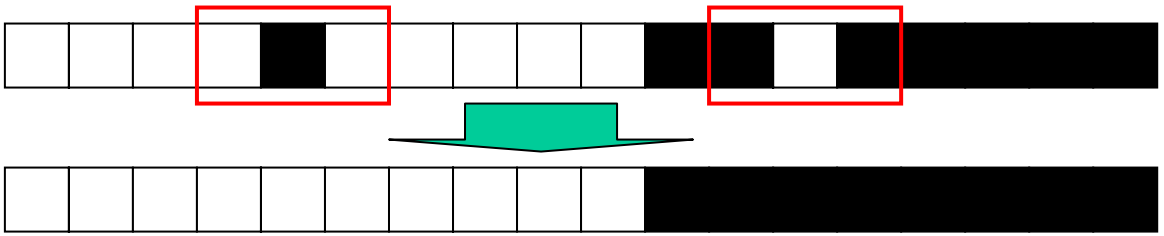
意味のある画像だと、ぽつんと点があることはあまりない
そこで、ある点を見た時

自分が黒点で、両隣の点が白い場合
自分が白点で、両隣の点が黒い場合

は「ノイズ＝雑音（この場合は画像中のゴミ）」と認識して
両隣の色で置き換えることにする。
つまり:



例えば以下のような感じで画像が改善される



ノイズ除去プログラム

```
printf("P1\n");
printf("%d %d\n", x_size, y_size);
for (i = 0; i < y_size; i++){
  printf("%d ",img_data[i][0]);
  for (j = 1; j < x_size-1; j++){
    if(img_data[i][j-1] == WHITE &&
       img_data[i][j]   == BLACK &&
       img_data[i][j+1] == WHITE){
      printf("%d ",WHITE);
    }
    else if(img_data[i][j-1] == BLACK &&
            img_data[i][j]   == WHITE &&
            img_data[i][j+1] == BLACK){
      printf("%d ",BLACK);
    }
    else printf("%d ",img_data[i][j]);
  }
  printf("%d ",img_data[i][x_size-1]);
  printf("\n");
}
```

行の先頭は隣がないのでそのまま出力

横方向の両端を除いた点に関して処理を行う

白黒白なら白を出力

黒白黒なら黒を出力

2つのケース以外の場合はそのまま出力

行の最後は隣がないのでそのまま出力

/home/course/prog0/public_html/2006/lec/source/lec12-nc.c

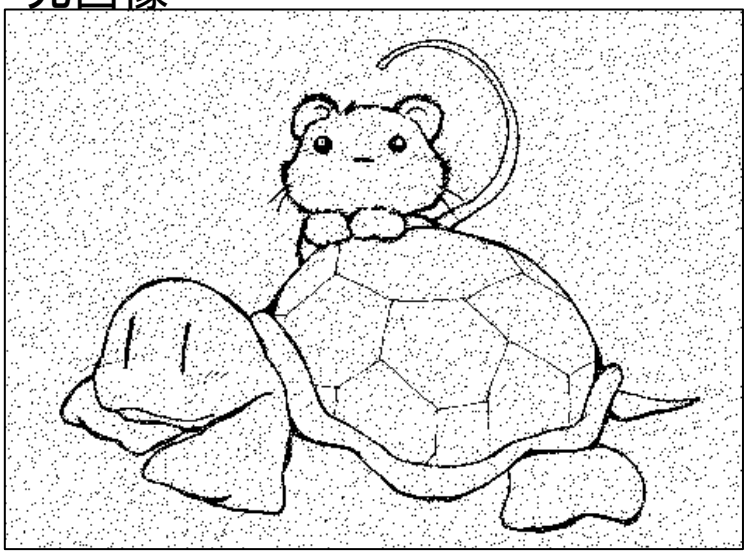
実行結果

```
std1dc1{s1000000}1: gcc lec12-nc.c -o hncut
std1dc1{s1000000}2: ./hncut < lec12-2.pbm | display &
std1dc1{s1000000}3:
```

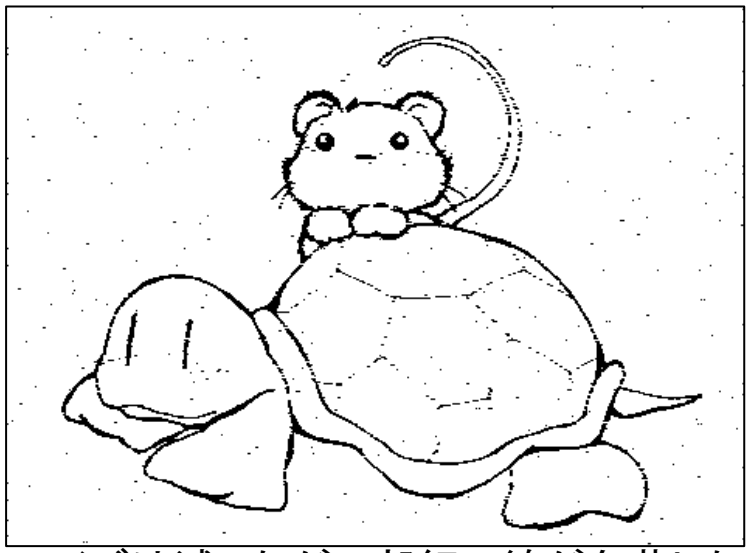
結果をdisplayコマンドにパイプ

ファイルからリダイレクト

元画像



細かい点状のノイズを加えた



ノイズは減ったが一部細い線が欠落した

パイプとリダイレクションのまとめ

- パイプのまとめ
  - 標準入力の切り替え　→　コマンドの前に「｜」　例：**ls | wc -l**
  - 標準出力の切り替え　→　コマンドの後に「｜」　例：**cat hoge.txt | head**

- リダイレクションまとめ
  - 標準入力の切り替え　→　<　　例：**./a.out < in.txt**
  - 標準出力の切り替え(ファイル書き換え)　→　>　　例：**ls –l > out.txt**
  - 標準出力の切り替え(ファイルに追加)　→　>>　　例：**ls –l >> out.txt**