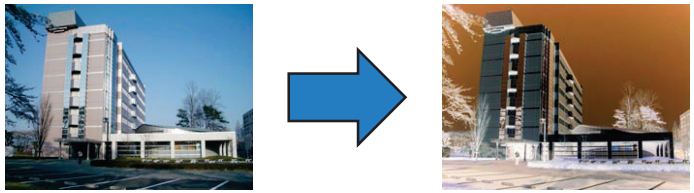
プログラミング演習II
課題4　画像処理
第1週目

担当：篠田一馬

課題4の概要

- 画像処理プログラムを作成することで，ファイル入出力，コマンドライン引数，ポインタ配列，ビット演算を学ぶ



課題4の最終レポート

- ユーザが指定した方法で入力画像を処理し，結果を別ファイルとして出力するプログラムを作成
  - 減色処理と上下反転処理は必須
  - 他の画像処理はオプション

余力があれば

- 画像へのノイズ付加やノイズ除去，特定の物体の自動検出なども可能

課題4の注意点

- 最終レポートのソースコードと実行結果は全員異なる内容になるので，レポートのコピペは不可能
- 第1週目の最初の作業課題からレポート用のプログラムの作成を始めるため，最初からしっかり取り組む必要がある
- 前週までの課題を終わらせないと，次週の授業についていけなくなるため注意
- 途中，解答は公開しない

わからないことがあれば

- 疑問，質問があれば遠慮なくTAや篠田まで
  - メールの場合
    - shinoda@a.utsunomiya-u.ac.jp
  - C-learningの「連絡・相談」やwebclassのメッセージ機能でもOK
  - いずれもソースコード添付を推奨
- 質問する学生＝熱心な学生
  - 質問すれば，わからないことが解決する上，教員に良い印象を与える（決して減点にはならない）

【参考】画像ビューア IrfanView

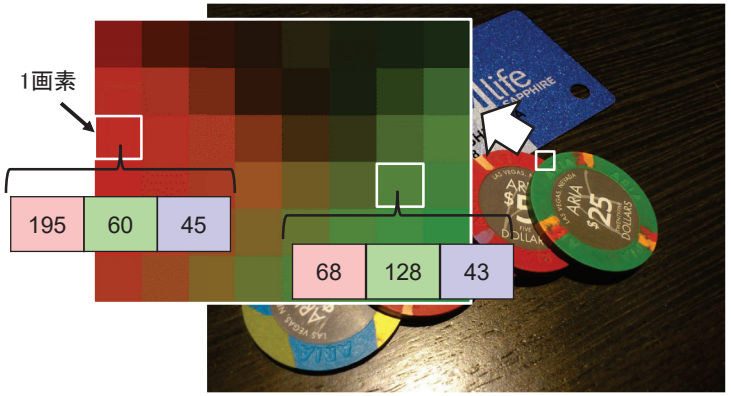
- 自宅のPCで課題を進める場合は，IrfanViewをダウンロード，インストールすることを推奨
  - 本課題で扱う画像ファイルを閲覧するために必要
- PPM画像ファイルを閲覧できれば，他のソフトでも問題ない

画像データ構造の基本

ディジタル画像データの基本

ディジタル画像データの基本

- 画素ごとに赤，緑，青の3色の情報を持っている
- 赤，緑，青（RGB）の組み合わせで色を表現



画素値の例

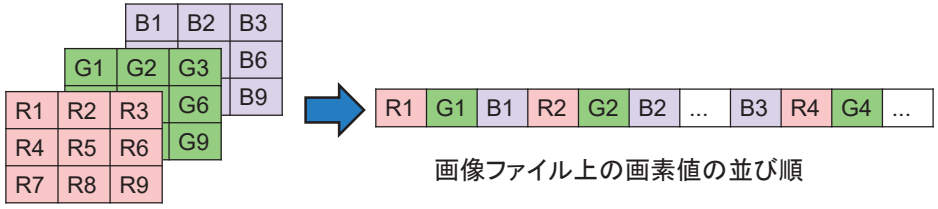
- 一般的には0から255（8 bit）で表現
- 全部で16,777,216色を表現可能

R	G	B	色
0	0	0	
255	0	0	
0	255	0	
0	0	255	
255	255	0	
0	255	255	
255	0	255	

R	G	B	色
10	10	10	
128	128	128	
200	200	200	
255	255	255	
255	192	0	
202	254	207	
235	189	173	

画像ファイルの基本

- 基本的にR，G，Bの順に数値が一列に並ぶ
  - メモリは1次元なので，1次元に展開される



- 画素値やヘッダの表現方法の違いで様々なフォーマット（拡張子）がある
  - bmp, jpg, png, tif, raw, ppm, …

最も単純な画像ファイル（raw画像）

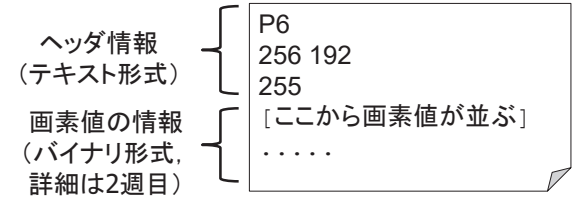
- 画素値の情報のみを持つファイル
  - 未圧縮，劣化なし
- 一次元に展開されているので，元々の縦画素数と横画素数がわからない
  - 別途，縦画素数と横画素数の情報が必要

253	22	89	254	22	89	...	33	189	55	...

ファイル上のデータ

PPM画像

- 画像に関する情報をヘッダとして持つ画像形式
  - 縦画素数と横画素数の情報が含まれる
  - 余計な情報が少ないため，処理しやすい
- 本課題ではPPM画像を用いる



ファイル上のデータ

PPM画像のヘッダの詳細

P6
256 192
255
［ここから画素値が並ぶ］
・・・・・

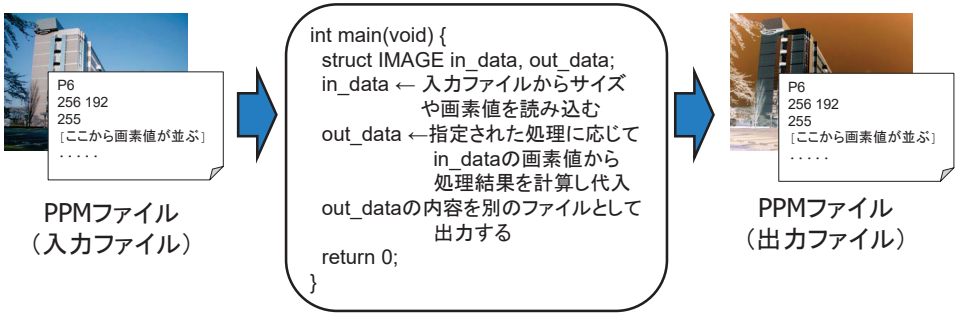
ファイル上のデータ

- P6
  - PPMを意味する文字列
  - 本課題では常にP6で固定
- 256[半角空白]192
  - 横画素数，縦画素数
  - 半角空白を忘れずに
- 255
  - 画素値の最大値
  - 本課題では常に255で固定

プログラム設計の指針

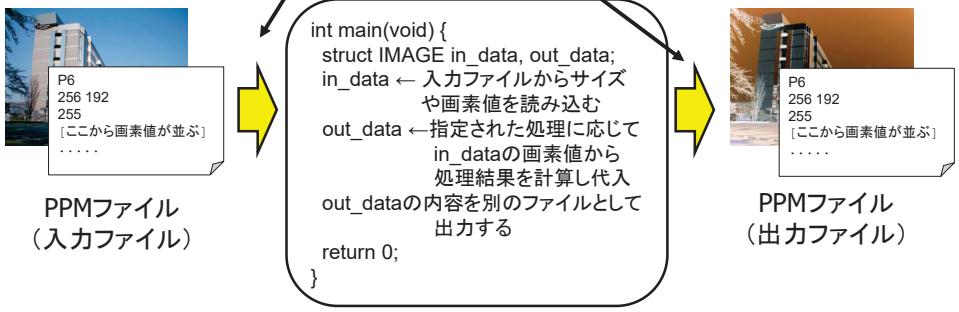
画像処理プログラムの全体像

1. 入力ファイルからヘッダと画素値を読み込む
2. 画素値を処理に応じて適宜変更する
3. 変更した画像情報を別ファイルとして出力する



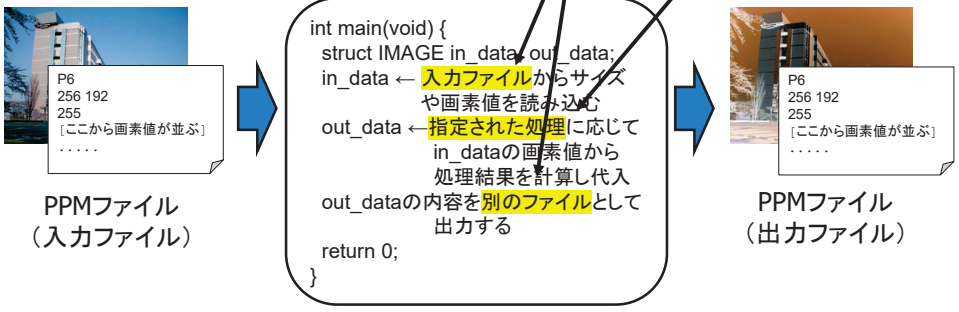
必要な知識①

- ファイル操作
  - ファイルの読み込み，書き込み



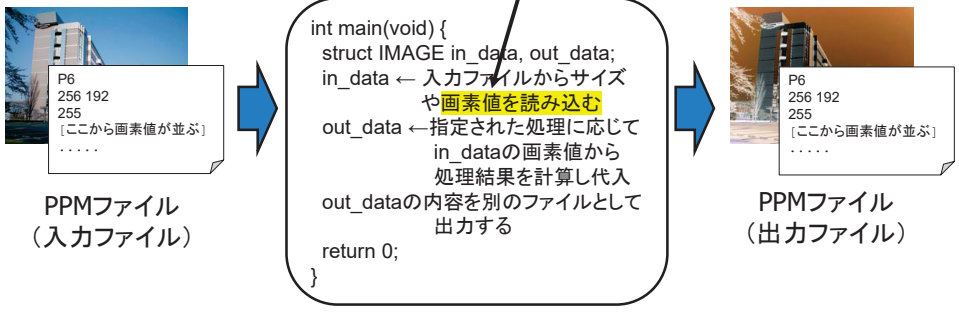
必要な知識②

- コマンドライン引数
  - 画像処理をしたいユーザがファイル名や処理を指定



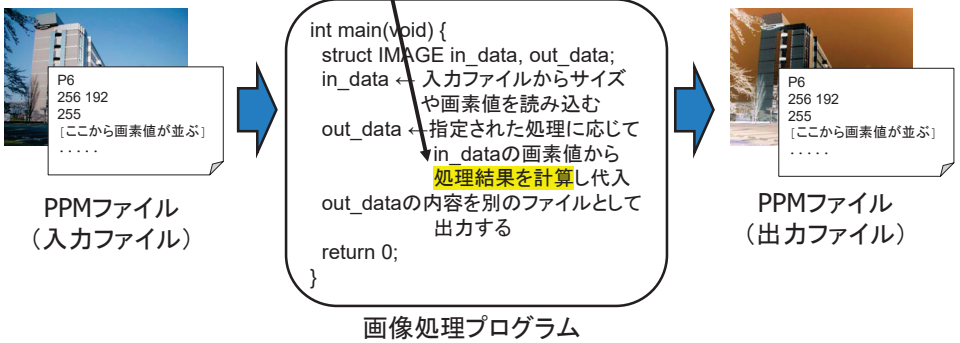
必要な知識③

- ポインタ配列と動的確保
  - 画素値を格納する配列の長さはファイル依存



必要な知識④

- ビット演算
  - 減色処理はビット演算を使うと楽



全体の流れ

- 1週目（ファイル操作，コマンドライン引数）
  - 画像ファイルのヘッダ情報をテキストファイルに出力
- 2週目（ファイル操作）
  - 画像ファイルの画素情報をバイナリファイルに出力
- 3週目（ポインタ配列と動的確保）
  - 様々な画素数の画像ファイルの入出力に対応
- 4週目（ビット演算）
  - 画像を減色または上下反転させて出力

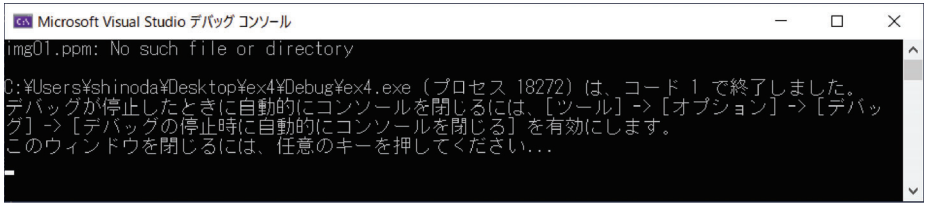
プログラムテンプレートの
ダウンロードと実行

【作業課題】ひな形のダウンロードと実行

- 課題4，1週目の講義資料webページからプログラムテンプレートと画像ファイルをダウンロード
  - プロジェクトフォルダ”ex4”を作成し，main.cとimg01.ppmをダウンロードして同じフォルダにおく
  - img01.ppmをIrfanViewで表示できるか確認しておく
- 今週の目標は，入力画像img01.ppmのヘッダ情報をresult.txtに出力させること
  - 実行してもまだresult.txtは作成されないことを確認

実行結果の表示について

- 「xxxx: No such file or directory」
  - 入力ファイルが存在しない（または入力ファイル名が間違っている）ため，ファイルの有無を要確認
- 「コード1で終了しました」（コードが0以外で終了）
  - 異常終了なので見直しが必要
  - 正常終了であればmain.cはコード0で終了する



作業課題の目安: 10分

プログラムテンプレートの流れ

プログラムの流れ

```
typedef struct {
  unsigned char r;
  unsigned char g;
  unsigned char b;
} PIXEL;

typedef struct {
  int    xsize;
  int    ysize;
  int    level;
  PIXEL pBuffer[49152];
} IMAGE;
```

画素値を格納するための構造体
（詳細は2週目）

ヘッダ情報と画素値をまとめて
扱うための構造体で，
1週目はxsize, ysize, levelのみ使用

プログラムの流れ

```
typedef struct {
  unsigned char r;
  unsigned char g;
  unsigned char b;
} PIXEL;

typedef struct {
  int    xsize;
  int    ysize;
  int    level;
  PIXEL pBuffer[49152];
} IMAGE;
```

typedef structで定義しているので，
プログラム中で変数を宣言するときは
structを記載する必要なし

```
int main(…) {
    struct IMAGE a, b;
    …
}
```
NG

```
int main(…) {
    IMAGE a, b;
    …
}
```
OK

プログラムの流れ

```
int main(int argc, char* argv[])
{
  IMAGE in_image_data, out_image_data;

  /* 画像情報のロード */
  if (iioLoadFile(&in_image_data, "img01.ppm")) {
    return 1;
  }
  /* 画像情報のコピー */
  ipCopy(&in_image_data, &out_image_data);
  /* 画像情報の書き出し */
  iioSaveFile(&out_image_data, "header.txt");
  /* メモリリークチェック */
  _CrtDumpMemoryLeaks();
  return 0;
}
```

- 構造体の変数の宣言
- 画像ファイルからヘッダ情報を in_image_dataに 読み込む関数
- in_image_dataの画像情報を out_image_dataにコピーする関数
- out_image_dataをテキスト ファイルに出力する関数

プログラムの流れ

```
void ipCopy(IMAGE* p_in_image, IMAGE* p_out_image)
{
  /* ヘッダ情報のコピー */
  p_out_image->xsize = p_in_image->xsize;
  p_out_image->ysize = p_in_image->ysize;
  p_out_image->level = p_in_image->level;

}
```

入力ファイルから読み込んだ画像情報のうち，ヘッダ情報（横画素数，縦画素数，階調数）のみをp_out_imageにコピーしている．
（画素値のコピーは2週目に行うので，1週目は省略）

main関数内のIMAGE構造体を編集する必要があるため，
ipCopy関数へはポインタ渡しを行っている点に注意

（補足）ポインタ渡し（アドレス渡し）

- 呼出し元の変数を呼出し先の関数内で変更可
  - 値渡しだと変更できない

値渡しの例

```
int main(void)
{
  int a = 0;

  func(a);
  printf("%d¥n", a);

  return 0;
}
```

```
void func(int b)
{
  b = 2;
}
```

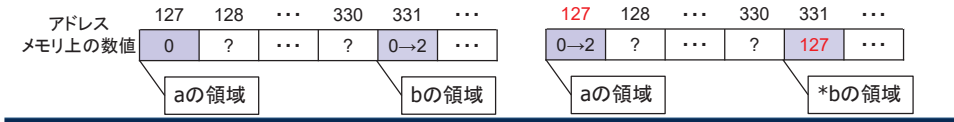
→ 0が出力される

ポインタ渡し（アドレス渡し）の例

```
int main(void)
{
  int a = 0;

  func(&a);
  printf("%d¥n", a);

  return 0;
}
```

```
void func(int *b)
{
  *b = 2;
}
```

→ 2が出力される



ファイル操作の基本

ファイル操作の基本

1. ファイルを開いてファイルポインタをセット
   fopen関数
2. ファイル操作用の関数を使用してデータを読み込む/書き込む
   fgets, fscanf, fread, fputs, fprintf, fwrite…
3. ファイルを閉じる
   fclose関数

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

Hello

input.txt

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

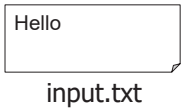
- ファイルポインタの定義
- ファイル名を表す文字列
- ファイル中のテキストを格納するための char配列

Hello

input.txt

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

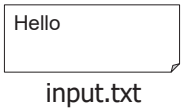


input.txt

input.txtファイルを”read”モードで開き，ファイルポインタfpに関連付け

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```



input.txt

【fopen関数】
第一引数：ファイル名
第二引数：モード
"r": テキストファイルの読み込み
"w": テキストファイルへの書き込み
"a": テキストファイルへの追加書き込み
"rb", "wb", "ab": バイナリファイル用の
　読み込み/書き込み/追加
（画像ファイルはバイナリファイル）

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

input.txt

fpに関連付けられたファイル（input.txt）から一行だけlineBufferに読み込む．
ただし読み込み上限は100バイトとする．
→lineBufferに'H' 'e' 'l' 'l' 'o' '¥0'が格納される

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

input.txt

【fgets関数】
第一引数：配列の先頭アドレス
第二引数：読み込み上限のバイト数
第三引数：ファイルポインタ

ファイル読み込みの例

```
int main(void)
{
  FILE *fp;
  char *fname = “input.txt”;
  char lineBuffer[100];

  fp = fopen(fname, "r");

  fgets(lineBuffer, 100, fp);

  fclose(fp);

  return 0;
}
```

ファイルクローズ

input.txt

ファイル書き込みの例

```
int main(void)
{
  FILE *fpo;
  char *fname = “output.txt”;
  char lineBuffer[100] = “Hello”;

  fpo = fopen(fname, "w");

  fprintf(fpo, "%s¥n", lineBuffer);

  fclose(fpo);

  return 0;
}
```

output.txt

ファイル書き込みの例

```
int main(void)
{
  FILE *fpo;
  char *fname = “output.txt”;
  char lineBuffer[100] = “Hello”;

  fpo = fopen(fname, "w");

  fprintf(fpo, "%s¥n", lineBuffer);

  fclose(fpo);

  return 0;
}
```

output.txtをwriteモードで開く

output.txt

ファイル書き込みの例

```
int main(void)
{
  FILE *fpo;
  char *fname = “output.txt”;
  char lineBuffer[100] = “Hello”;

  fpo = fopen(fname, "w");

  fprintf(fpo, "%s¥n", lineBuffer);

  fclose(fpo);

  return 0;
}
```

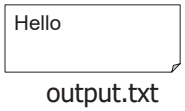
【fprintf関数】
第一引数：出力先のファイルポインタ．
他の引数はprintf関数と同じ書式．
つまり，output.txtに”Hello”が
書き込まれる

output.txt

ファイル書き込みの例

```
int main(void)
{
  FILE *fpo;
  char *fname = “output.txt”;
  char lineBuffer[100] = “Hello”;

  fpo = fopen(fname, "w");

  fprintf(fpo, "%s¥n", lineBuffer);

  fclose(fpo);

  return 0;
}
```

ファイルクローズ



output.txt

ファイル操作のまとめ

- ファイルポインタの宣言，open, closeは定型句
  - ほとんど考える必要がない
  - ファイル名とモード指定のみ注意
- ファイルへの文字列の入出力はコマンドラインの入出力関数（scanfやprintf）とほぼ同じ
  - ファイルポインタの指定を忘れずに

画像ファイルの操作

プログラムの流れ

```
int main(int argc, char* argv[])
{
  IMAGE in_image_data, out_image_data;

  /* 画像情報のロード */
  if (iioLoadFile(&in_image_data, "img01.ppm")) {
    return 1;
  }
  /* 画像情報のコピー */
  ipCopy(&in_image_data, &out_image_data);
  /* 画像情報の書き出し */
  iioSaveFile(&out_image_data, "header.txt");
  /* メモリリークチェック */
  _CrtDumpMemoryLeaks();
  return 0;
}
```

構造体の変数の宣言

画像ファイルからヘッダ情報を
in_image_dataに
読み込む関数

in_image_dataの画像情報を
out_image_dataにコピーする関数

out_image_dataをテキスト
ファイルに出力する関数

プログラムの流れ

- iioLoadFileの目的
  - ファイル名がfnameのPPMファイルを開き，画像情報をpImageに格納する
  - ひな形ではヘッダ情報の読み込み部分を作成済み
  - ヘッダ情報の読み込みは少々ややこしいので，以下はあくまで参考情報

```
int iioLoadFile(IMAGE* pImage, char* fname)
{
  FILE* fpr;
  char lineBuffer[LINEMAX];
  int i;
```


【参考】プログラムの流れ

```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```

ファイル名fnameをreadモードで開く．ただし画像ファイルなので，バイナリモード"rb"にする

【参考】プログラムの流れ

```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```

もしファイルが存在しない場合（ファイル名を間違えたなど）は，fopenはNULLを返す
→fprにNULLが代入される

【参考】プログラムの流れ

```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```

もしファイルが存在しない場合（ファイル名を間違えたなど）は，fopenはNULLを返す
→fprにNULLが代入される

NULLかどうかを判定し，Yesであれば処理を中断（return）する．

NULLでなければ，ifの中は実行されない．

【参考】プログラムの流れ

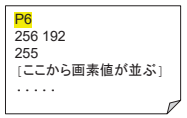
```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```

perrorはエラー出力用の関数.
printfでも十分だが,
fopen失敗の直後に呼び出すと,
自動的に以下を出力するので便利

> fname: No such file or directory

【参考】プログラムの流れ

```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```

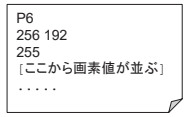


img01.ppm

PPMファイルから先頭1行を
読み込む.
つまり, lineBuffer ← "P6"

【参考】プログラムの流れ

```
/* ファイルオープン */
if ((fpr = fopen(fname, "rb")) == NULL) {
  perror(fname);
  return 1;
}

/* ヘッダ部読み込み開始 */
fgets(lineBuffer, LINEMAX, fpr);

if (strcmp(lineBuffer, "P6¥n")) {
  // ERROR
  fclose(fpr);
  return 1;
}
```



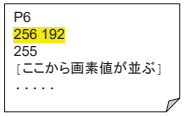
img01.ppm

lineBufferに読み込まれた文字列が
P6でなければPPMファイルではない
ので, if文の中を実行してreturn.

【参考】プログラムの流れ

```
fgets(lineBuffer, LINEMAX, fpr);

while (lineBuffer[0] == '#')
  fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d %d¥n", &pImage->xsize, &pImage->ysize);

fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d", &pImage->level);
```

次の1行をlineBufferに読み込む
lineBuffer ← "256 192"

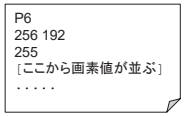


img01.ppm

【参考】プログラムの流れ

```
fgets(lineBuffer, LINEMAX, fpr);

while (lineBuffer[0] == '#')
  fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d %d¥n", &plmage->xsize, &plmage->ysize);

fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d", &plmage->level);
```

#から始まる文字列だった場合は
コメント文なので読み飛ばす
（今回は該当せず）

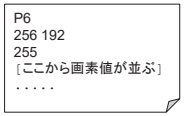


img01.ppm

【参考】プログラムの流れ

```
fgets(lineBuffer, LINEMAX, fpr);

while (lineBuffer[0] == '#')
  fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d %d¥n", &plmage->xsize, &plmage->ysize);

fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d", &plmage->level);
```

lineBufferの文字列"256 192"から
sscanfでxsizeとysizeを取得

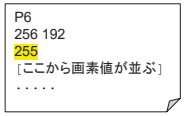


img01.ppm

【参考】プログラムの流れ

```
fgets(lineBuffer, LINEMAX, fpr);

while (lineBuffer[0] == '#')
  fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d %d¥n", &plmage->xsize, &plmage->ysize);

fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d", &plmage->level);
```

次の1行をlineBufferに読み込む
lineBuffer ← "255"

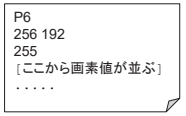


img01.ppm

【参考】プログラムの流れ

```
fgets(lineBuffer, LINEMAX, fpr);

while (lineBuffer[0] == '#')
  fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d %d¥n", &plmage->xsize, &plmage->ysize);

fgets(lineBuffer, LINEMAX, fpr);

sscanf(lineBuffer, "%d", &plmage->level);
```

lineBufferの文字列"255"から
sscanfでlevelを取得



img01.ppm

【参考】プログラムの流れ

```
 /* バッファ用メモリ確保と画像
データの格納（2週目） */


 /* ファイルクローズ */
 fclose(fpr);

 return 0;
}
```

本来はここで画素値を読み込むが，
1週目は何もしない．
ヘッダ情報のみpImageに格納して終了．

ファイルクローズ

プログラムの流れ

```
int main(int argc, char* argv[])
{
 IMAGE in_image_data, out_image_data;

 /* 画像情報のロード */
 if (iioLoadFile(&in_image_data, "img01.ppm")) {
  return 1;
 }
 /* 画像情報のコピー */
 ipCopy(&in_image_data, &out_image_data);
 /* 画像情報の書き出し */
 iioSaveFile(&out_image_data, "header.txt");
 /* メモリリークチェック */
 _CrtDumpMemoryLeaks();
 return 0;
}
```

構造体の変数の宣言

画像ファイルからヘッダ情報を
in_image_dataに
読み込む関数

in_image_dataの画像情報を
out_image_dataにコピーする関数

【演習課題】
out_image_dataをテキスト
ファイルに出力する関数を
作成する

プログラムの流れ

- iioSaveFileの目的
  - （本来の目的）pImageの画像情報全てをファイル名fnameのPPMファイルに出力する
  - （今週の目標）pImageのヘッダ情報のみをファイル名fnameのTXTファイルに出力する

```
int iioSaveFile(IMAGE* pImage, char* fname)
{
 int i;
 FILE* fpo;
```


【演習課題4_1_1】

- iioSaveFileのヘッダ出力部分を完成させる
  - “P6”と，pImageのxsize, ysize, levelを，PPMのヘッダ形式と同じようにテキストファイルfnameに出力
  - header.txtが以下と同じ出力になるか確認
  - 最後は改行必須

```
P6
256 192
255
```

header.txt

```
int iioSaveFile(IMAGE* pImage, char* fname)
{
 int i;
 FILE* fpo;

 /* ファイルオープン */

 /* PPMのヘッダ情報と画素値をファイルに出力 */

 /* ファイルクローズ */

 return 0;
}
```

演習課題に取り組む前に

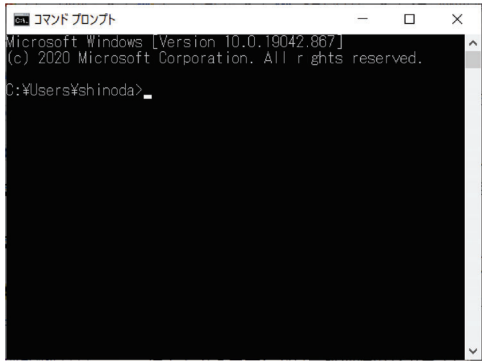
- 演習課題ごとにファイルやプロジェクトのログを残しておくこと
  - できればプロジェクトのフォルダを丸ごとコピーして，演習課題番号をふっておく
  - そうしないと，間違って編集した際に1日目の最初からやり直すことになる

演習課題の目安: 20分

コマンドライン引数

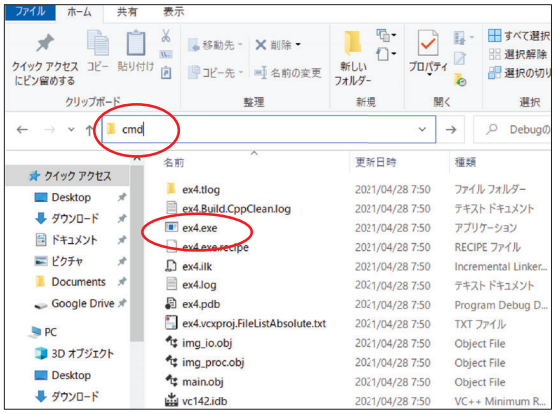
コマンドプロンプト（コマンドライン）

- 文字入力によってプログラムの実行や結果表示を行うもの



コマンドプロンプトの簡単な起動方法（Win）

- 実行したいファイル(exe)があるフォルダを開き，アドレスバーに”cmd”と入力してEnter



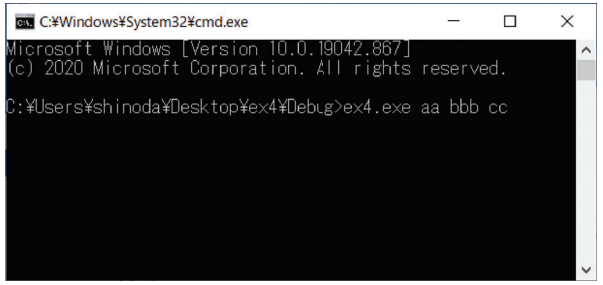
【作業課題】

- x64¥Debugフォルダのexeをコマンドライン実行
  - ppmファイルはDebugフォルダ（exeファイルと同一のフォルダ）に移動しておくこと
  - header.txtを削除しておく
  - Debugフォルダ内を表示した状態で，コマンドプロンプト起動
  - コマンドラインで”ex4.exe”と入力して実行
  - プログラムが実行され，header.txtが作成されることを確認

作業課題の目安: 5分

コマンドライン引数

- ex4.exeの後に，半角空白区切りで文字列を入力
- 空白区切りごとの各文字列（“ex4.exe”, ”aa”, “bbb”, “cc”）をプログラム内で使用できる



main関数におけるコマンドライン引数

- main関数の引数を下記の通り記載する
  - C言語の規格で引数の型と個数が定まっているため下記以外の書き方はできない

```
int main(int argc, char* argv[])
{

  return 0;
}
```

- プログラム実行時，argcとargvには自動的に値やポインタが割り当てられた状態で開始される

main関数におけるコマンドライン引数

- int argc
  - ユーザがコマンドラインで入力した文字列の数
  - “ex4.exe aa bbb cc” の場合，4
- char* argv[]
  - コマンドラインで入力した各文字列へのポインタ
  - “ex4.exe aa bbb cc”の場合

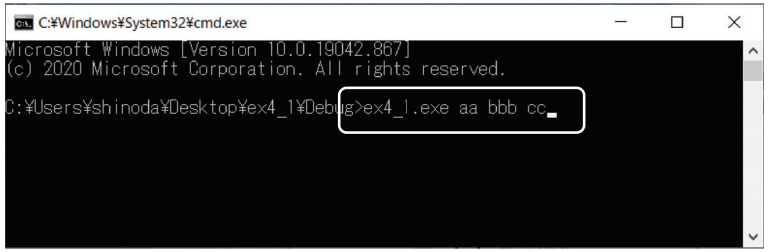
    argv[0]は”ex4.exe”，argv[1]は”aa”，argv[2]は”bbb”…

```
int main(int argc, char* argv[])
{

  return 0;
}
```


コマンドライン引数の挙動例

```
int main(int argc, char* argv[])
{
  int i;
  for (i = 0; i < argc; ++i)
    printf("%s¥n", argv[i]);

  return 0;
}
```

このプログラム（ex4_1.c）を作成・ビルドし，
コマンドプロンプトから
> ex4_1.exe aa bbb cc
と実行したとする



コマンドライン引数の挙動例

```
int main(int argc, char* argv[])
{
  int i;
  for (i = 0; i < argc; ++i)
    printf("%s¥n", argv[i]);

  return 0;
}
```

ユーザが入力したコマンドライン引数に従い，
main開始時に自動的にargcとargvに
データが入力されている．
argc ← 4
argv[0] ← “ex4_1.exe”の先頭アドレス
argv[1] ← “aa”の先頭アドレス
argv[2] ← “bbb”の先頭アドレス
argv[3] ← “cc”の先頭アドレス

コマンドライン引数の挙動例

```
int main(int argc, char* argv[])
{
  int i;
  for (i = 0; i < argc; ++i)
    printf("%s¥n", argv[i]);

  return 0;
}
```

argv[0]: "ex4_1.exe"の先頭アドレスなので，
i = 0では"ex4_1.exe"が出力される．
argv[1]: "aa"の先頭アドレスなので，
i = 1では"aa"が出力される．
.....
つまり，実行すると下記が出力される

プログラムの流れ

```
int main(int argc, char* argv[])
{
  IMAGE in_image_data, out_image_data;

  /* 画像情報のロード */
  if (iioLoadFile(&in_image_data, "img01.ppm")) {
    return 1;
  }
  /* 画像情報のコピー */
  ipCopy(&in_image_data, &out_image_data);
  /* 画像情報の書き出し */
  iioSaveFile(&out_image_data, "header.txt");
  /* メモリリークチェック */
  _CrtDumpMemoryLeaks();
  return 0;
}
```

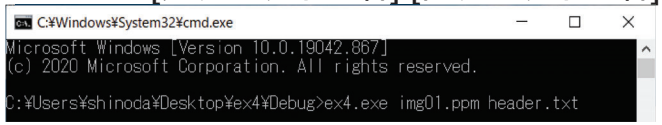
入力ファイル名と出力ファイル名が
固定されているため，
様々な画像を処理したいときに
再ビルドが必要で不便．
scanfで複数回入力させるのも手間．

コマンドライン引数で各ファイル名の
文字列を与えられるように改良する

【演習課題4_1_2】

- 入力ファイル名と出力ファイル名をコマンドライン引数で与えられるようにmain.cを改良する
  - ppmファイルはDebugフォルダ（exeファイルと同一のフォルダ）に移動しておくこと
  - コマンドライン入力は下記を想定する

    ex4.exe [入力ファイル名] [出力ファイル名]



- コマンドラインから実行し，出力ファイル名をユーザが自由に変えられることを確認

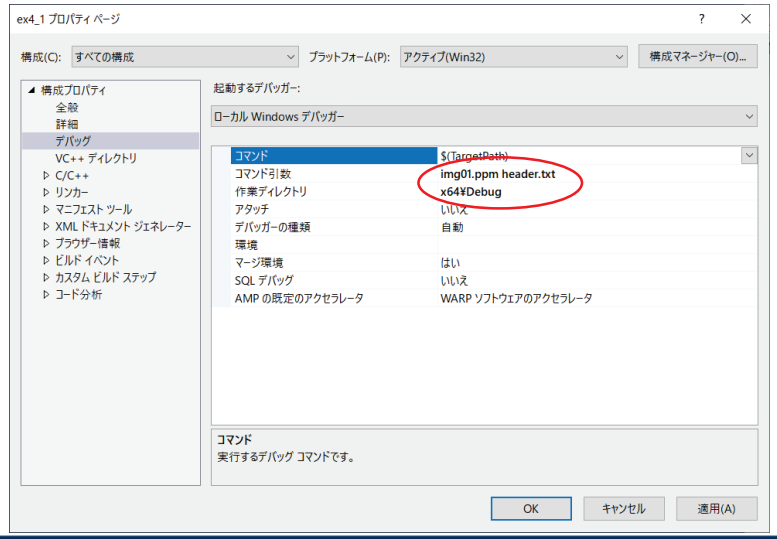
演習課題の目安: 10分

Visual Studioにおける実行時の
コマンドライン引数の設定方法
（以下はデバッグ実行をする場合や
レポート課題では必須）

【作業課題】

- Visual Studioのコマンドライン引数の設定方法
  - Visual Studioのメニューの「プロジェクト」→「プロパティ」→構成を「すべての構成」→「デバッグ」→「コマンド引数」に，exeファイル名以外の引数を入力
    - img01.ppm header.txt
  - また，同ウィンドウの「作業ディレクトリ」を以下にする
    - x64¥Debug
  - PPMファイルはDebugフォルダにおいておく
  - 演習課題4_1_2と同じくファイルが出力されるか確認
  - この方法を使えばコマンドプロンプトの起動は不要

設定画面



Visual Studio以外の環境の場合

- 各自でデバッグ時のコマンドライン引数の設定方法を調査し，設定すること
  - 設定しないとデバッグ実行ができなくなる
  - メモリリークの確認も困難になる

終わった人は

- 余裕があれば次週の課題に取り組むことを推奨
- 最終レポートは，画像処理の実装種類数が多いほど評価UP
  - 例年，10種類以上実装する人も数名