全体の流れ

画像ファイルを開き，画像データをメモリ上にロード

↓

メモリ上にロードした画像データに処理を加える

↓

処理後のデータを出力ファイルに書き出す

↓

画像データ用に確保したメモリを解放

データ構造～1画素をどう表現するか～

画像ファイル内での画素データのフォーマット

88 (136)	B1 (177)	E7 (231)	8A (138)	B2 (178)	E7 (231)	…

1画素あたり RGB各8ビット　合計3バイト

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

PIXEL構造体

r	88
g	B1
b	E7

データ構造～画像データをどう表現するか～

画像ファイルのフォーマット　88 B1 E7

ヘッダー(画像サイズ，最大輝度)	画像データ

画像データのサイズは，ヘッダ読み込み後でないと分からない
→画像用のメモリは[　　]

```
typedef struct {
     int xsize;
     int ysize;
     int level;
     PPIXEL *pBuffer;
} IMAGE;
```

IMAGE構造体

xsize	256
ysize	192
level	255
pBuffer	

ポインタの配列～課題2の文字列のソートの復習～

char *lines[MAXLINES];

文字列にあわせた長さ

定数
MAXLINES個
（20000）

lines[0] strawberry¥0
lines[1] watermelon¥0
lines[2] banana¥0
lines[MAXLINES-1]

ポインタの配列～画像バッファの確保～

```
void iioMallocImageBuffer(IMAGE *pImage)
```

pImage

xsize	256
ysize	192
level	255
pBuffer	

xsize個の[　　]
の配列を確保
malloc(xsize * sizeof(PIXEL));

ysize個の[　　]
の配列を確保
malloc(ysize * sizeof(PPIXEL));

モジュール設計

- どういう機能を用意するか
- モジュールごとの独立性をどう高めるか

メイン
main.c

画像のファイル入出力
画像データ用メモリ確保と解放
img_io.c
img_io.h

画像処理
img_proc.c
img_proc.h

復習:コンパイラの動作(1)

- コンパイラはソースファイルを[　　]

```
//呼び出される関数
double func(double d)
{
   return 0.0;
}

//呼び出し側関数
int caller(void)
{
  f = func(4); // 関数呼び出し
}
```

func という関数は、doubleを受け取ってdoubleを返すんだな‥

double を渡してdoubleを返してもらうようなコードを生成しよう‥4は4.0に変換してから渡そう･･･

関数呼出部分をコンパイルするときに、コンパイラは呼び出される関数の引数の数・型、戻り値の型を知っておく必要がある

10

コンパイラの動作(2)

- 関数記述位置の上下を逆にすると･･･

```
//呼び出し側関数
int caller(void)
{
  f = func(4); // 関数呼び出し
}

//呼び出される関数
double func(double d)
{
   return 0.0;
}
```

func 関数の引数の型も引数の個数も知らないぞ･･･

正常にコンパイルできない

（C処理系ではwarningが、C++処理系ではエラーが発生する）

12

コンパイラの動作(2)

- 関数記述位置の上下を逆にすると･･･

```
//呼び出し側関数
int caller(void)
{
  f = func(4); // 関数呼び出し
}

//呼び出される関数
double func(double d)
{
   return 0.0;
}
```

func 関数の引数の型も引数の個数も知らないぞ･･･

正常にコンパイルできない

（C処理系ではwarningが、C++処理系ではエラーが発生する）

13

関数プロトタイプ宣言

関数呼出部分より前に
[　　]
を行うことにより、正常にコンパイルが可能となる

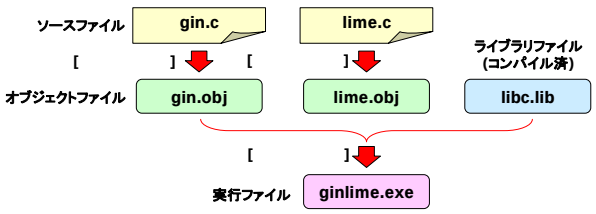
```
double func(double d);

//呼び出し側関数
int caller(void)
{
  f = func(4); // 関数呼び出し
}

//呼び出される関数
double func(double d)
{
   return 0.0;
}
```

func という関数は、doubleを受け取ってdoubleを返すんだな‥

double を渡してdoubleを返してもらうようなコードを生成しよう‥4は4.0に変換してから渡そう･･･

14

分割コンパイル



- ソースファイルを複数に分割し、それぞれをコンパイルしてできたオブジェクトファイルをリンカで結合
- 更新されたソースファイルのみコンパイルすればよいため、プログラムの規模が大きくなった場合でも、微細な変更時のビルド時間を大幅に短縮可能

16

単純にファイルを分割すると…

module.c

```
//呼び出される関数
double func(double d)
{
   return 0.0;
}
```

caller.c

```
//呼び出し側関数
int caller(void)
{
  f = func(4); // 関数呼び出し
}
```

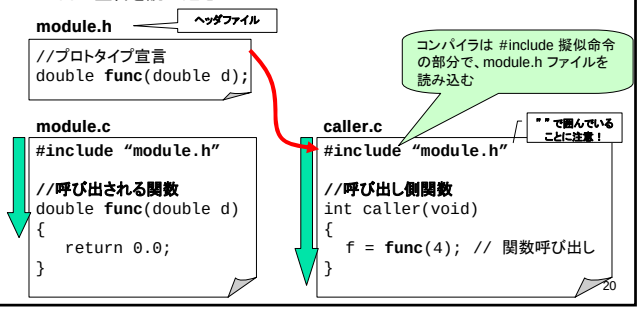
func って何?

コンパイルは
[　　]
ため、module2.c は正常にコンパイルできない。

18

ヘッダファイルの利用

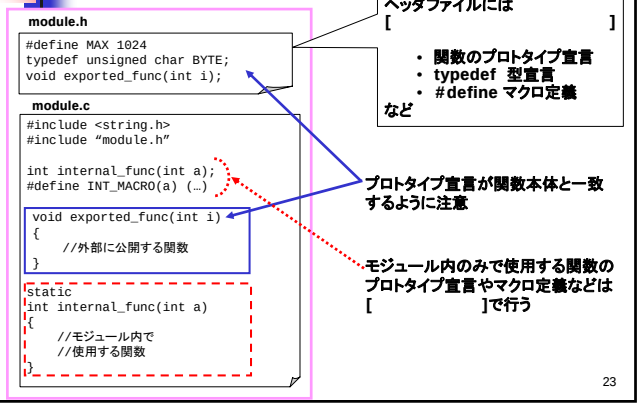
- 呼び出される関数のプロトタイプ宣言は＿＿＿＿＿＿＿＿＿＿に記述
- 呼び出し側のソースファイルで **#include 擬似命令**により、ヘッダファイル内の宣言を読み込む



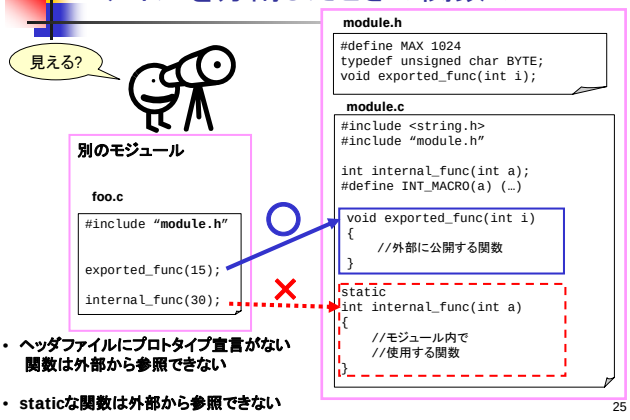
< > と " "

- #include <stdio.h>のように< >で囲まれていると，プリプロセッサはシステムで定められた場所にあるファイルを取り込む
- #include “module.h”のように” ”で囲まれていると，まずプログラムファイルと同じ場所を探し，存在しない場合はシステムで定められた場所を探す

モジュール化の基本

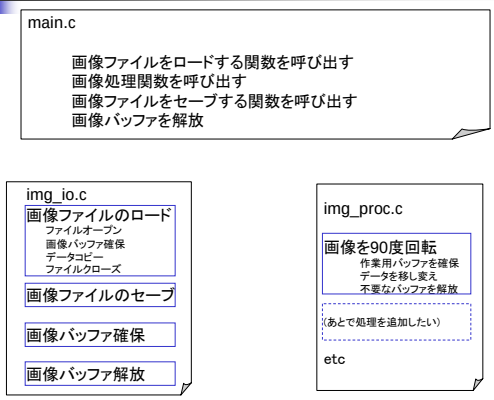


ファイルを分割したときの関数のスコープ



- ヘッダファイルにプロトタイプ宣言がない関数は外部から参照できない
- staticな関数は外部から参照できない

サンプルプログラム2の関数呼び出し関係



関数の設計

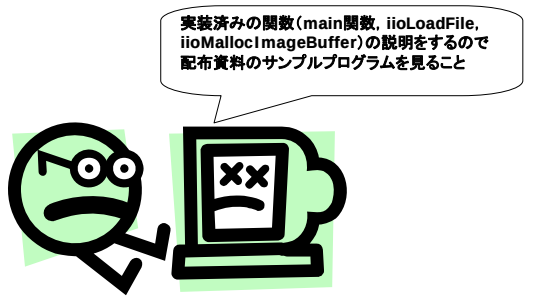
- ファイル入出力＆メモリ確保/解放　関連

```
int iioLoadFile(IMAGE *pImage, const char *fname);
int iioSaveFile(IMAGE *pImage, const char *fname);

void iioMallocImageBuffer(IMAGE *pImage);
void iioFreeImageBuffer(IMAGE *pImage);
```

- 画像処理関連

```
void ipRotateImage(IMAGE *pImage);
```

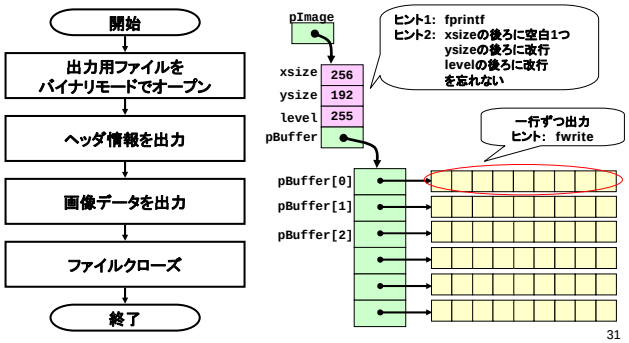
サンプルプログラム2の説明



30

画像ファイルのセーブ

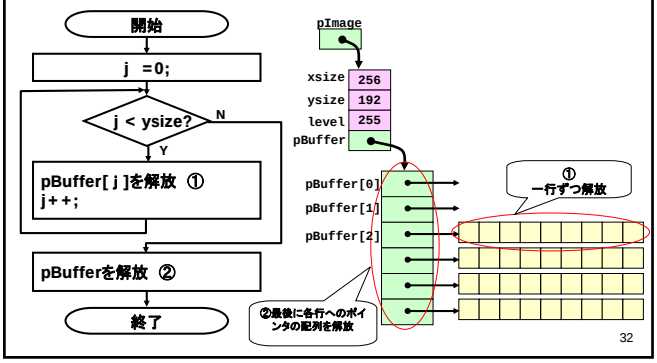
int iioSaveFile(IMAGE *pImage, const char *fname)



31

画像バッファの解放

void iioFreeImageBuffer(IMAGE *pImage)



32