情報システムプログラミングⅡ（18回目）

2024年10月16日（水）

3～4限

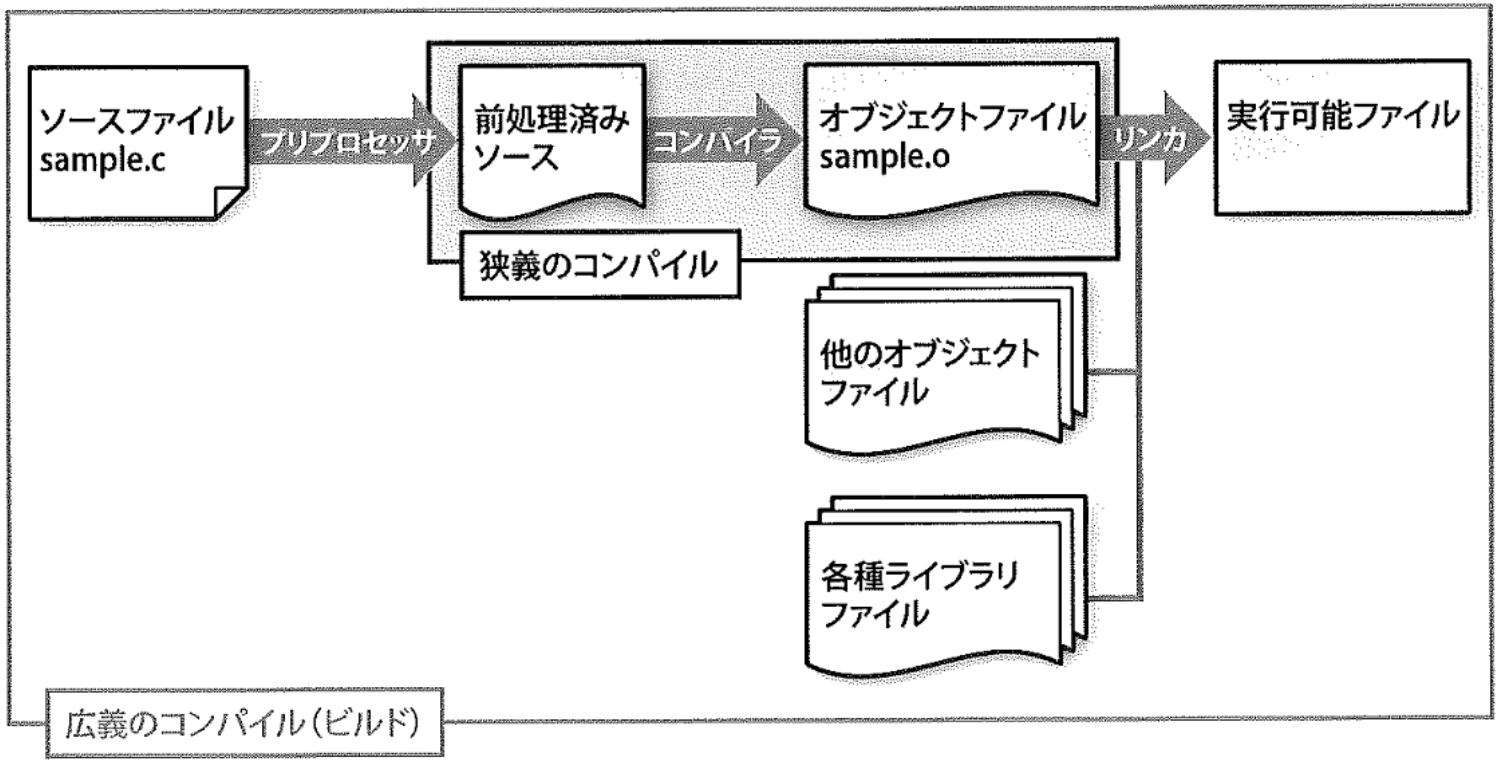
授業内容

- 講義内容（教科書の490～527ページ）
  - ビルドシステム
  - プリプロセッサ
  - コンパイラとリンカ
  - ソースコードの分割
  - 標準ライブラリ
- 演習課題

ビルドシステム

■ビルドの流れ

- C言語におけるソースコードから実行可能ファイルを生成するための処理（ビルド）の流れは以下の通り



プリプロセッサ

■プリプロセッサとは

- ソースコードをコンパイルできるように前処理を行うもの
- 「#」で始まる部分を目印として以下のような処理を行う
  - インクルード処理
  - マクロ処理
  - コンパイル対象の条件分岐

プリプロセッサ

■インクルード処理

- 「#include」で指定されたファイルに置き換える処理

```
$ gcc -E -P -C code1301.c
```

「-C」はコメントを削除しないオプション

「-P」はプリプロセッサの動作結果に行番号を表示しないオプション

```
// コメントA
#include <stdio.h>

// コメントB
int main(void)
{
  printf("hello");
  return 0;
}
```


実行結果

```
// コメントA
 ⋮
typedef unsigned long size_t;
typedef long size_t;
typedef long off_t;
typedef struct _IO_FILE FILE;
 ⋮
 (略)
 ⋮
// コメントB
int main(void) {
  printf("hello");
  return 0;
}
```

stdio.h の内容がここに展開されている

プリプロセッサ

■インクルード処理

- 「#include」によりソースファイルを分割できる
- 囲む文字の違いにより動作が異なる

#include < ファイル名 >

コンパイラが定めるインクルード用ディレクトリからファイルを探す。

#include " ファイル名 "

コンパイル中のファイルのあるディレクトリからファイルを探す。

※見つからなければコンパイラが定めるインクルード用ディレクトリからファイルを探す。

akagi.c

```
#include <stdio.h>

void akagi(void)
{
  printf("akagi!");
}
```

gcc misaki.c

misaki.c

```
#include <stdio.h>
#include "akagi.c"

int main(void)
{
  akagi();
  printf("misaki!");
  return 0;
}
```

プリコンパイル時に akagi.c の内容に置き換えられる

プリプロセッサ

■マクロによる定数（マクロ定数）の利用

- 「#define」で文字列を置換（定数として利用）できる

#define 置換前の文字列 置換後の文字列

```
#include <stdio.h>

#define PI 3.1415

int main(void)
{
  printf("%f", 2 * 2 * PI);
  return 0;
}
```

プリコンパイル時に、PI が 3.1415 に置き換えられる

置き換わることを「展開」ともいう

プリプロセッサ

■マクロによる関数（マクロ関数）の利用

- 「#define」で引数を伴う置換（関数のように利用）もできる

#define 置換前の文字列 (引数 , …) 置換後の文字列

```
#include <stdio.h>

#define ADD(X,Y) X+Y

int main(void)
{
  printf("%d", ADD(3, 10));
  return 0;
}
```

プリプロセッサ

■マクロ定数とマクロ関数の注意点

- 型や構文の文法が確認されない
- 予約語なども変更できてしまう
  - ➢ 定数であればconstなどを利用した方がよい

```
const double PI = 3.1415;
```

- 引数の展開で予期せぬ動作となる可能性あり
  - ➢ マクロ関数の置換後文字列は「( )」で囲む

マクロを積極的に利用する必要はない（利用しなくてよい）

プリプロセッサ

■定義済みマクロ定数

- デバッグ時などに用いられる代表的なものは以下の通り

__FILE__ ：このマクロが書かれているソースファイル名
__LINE__ ：このマクロが記述された位置（ソースファイル内での行番号）
__DATE__：プリプロセッサが起動された日付
__TIME__：プリプロセッサが起動された時刻

プリプロセッサ

■条件付きコンパイル

- 指定したマクロ定数の有無により，ソースコードを部分的に有効または無効にできる

#ifdef マクロ定数 ～ #endif

マクロ定数が宣言されていれば有効とする。

#ifndef マクロ定数～ #endif

マクロ定数が宣言されていなければ有効とする。

#if 条件式 ～ #elif 条件式 ～ #else ～ #endif

合致する条件式に記述された処理を有効とする。どの条件式にも合致しないときは #else に記述された処理を有効とする。

プリプロセッサ

■条件付きコンパイル

- このプログラムでは,
  「DEBUG_MODE」に「5」が
  設定されているため,
  「#else」の部分が有効となる

```
#define DEBUG_MODE 5

int main(void)
{
  int x = 0;

#ifdef DEBUG_MODE          ← DEBUG_MODE が宣言されていれば有効
  printf("This is DEBUG MODE!¥n");
#endif

#ifndef DEBUG_MODE         ← DEBUG_MODE が宣言されていなければ有効
  printf("This is RELEASE MODE!¥n");
#endif

#if (DEBUG_MODE == 1)      ← DEBUG_MODE が 1 のとき有効
  x = 1;
#elif (DEBUG_MODE == 2)    ← DEBUG_MODE が 2 のとき有効
  x = 2;
#elif (DEBUG_MODE == 3)    ← DEBUG_MODE が 3 のとき有効
  x = 3;
#else
  x = 9;
#endif

  printf("%d", x);
```

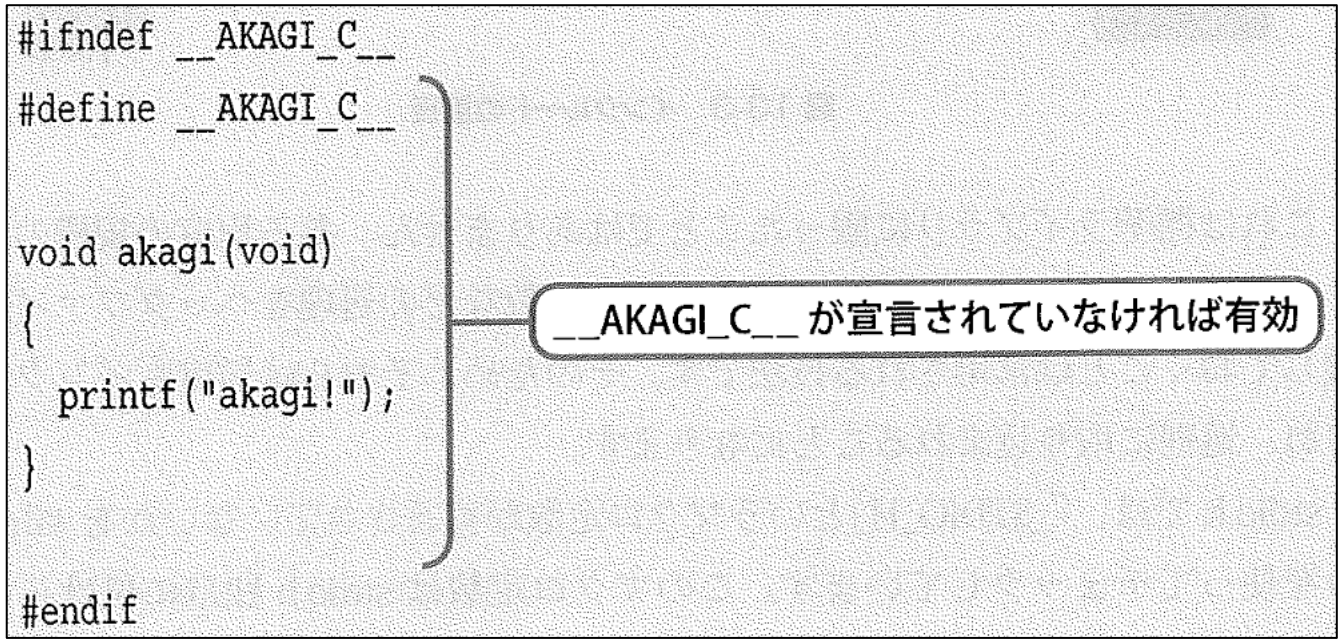
実行結果

```
This is DEBUG MODE!
9
```

プリプロセッサ

■条件付きコンパイルの利用例

- 実行環境やデバッグ作業に応じたコードを記述する
- インクルードの重複を回避できる（インクルードガード）



既に「akagi.c」（このコードが記述されたファイル）がインクルードされていたら無視する

コンパイラとリンカ

■コンパイラ

- プリプロセッサの出力を，文法を確認した上で機械語に変換（コンパイル）するものであり，C言語だとGCCなどがある
  - ➢ オブジェクトファイル（～.oのファイル）が出力される

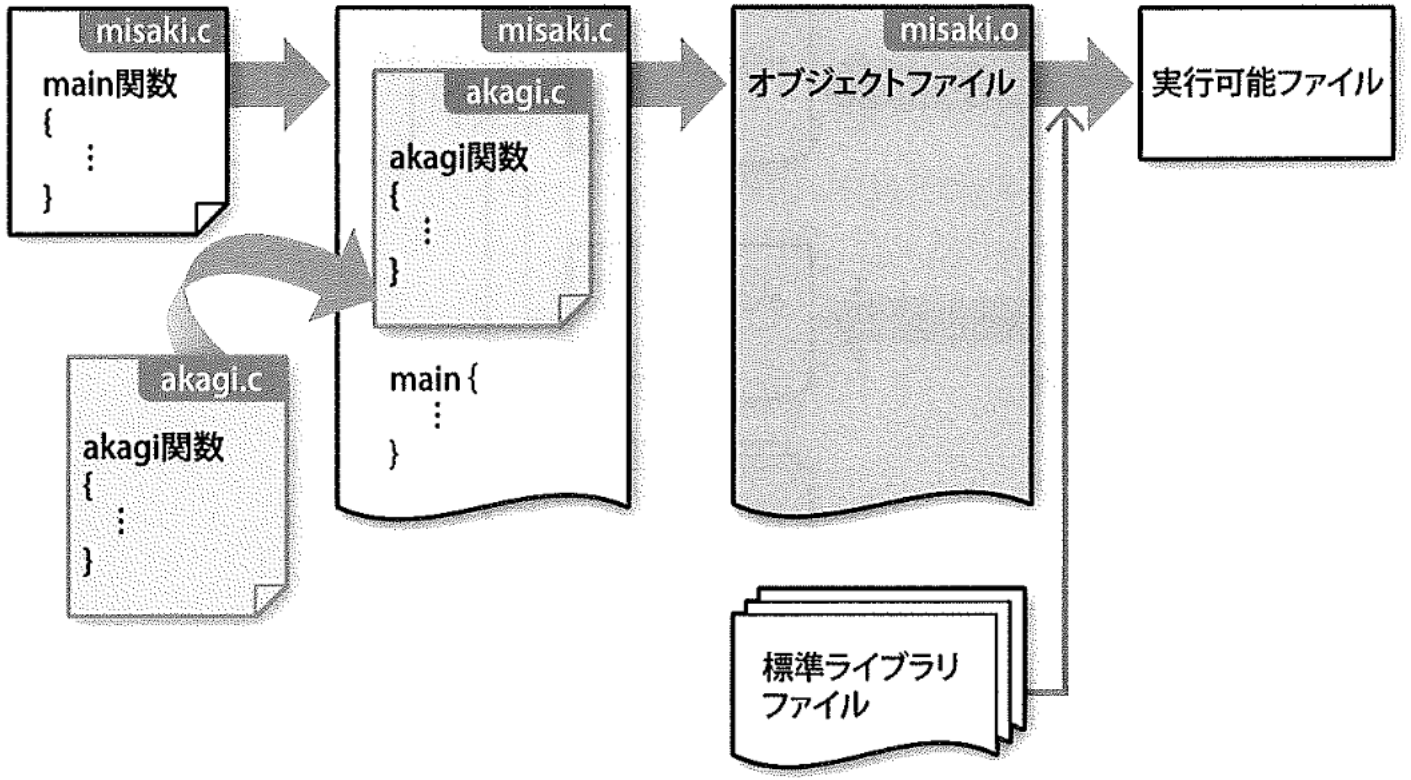
■リンカ

- コンパイラの出力（オブジェクトファイル）を結合して，実行可能ファイルを作成するもの

ソースコードの分割

■コンパイルの効率化

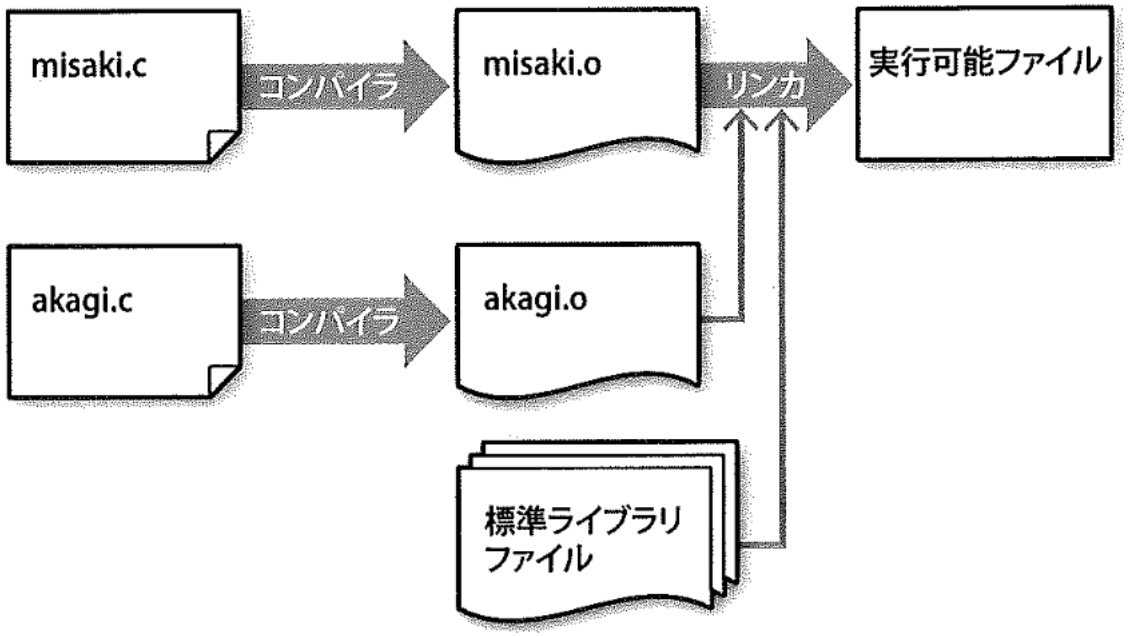
- インクルードされるファイルが多いと効率が悪い



ソースコードの分割

■コンパイルの効率化

- 個別にコンパイルしてオブジェクトファイルを結合する方が効率が良い



ソースコードの分割

■コンパイルの効率化

- このプログラムでは，プロトタイプ宣言を含む「akagi.h」を用意することで「misaki.c」単体でのコンパイルが可能

akagi.h

```
#ifndef __AKAGI_H__
#define __AKAGI_H__

void akagi(void);

#endif
```

ヘッダファイルでは akagi() が存在することだけを宣言しておく

misaki.c

```
#include <stdio.h>
#include "akagi.h"

int main(void)
{
  akagi();
  printf("misaki!");
  return 0;
}
```

akagi() の存在だけをインクルード

akagi() を呼び出しても OK

akagi.c

```
#include <stdio.h>
#include "akagi.h"

void akagi(void)
{
  printf("akagi!");
}
```

akagi() の本体を記述

➢ コンパイル後は「akagi.o」と結合して実行ファイルを生成

```
$ gcc misaki.c akagi.o⏎
```

misaki.c から misaki.o を作り、コンパイル済みの akagi.o と結合する

標準ライブラリ

■標準ライブラリとは

- どのようなc言語の処理系でも用意されているヘッダファイル
- 代表的な標準ライブラリは以下の通り

assert.h	complex.h	ctype.h	errno.h
float.h	limits.h	locale.h	math.h
setjmp.h	signal.h	stdarg.h	stddef.h
stdio.h	stdlib.h	string.h	time.h