# I117（18）分割コンパイルと make

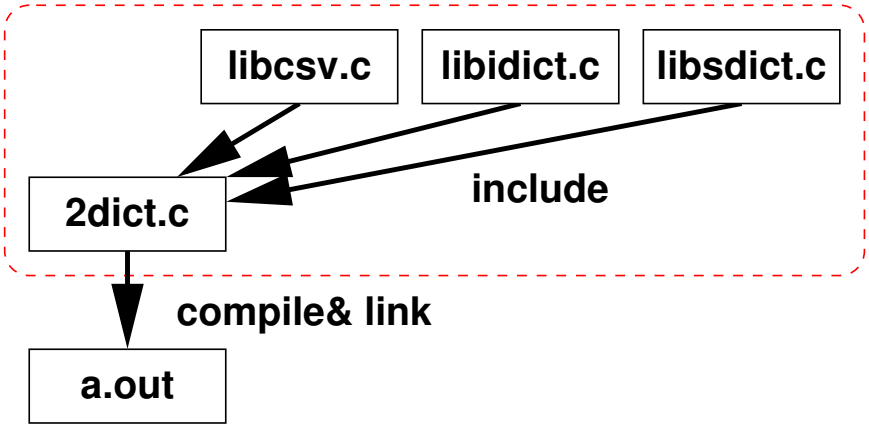
知念

北陸先端科学技術大学院大学 情報科学研究科
School of Information Science,
Japan Advanced Institute of Science and Technology

# 分割コンパイル

CSV のデータ活用の演習で作った2段辞書の出納プログラムを対象にする
このプログラムは4つのファイルが必要で、2dict.c から libcsv.c libidict.c libsdic.c をインクルード

```
% cc 2dict.c
```




今回は .c ファイルが4つなのでさほど問題ないが、ファイルが増えると全てをインクルードするのは大変
そこで、分割してコンパイルする方法を紹介する。

# 分割コンパイル *(cont.)*

2dict.c が libcsv 等をインクルードしているのは 2つの意味がある

a) 型定義や各種宣言

b) 変数や関数の実体

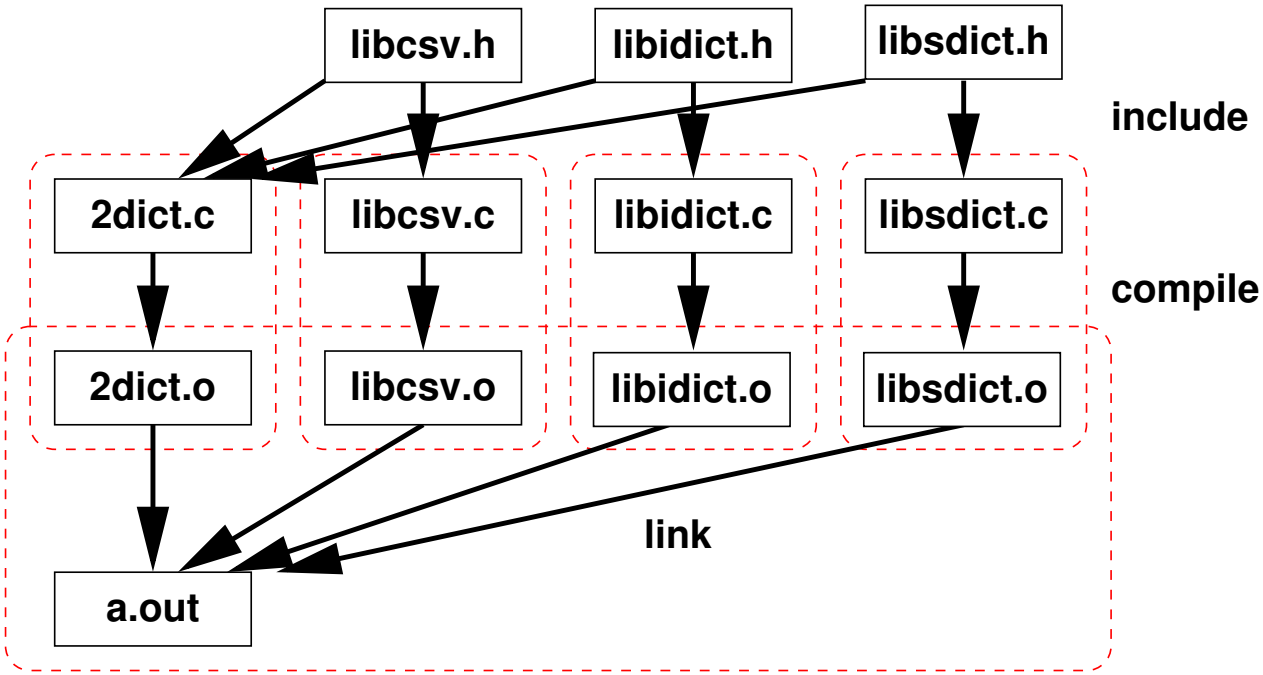
このうち、a) がないと 2dict.c はコンパイルできない
また、b) は最終的になリンク時点であれば問題ない
型定義や各種宣言を抜きだして、libcsv.h libidict.h lib-sdic.h を作る

# 分割コンパイル *(cont.)*

型定義や各種宣言を抜きだして、libcsv.h libidict.h libsdic.h を作る

- 型定義をコピー
- 変数宣言をコピー
- 初期化は消す

2dict.c から .h をインクルードする



# 分割コンパイル *(cont.)*

| | *old libcsv.c* | *libcsv.h* | *new libcsv.c* |
|---|---|---|---|
| マクロ | #define ISLF ... | #define ISLF ... | なし |
| 変数 | int ch=-1; | extern int ch; | int ch=-1; |
| 関数 | int readch() {... | int readch(); | int readch() {... |
| | *old libidict.c* | *libidict.h* | *new libidict.c* |
| 型 | typedef struct {... | typedef struct {... | なし |

# libcsv.h

```
/* CSV, accroding to RFC4180 */
#define ISCOMMA     (ch==0x2c)
#define ISCR        (ch==0x0d)
#define ISDQUOTE    (ch==0x22)
#define ISLF        (ch==0x0A)
#define ISTEXT      \
        ( (ch>=0x20&&ch<=0x21)|| \
        (ch>=0x23&&ch<=0x2b)|| \
        (ch>=0x2d&&ch<=0x7e)|| \
        (ch>=0x80&&ch<=0xff))
extern int  hasheader;
extern int  vused, vlen;
extern char value[BUFSIZ];
extern int  ch, rno, fno;
extern int (*readfield_callback)(int,int,char*);
extern int (*readrecord_callback)(int);
```

```
int readch();
int clearvalue();
int addch();
int readescaped();
int readnonescaped();
int readfield();
int readname();
int readrecord();
int readheader();
int readfile();
```

# libidict.h

```
typedef struct {    typedef struct {
    char *key;          idict_c *slot;
    int   value;        int      len;
} idict_c;              int      use;
                    } idict_a;
#ifndef IDICTLEN
#define IDICTLEN     (10)
#endif
int idict_keycmp(const void *ap, const void *bp);
int idict_valuecmp(const void *ap, const void *bp);
int idict_init(idict_a *dict);
idict_a *idict_new();
int idict_expand(idict_a *dict);
int idict_print(idict_a *dict);
idict_c *idict_findpos(idict_a *dict, char *xkey);
int idict_add(idict_a *dict, char *xkey, int xvalue);
```

# libsdict.h

```
typedef struct {    typedef struct {
    char *key;          sdict_c *slot;
    char *value;        int      len;
} sdict_c;              int      use;
                    } sdict_a;
#ifndef SDICTLEN
#define SDICTLEN     (10)
#endif
int sdict_keycmp(const void *ap, const void *bp);
int sdict_valuecmp(const void *ap, const void *bp);
int sdict_init(sdict_a *dict);
sdict_a *sdict_new();
int sdict_expand(sdict_a *dict);
int sdict_print(sdict_a *dict);
sdict_c *sdict_findpos(sdict_a *dict, char *xkey);
int sdict_add(sdict_a *dict, char *xkey, char *xvalue);
```

# 2dict.c

2dict.c からヘッダファイル群を読み込む

```
...
#include "libcsv.h"
#include "libidict.h"
#include "libsdict.h"
...
```

二重定義を防ぐため、libcsv.c も libcsv.h を読み込む

- libcsv.c 内には libcsv.h で定義していることを再定義しない

```
#include "libcsv.h"
```

# 2dict.c *(cont.)*

libidict.c や libsdict.c も同様

```
#include "libidict.h"
```

```
#include "libsdict.h"
```

# コンパイル

cc のオプション -c はコンパイルのみを指示する

```
% cc -c libcsv.c
% cc -c libidict.c
% cc -c libsdict.c
% cc -c 2dict.c
% cc libcsv.o libidict.o libsdict.o\
? 2dict.o
```

こうすると、libcsv.c が変更されても libidict.c をコンパイルする必要はない

# コンパイル *(cont.)*

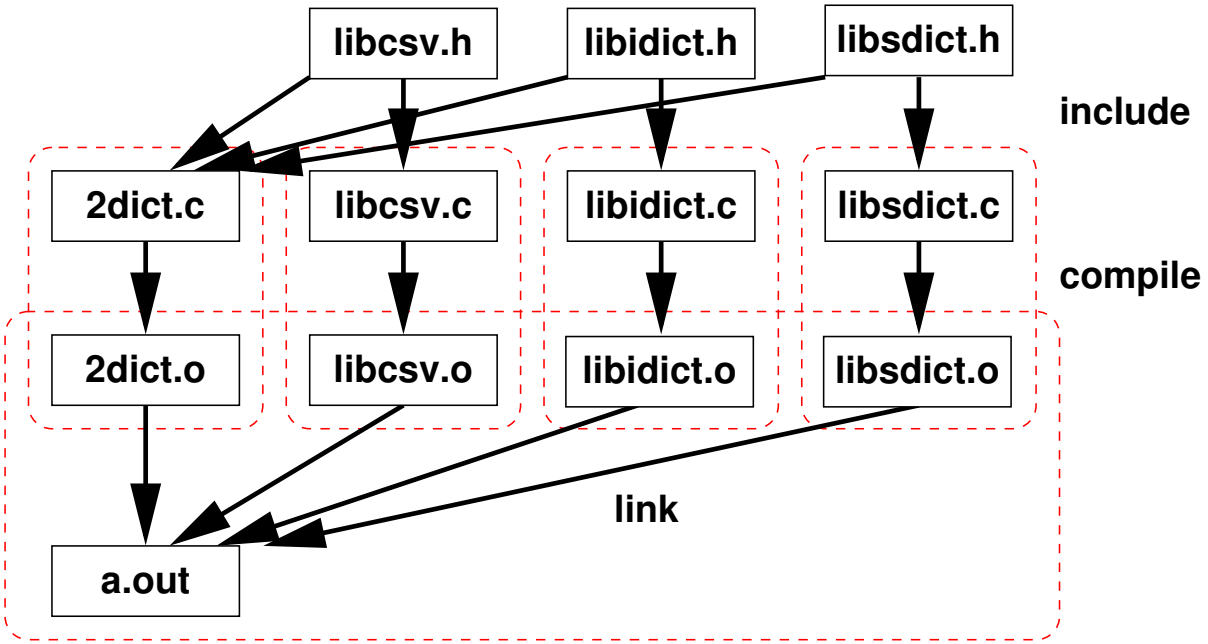
ちなみに、この時点で一括コンパイルも可能、内部で先の４コンパイル１リンクが行われる

```
% cc libcsv.c libidict.c libsdict.c \
? 2dict.c
libcsv.c:
libidict.c:
libsdict.c:
2dict.c:
```

これでは、2dict.c から全てインクルードした場合とさほど変わらないので今回はこの方向は触れない

# make

make はファイルの依存関係を考慮してプログラムを作るツール




※ コンパイル以外にも依存関係指向の作業に使われる

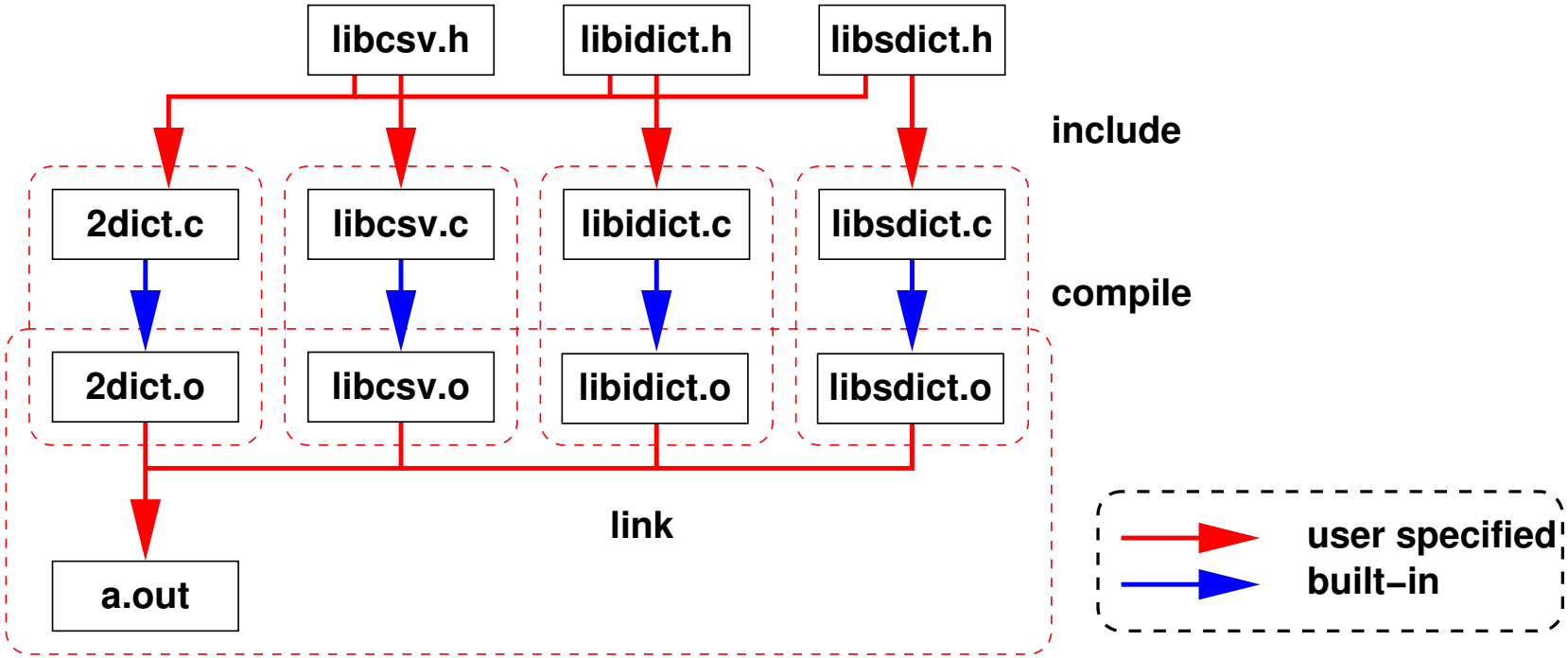
# make *(cont.)*

今回のコンパイルのための設定ファイルの内容は以下のとおり(→はタブ(0x09))

```
a.out: 2dict.o libcsv.o libidict.o libsdict.o
→cc 2dict.o libcsv.o libidict.o libsdict.o
2dict.o: libcsv.h libidict.h libsdict.h
libcsv.o: libcsv.h
libidict.o: libidict.h
libsdict.o: libsdict.h
```

- .o は .c から作る、等の典型的なルールは内蔵
- 依存関係と今回の特別の処理を記述

# make *(cont.)*




図中の内蔵されていない依存関係の矢印は 5本
設定ファイルには 5件のルールを記述

# make *(cont.)*

## 設定ファイル

- 名前は makefile か Makefile
  - この二つは自動的に探す
  - それ以外は -f で指定する

```
% make -f ourmakefile
```

  - GNU make 等は他にも読み込むファイルがある
- 最初のルールが最終的な目標として認識される
  - 先の例では a.out が最終ターゲット
  - 順序を間違えると想定外の動きをする

# make *(cont.)*

## 文法

```
マクロ1=値1
マクロ2=値2
...
ターゲットファイル1: 依存するファイル1 依存するファイル2 ...
→作り方1
ターゲットファイル2: 依存するファイル1 依存するファイル2 ...
→作り方2
...
```

## ※ タブが必要な点に注意

# make *(cont.)*

## 実行例

```
% make
cc   -c  2dict.c
cc   -c  libcsv.c
cc   -c  libidict.c
cc   -c  libsdict.c
cc 2dict.o libcsv.o libidict.o libsdict.o
```

# make *(cont.)*

libcsv.c が変更されると(今回は touch)

```
% touch libcsv.c
% make
cc   -c  libcsv.c
cc 2dict.o libcsv.o libidict.o libsdict.o
```

想定通り libcsv.c がコンパイルされ、リンクしなおし

※ touchはファイルの日付を変更するプログラムで実行時の日付になる

# make *(cont.)*

libcsv.h が変更されると(今回は touch)

```
% touch libcsv.h
% make
cc   -c  2dict.c
cc   -c  libcsv.c
cc 2dict.o libcsv.o libidict.o libsdict.o
```

想定通り、libcsv.c だけでなく 2dict.c もコンパイルされる

依存関係にそって、変更の影響がある分だけコンパイルできる

# make コンパイルエラー

- make は呼び出したプログラムの失敗（終了コードが0以外）を監視している
- コンパイラはコンパイルに失敗すると終了コードをセットする

```
% make
cc   -c  2dict.c
cc   -c  libcsv.c
"libcsv.c", line 11: #error:
cc: acomp failed for libcsv.c
*** Error code 2
make: Fatal error: Command failed for target ‘lib
```

# make コンパイルエラー *(cont.)*

- make はプログラムが失敗するとそこで停止する
  - つまり、それ以降のコンパイルをしない
- 終了コードが 0 以外でも続行したい場合は、ルールの作り方の前に- を付ける

たとえば、いつでも終了コードを 1 にするプログラム false を使って試してみる

```
a.out: 2dict.o libcsv.o libidict.o libsdict.o
→false
→cc 2dict.o libcsv.o \
→libidict.o libsdict.o
```

いつでもエラーでとまる

```
% make
cc   -c  2dict.c
cc   -c  libcsv.c
cc   -c  libidict.c
cc   -c  libsdict.c
false
*** Error code 1
make: Fatal error: Command failed for target ‘a.out’
```

```
a.out: 2dict.o libcsv.o libidict.o libsdict.o
→-false
→cc 2dict.o libcsv.o \
→libidict.o libsdict.o
```

作り方に - をつけると、無視して続行する

```
% make -f Makefile.5
cc   -c  2dict.c
cc   -c  libcsv.c
cc   -c  libidict.c
cc   -c  libsdict.c
false
*** Error code 1 (ignored)
cc  2dict.o libcsv.o \
libidict.o libsdict.o
```

# make clean

Makefile には伝統的につけるターゲットがある（強制ではない）

**all:** 全てのプログラムを作る

**clean:** プログラムや中間ファイルを消す

```
clean:
→rm 2dict.o libcsv.o libidict.o libsdict.o a.out
```

```
% make clean
rm 2dict.o libcsv.o libidict.o libsdict.o a.out
```

こうするとやり直しが簡単

# make の便利な機能

コンパイラや引数などもデフォルトを内蔵している

- $CC コンパイラ
- $CFLAGS コンパイラに渡すオプション群

全てのコンパイルで共通に使うため

```
% make CFLAGS=-g CC=gcc
gcc -g  -c  2dict.c
gcc -g  -c  libcsv.c
gcc -g  -c  libidict.c
gcc -g  -c  libsdict.c
cc 2dict.o libcsv.o libidict.o libsdict.o
```

# make の便利な機能 *(cont.)*

先の Makefile で CC や CFLAGS を使いたい場合は以下のように記述

```
a.out: 2dict.o libcsv.o libidict.o libsdict.o
→$(CC) $(CFLAGS) 2dict.o libcsv.o libidict.o \
libsdict.o
2dict.o: libcsv.h libidict.h libsdict.h
libcsv.o: libcsv.h
libidict.o: libidict.h
libsdict.o: libsdict.h
```

# make の便利な機能 *(cont.)*

make にオプション -p を渡すと内蔵ルールが表示される

```
% make -p |less
```

C言語周りを抜粋すると...

```
.c.o:
→$(COMPILE.c) $(OUTPUT_OPTION) $<
CC= cc
COMPILE.c= $(CC) $(CFLAGS) $(CPPFLAGS) -c
```

CPPFLAGS や OUTPUT__OPTION というマクロもあるらしい

# make の便利な機能 *(cont.)*

RM も定義されている

```
RM= rm -f
```

そこで、clean のルールはこうすると良いだろう

```
clean:
→$(RM) 2dict.o libcsv.o libidict.o \
→libsdict.o a.out
```

※ rm のオプション -f はファイルがなくてもエラーとしない指示

# make の便利な機能 *(cont.)*

## 最終的な Makefile の内容

```
a.out: 2dict.o libcsv.o libidict.o libsdict.o
→$(CC) $(CFLAGS) 2dict.o libcsv.o \
→libidict.o libsdict.o
clean:
→$(RM) 2dict.o libcsv.o libidict.o \
→libsdict.o a.out
2dict.o: libcsv.h libidict.h libsdict.h
libcsv.o: libcsv.h
libidict.o: libidict.h
libsdict.o: libsdict.h
```

# 補足

make はいくつか流派がある

- AT&T SystemV make
- BSD make
- GNU make
- その他

専用の構文で記述すると他で動かないので注意

# 補足 *(cont.)*

## 拡張子のルール追加
.tex から .dvi、.dvi から .pdf を作るルールの例

```
.SUFFIXES: .tex .dvi .pdf $(SUFFIXES)

.tex.dvi:
→platex $<
→platex $<

.dvi.pdf:
→dvipdfmx $<
```

# 補足2 libcsv 内部変数の隠蔽

- 先日述べたように、今回の libcsv.h や libcsv.c の関数や変数名は安直
- 特に変数名は ch, value 等と短く、衝突の恐れ
- 幸いほとんどは libcsv.c 内部でしか使わない
  - libcsv.c 外からアクセスできないようにする
  - 例外 hasheader やコールバック群
- static をつけるとそのファイルの中でのみアクセスを許可する

# 補足2 libcsv 内部変数の隠蔽 *(cont.)*

アクセス不可 — ch, vused, vlen, value, rno, fno

| | *old* | *new* |
|---|---|---|
| *.h* | extern int ch; | なし |
| *.c* | int ch=-1; | static int ch=-1; |

アクセス可能（変更無し）— hasheader, readfield_callback readrecord_callback

| | *old* | *new* |
|---|---|---|
| *.h* | extern int hasheader; | extern int hasheader; |
| *.c* | int hasheader=0; | int heasheader=0; |

**libcsv.h**

```
extern int  hasheader;
extern int (*readfield_callback)(int,int,char*);
extern int (*readrecord_callback)(int);
```

**libcsv.c**

```
int  hasheader=0;
int (*readfield_callback)(int,int,char*)=NULL;
int (*readrecord_callback)(int)=NULL;
static int  ch=-1;
static int  vused;
static int  vlen;
static char value[BUFSIZ];
static int  rno=0;
static int  fno=0;
```

# 演習

1) 実際に libcsv, libidict, libsdict を分割コンパイル向けに変更せよ★
2) libcsv, libidict, libsdict を分割コンパイルするための Makefile を作成せよ★
3) 過去の演習で作った rdict や lldict を分割コンパイル向けのファイルに変更せよ★
4) 過去の演習で作った split と join の関数群を分割コンパイル向けのファイルに変更せよ★