# ctagsの使い方

# tagging（狭義）

ソースコード中、プログラミング言語の文法の元で名前を付けて定義された 「何か」について
名前、ソースファイル、ファイル中の場所、その他の情報を 組として記録すること。

---

「何か」には、例えば関数、マクロ、変数、型などがあります。

# taggingツールと出力フォーマット

| ツール | フォーマット | ツールの出力を活用できるエディタ |
|---|---|---|
| etags | TAGS | emacs |
| ctags | tags | vi, less, 様々 |

ctagsに -e オプションをつけると TAGSフォーマットでも出力できます。

# tags出力の例

## 入力（input.c）

```
#include <stdio.h>
static boolean debug;
#include DEBUG(X) (debug=X)

struct point2d {int x, y;};
typedef struct point2d *pp;

int distance (pp p0, pp p1) {   /* ... */}
int main (int argc, char** argv) {

  int local;
/* ... */
 goto_label:
  return 0;
}
```

# tags出力の例

## 出力（tags）

```
$ ctags -o - input.c
debug       input.c /^static boolean debug;$/;”     v       file:
distance    input.c /^int distance (pp p0, pp p1) {   ¥/* ... *¥/}$/;”      f
main        input.c /^int main (int argc, char** argv) {$/;”        f
point2d     input.c /^struct point2d {int x, y;};$/;”       s       file:
pp  input.c /^typedef struct point2d *pp;$/;”       t       typeref:struct:point2d  file:
x   input.c /^struct point2d {int x, y;};$/;”       m       struct:point2d  file:
y   input.c /^struct point2d {int x, y;};$/;”       m       struct:point2d  file:
```

各行は次の形式をしています。

```
<名前>tab<ファイル名>tab/<行パターン>/;”tab<KIND>tab...
```

<KIND>については後述します。<KIND>の後に「その他」の情報が続きます。

# 注意

ツールの出力は完全とは期待できません。特にプリプロセッサによるマクロ展開 を前提とした入力からは定義をみつけることができません。

```
$ cat hello-macro.c
#include <stdio.h>
#define defineEchoFunction(NAME,MESSAGE) ¥
  int NAME(void) { printf(”%s¥n”, MESSAGE); }

defineEchoFunction (main, ”hello, world)
$ ctags -o - hello-macro.c
defineEchoFunction  hello-macro.c   2;”     d       file:
```

defineEchoFunction マクロが展開されて定義される関数mainについてctags はそれを発見し損ねています。

# 基本的なコマンドラインオプション

```
ctags [オプション] 入力ファイル
```

| オプション | 意味 |
|---|---|
| -R | 「入力ファイル」に指定したディレクトリ以下の全てのファイルを入力とする。 |
| --exclude=pattern | pattern に照合する名前を持つファイルを無視する。 |
| -o OUTPUT | 出力ファイル名を OUTPUT と指定する。- を指定すると標準出力に結果を書き出す。 |
| -L | 入力ファイルのリストを標準入力から受けとる。 |
| --sort=yes\|no | 結果を(出現順ではなく)アルファベット順にソートする。（デフォルト: yes) |
| --filter | 入力ファイルの一覧を標準入力から与える。 |
| -n | 行パターンのかわりに行番号を出力します。 |
| -x | cxref形式で出力する。 |

# 対応する言語

- 品質にバラツキがありますがありますが、様々な言語に対応しています。
  - `--list-languages` で対応する言語を一覧できます。
  - 名前を捕捉する条件を正規表現で記述することで、未知の言語に対応できます。
- 主に拡張子から、言語を推測します。
  - `--lang-map` で 拡張子と言語の対応関係を一覧できます。
  - `--language-force=LANG` で 推測を無効にして特定の言語を想定した処理を強要できます。

# 「その他」の情報（fieldの制御）

- 「その他」の情報をfieldと呼びます。
- 様々なfieldがあります。(universal-ctagsであれば --list-fields で一覧できます。)

kind

「何か」の種別。言語毎に異なる。(後述)

file

「何か」に対する名前の入力ファイルを範囲とするスコープを持つ。 C言語で言うところの static

typeref

- 「何か]が変数の場合、型の名前
- 「何か]が型の別名の場合、元の型

language

入力ファイルの記述言語

など色々あります。

- --fields にはアルファベット一文字の名前がついています。
- --fields=[+|-]F オプションでフィールドFを有効、あるいは無効にできます。

# kind

- 言語LANGに対するkindは --list-kinds=LANG で一覧できます。

  例 C 言語に対するkindの一覧

```
$ ctags --list-kinds=C
c  classes
d  macro definitions
...
l  local variables [off]
```

- デフォルトで無効となっているkindについては [off] と印付けされています。
- --<LANG>-kinds=[+|-]L として kind Lを有効、あるいは無効にできます。

  C言語におけるd(マクロ)を無効にして、l(ローカル変数)を有効にして kind一覧を表示し直した例

```
$ ctags --c-kinds=-d+l --list-kinds=C
c  classes
d  macro definitions [off]
...
l  local variables
```

# C言語に対するkindの制御の例

```c
#include <stdio.h>
int main(void)
{
  const char *msg = "hello, world";
  puts(msg);
  return 0;
}
```

```
$ ctags -o - input2.c
main        input2.c        /^int main(void)$/;"    f
$ ctags -o - --c-kinds=+l input2.c
main        input2.c        /^int main(void)$/;"    f
msg input2.c        /^  const char *msg = "hello, world";$/;"       l
```

# ctagsを用いた演習2の解法

演習2でいくつかのコマンドについて main 関数を探しました。 ctagsを使うと次のようにmain関数を列挙できます。

```
$ cd ~/fedora/coreutils*
$ find . -name '*.c' |
  ctags --filter -o - --c-kinds=f -n |
  grep ^main

 main        ./gnulib-tests/test-dirent.c    29;"    f
 main        ./gnulib-tests/test-c-strcasecmp.c      29;"    f
 main        ./gnulib-tests/uniwidth/test-uc_width.c 26;"    f
 main        ./gnulib-tests/uniwidth/test-uc_width2.c        61;"    f
 ...
```

(紙面の都合で改行を入れています。)

この例では coreutilsのソースコードツリーから拡張子 .c を持つファイルに 限定してC言語の関数(f)の定義箇所を取り出しています。ctagsの出力から grepを使ってmainで始まるものだけに限定しています。テストケースに含まれる main関数が多数表示されるので、さらに ”grep -v gnulib-tests”をつけて、 それらを出力から排除すると良いでしょう。