技術ノート

ツール，ライブラリ，処理環境

中村勝則

2018 年 12 月 14 日

免責事項

本書の内容は参考的資料であり，掲載したプログラムリストは全て試作品である．本書の使用に伴って発生した不利益，損害の一切の責任を筆者は負わない．

目 次

1　ツール群

1.1　文字コード／改行コード変換

1.1.1　iconv

iconv はテキストデータの文字コードを変換するツールとライブラリであり，POSIX 規格で標準化されている．このため iconv は多くの UNIX 系 OS で利用可能な場合が多く，Windows 環境でも POSIX 系ツール群を導入している場合は利用可能なことが多い．ここではコマンドツールとしての iconv の使用方法について説明する．

■ 使い方

iconv コマンドには，変換元のファイルの文字コードと，変換先の文字コードを起動時のオプションとして与える．

コマンド書式： iconv -f **変換元の文字コード** -t **変換先の文字コード　変換元ファイル名**

変換元のデータは iconv の標準入力に与えることもできる．変換先のデータは標準出力に出力される．

コマンドオプションに指定できる文字コードは ‘-l’ オプションで起動することで確認できる．

例. iconv で扱うことのできる文字コードの確認（bash でのコマンド操作の例）

```
$ iconv -l  Enter      ←コマンドの投入
ANSI_X3.4-1968 ANSI_X3.4-1986 ASCII CP367 IBM367 …      ←指定できる文字コードが表示される
UTF-8
          ⋮
        （途中省略）
          ⋮
CP949 UHC
CP1361 JOHAB
ISO-2022-KR CSISO2022KR
```

1.1.2　NKF（Network Kanji Filter）

NKF [1] はテキストデータの文字コードを変換するだけでなく，改行コードの変換など様々な機能を備えている．コマンド名は nkf で，‘–help’ オプションを与えて起動すると基本的な使用方法が表示される．

例. nkf のヘルプ表示（bash でのコマンド操作の例）

```
$ nkf --help  Enter      ←コマンドの投入
Usage:  nkf -[flags] [--] [in file] ..  [out file for -O flag]      ←ヘルプ表示が始まる
  j/s/e/w Specify output encoding ISO-2022-JP, Shift_JIS, EUC-JP
  UTF options is -w[8[0],16,32[B,L[0]]]
          ⋮
        （途中省略）
          ⋮
Network Kanji Filter Version 2.1.4 (2015-12-12)
Copyright (C) 1987, FUJITSU LTD. (I.Ichikawa).
Copyright (C) 1996-2015, The nkf Project.
```

変換元のテキストデータは nkf コマンドの標準入力から与えることができる．また，ファイルとして保存されているテキストデータを変換対象とする場合は，nkf のコマンド引数にそのファイル名を記述する．変換結果は標準出力に出力される．

[1] 1987 年に富士通が開発．現在は OSDN のインターネットサイト（https://ja.osdn.net/projects/nkf/）で保守されている．

例. ファイル `text1.txt` の文字コードを変換して `text2.txt` を作成する例（bash でのコマンド操作の例）

```
$ nkf -w text1.txt > text2.txt
```

この例では，‘-w’ オプションを与えることで，UTF-8 の文字コードに変換している．変換先の文字コードの指定など，出力データに関係するオプションとして重要なものを表 1 に挙げる．

表 1: 出力に関するオプション（一部）

オプション	解説
-j	JIS コードで出力（デフォルト）
-e	EUC コードで出力
-s	シフト JIS コードで出力
-w, -w80	UTF-8 コードで出力（BOM なし）
-w8	UTF-8 コードで出力する（BOM あり）
-w16, -w16B0	UTF-16 コードで出力（ビッグエンディアン／ BOM なし）
-w16B	UTF-16 コードで出力（ビッグエンディアン／ BOM あり）
-w16L	UTF-16 コードで出力（リトルエンディアン／ BOM あり）
-w16L0	UTF-16 コードを出力する（リトルエンディアン／ BOM なし）
-I	ISO-2022-JP 以外の漢字コードを「〓」（げた記号）に変換
-d, -Lu	改行を LF に変換（UNIX 系）
-c, -Lw	改行を CRLF に変換（Windows 系）
-Lm	改行を CR に変換（古い Macintosh）
-X	半角カナ（JIS X 0201 片仮名）を全角カナ（JIS X 0208 片仮名）に変換（デフォルト）
-x	半角カナを維持する
-Z,-Z0	全角を半角に変換（JIS X 0208 英数字といくつかの記号を ASCII に変換）
-Z1	JIS X 0208 和字間隔を ASCII スペース 1 つに変換
-Z2	JIS X 0208 和字間隔を ASCII スペース 2 つに変換
-Z3	「>」「<」「”」「&」をそれぞれ「>」「<」「"」「&」に変換
--hiragana	カタカナをひらがなに変換
--katakana	ひらがなをカタカナに変換
--overwrite	ファイルを変換して上書きする

改行コードに関するオプションをつけない場合は改行コードは変換しない

NKF は変換元のテキストデータの文字コードと改行コードを自動的に判別するが，明にオプション（表 2）として指定することもできる．

表 2: 入力に関するオプション（一部）

オプション	解説
-J	入力を ISO-2022-JP とみなす
-E	入力を日本語 EUC とみなす
-S	入力をシフト JIS とみなす．半角カナ（JIS X 0201 片仮名）も受け入れる
-W,-W8	入力を UTF-8 とみなす
-W16	入力を UTF-16（リトルエンディアン）とみなす
-W16B	入力を UTF-16（ビッグエンディアン）とみなす
--guess	対象ファイルの文字コードと改行コードを調べる

1.1.2.1 MIME 関連の機能

nkf は MIME [2] に関する変換機能を備えている．そのためのオプションを表 3 に示す.

表 3: MIME に関するオプション（一部）

オプション	解説
-m	MIME を解読.「-mS」と同じ（デフォルト）
-mB	base64 stream を解読
-mQ	Quoted stream を解読
-mS	MIME のチェックの厳格化（他のオプションと組み合わせ可）
-mN	MIME のチェックを緩く（他のオプションと組み合わせ可）
-m0	MIME を解読しない
-M	MIME に変換（JIS に変換してから base64 ヘッダ形式に変換）
-MB	base64 stream に変換
-MQ	Quoted stream に変換

1.1.2.2 若干の注意点

iconv と NKF は共に文字コードの変換機能を有するが，両者の処理結果は完全には一致しないことに注意すること.

[2] **MIME**（Multipurpose Internet Mail Extensions）：RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289, RFC 2049

1.2 MeCab(和布蕪)：日本語形態素解析

MeCab は，京都大学情報学研究科－日本電信電話株式会社コミュニケーション科学基礎研究所共同研究ユニットプロジェクトを通じて開発されたオープンソースの形態素解析エンジンである．[3]

MeCab を用いることで日本語の文章を解析して，文章の要素データを生成することができる．基本的に MeCab はコマンドラインツールであり，日本語文章のテキストを標準入力やテキストファイルから入力することで，標準出力やテキストファイルに文章の要素データを出力することができる．

次に示す test11.txt のようなテキストファイルを MeCab に与える例を示す．

サンプルテキスト：test11.txt

```
1  （フリー百科事典「ウィキペディア」より）
2
3  コードを単純化して可読性を高め、読みやすく、また書きやすくしてプログラマの
4  作業性とコードの信頼性を高めることを重視してデザインされた、汎用の高水準言語
5  である。反面、実行速度はCなどの低級言語に比べて犠牲にされている。
6
7  核となる文法（シンタックス）および意味（セマンティクス）は必要最小限に抑え
8  られている。その反面、豊富で大規模な文書（document）や、さまざまな領域に
9  対応する大規模な標準ライブラリやサードパーティ製のライブラリが提供されている。
10 またPythonは多くのハードウェアとOS（プラットフォーム）に対応しており、複数の
11 プログラミングパラダイムに対応している。Pythonはオブジェクト指向、命令型、
12 手続き型、関数型などの形式でプログラムを書くことができる。動的型付け言語であり、
13 参照カウントベースの自動メモリ管理（ガベージコレクタ）を持つ。
14
15 これらの特性により、PythonはWebアプリケーションやデスクトップアプリケーション
16 などの開発はもとより、システム用の記述（script）や、各種の自動処理、理工学や
17 統計・解析など、幅広い領域における支持を得る、有力なプログラム言語となった。
18 プログラミング作業が容易で能率的であることは、ソフトウェア企業にとっては
19 投入人員の節約、開発時間の短縮、ひいてはコスト削減に有益であることから、
20 産業分野でも広く利用されている。Googleなど主要言語に採用している企業も多い。
```

■ MeCab の実行

コマンドの書式： mecab test11.txt

この結果，次のような文章の要素データが標準出力に出力される．

出力例

```
（              記号,括弧開,*,*,*,*,（,（,（
フリー          名詞,一般,*,*,*,*,フリー,フリー,フリー
百科            名詞,一般,*,*,*,*,百科,ヒャッカ,ヒャッカ
事典            名詞,一般,*,*,*,*,事典,ジテン,ジテン
「              記号,括弧開,*,*,*,*,「,「,「
ウィキペディア  名詞,固有名詞,一般,*,*,*,*
」              記号,括弧閉,*,*,*,*,」,」,」
事典            名詞,一般,*,*,*,*,事典,ジテン,ジテン
                        :
                    (途中省略)
                        :
EOS
コード 名詞,一般,*,*,*,*,コード,コード,コード
を 助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
                        :
                    (以下省略)
                        :
```

この例にあるように，1 行の解析毎に EOS という行が出力される．

入力ファイルを指定せずに MeCab を起動すると，標準入力から文章の入力を受け付ける．

MeCab の出力の形式は次の通りである．

[3] この紹介文は MeCab のインターネットサイト http://taku910.github.io/mecab/ からの引用を元にしている．

表層形（タブ）品詞, 品詞細分類 1, 品詞細分類 2, 品詞細分類 3, 活用型, 活用形, 原形, 読み, 発音

MeCab は起動時に，出力先や出力の形式を変更するためのオプションを与えることができる．

出力に関するオプション（一部）

-O wakati　　**表層形**のデータのみをスペースで区切った形（分かち書き形式）で出力する．

-o **出力ファイル名**　　**出力ファイル名**に指定したファイルに出力する．

分かち書き出力の例

```
mecab -O wakati test11.txt
```

この結果次のような出力となる．

分かち書きの出力例

```
（ フリー 百科 事典 「 ウィキペディア 」 より ）
コード を 単純 化 し て 可読性 を 高め 、 読み やすく 、 また 書き やすく し て プログラマ
の 作業 性 と コード の 信頼 性 を 高める こと を 重視 し て デザイン さ れ た 、 汎用 の
高水準 言語 で ある 。
                    :
                （以下省略）
                    :
```

1.3　CaboCha(南瓜)：日本語係り受け解析

CaboCha(南瓜) は，MeCab の形態素解析の結果を用いて，日本語の文の係り受け構造を解析するツールである．CaboCha に関する詳しい情報は，公式インターネットサイト https://taku910.github.io/cabocha/ から得られる．

CaboCha は基本的にはコマンドツールであり，OS のコマンドシェルにて cabocha コマンドを投入することで起動する．例えば jtext-u.txt のようなテキストファイルの係り受け構造の解析について考える．

サンプルテキスト：jtext-u.txt

```
1 この文のような複雑な係り受け構造を持つ文を解析することができると人工知能プログラムのためのヒューマンインター
      フェースを開発するのに役立ちます.
```

このテキストファイルを次のようなコマンドで処理をする．

```
cabocha jtext-u.txt
```

これにより，次のような結果（木構造）が標準出力に出力される．

```
      この-D
    文のような----------D
          複雑な-D      |
          係り受け---D  |
              構造を-D  |
                  持つ-D
                    文を-D
                  解析する-D
                      ことが-D
                      できると-------D
            人工知能プログラムの-D    |
                              ための-D |
        ヒューマンインターフェースを-D
                          開発するのに-D
                            役立ちます.
EOS
```

また，cabocha コマンドに ‘-f1’ オプションを付けて実行する（`cabocha -f1 jtext-u.txt`）と次のような出力となり，機械可読な解析結果が得られる．

出力結果：

```
* 0 1D 0/0 0.047241
この　　連体詞,*,*,*,*,*,この,コノ,コノ
* 1 6D 2/3 1.151904
文　名詞,一般,*,*,*,*,文,ブン,ブン
の　助詞,連体化,*,*,*,*,の,ノ,ノ
よう　　名詞,非自立,助動詞語幹,*,*,*,よう,ヨウ,ヨー
な　助動詞,*,*,*,特殊・ダ,体言接続,だ,ナ,ナ
* 2 3D 0/1 0.705414
複雑　　名詞,形容動詞語幹,*,*,*,*,複雑,フクザツ,フクザツ
な　助動詞,*,*,*,特殊・ダ,体言接続,だ,ナ,ナ
* 3 5D 1/1 0.694835
係り　　名詞,一般,*,*,*,*,係り,カカリ,カカリ
受け　　動詞,自立,*,*,一段,連用形,受ける,ウケ,ウケ
* 4 5D 0/1 2.508135
構造　　名詞,一般,*,*,*,*,構造,コウゾウ,コーゾー
を　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
* 5 6D 0/0 1.335883
持つ　　動詞,自立,*,*,五段・タ行,基本形,持つ,モツ,モツ
* 6 7D 0/1 2.262625
文　名詞,一般,*,*,*,*,文,ブン,ブン
を　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
* 7 8D 1/1 1.993750
解析　　名詞,サ変接続,*,*,*,*,解析,カイセキ,カイセキ
する　　動詞,自立,*,*,サ変・スル,基本形,する,スル,スル
* 8 9D 0/1 1.854513
こと　　名詞,非自立,一般,*,*,*,こと,コト,コト
が　助詞,格助詞,一般,*,*,*,が,ガ,ガ
* 9 13D 0/1 0.124090
できる　動詞,自立,*,*,一段,基本形,できる,デキル,デキル
と　助詞,接続助詞,*,*,*,*,と,ト,ト
* 10 11D 2/3 1.869991
人工　　名詞,一般,*,*,*,*,人工,ジンコウ,ジンコー
知能　　名詞,一般,*,*,*,*,知能,チノウ,チノー
プログラム　名詞,サ変接続,*,*,*,*,プログラム,プログラム,プログラム
の　助詞,連体化,*,*,*,*,の,ノ,ノ
* 11 12D 0/1 2.088981
ため　　名詞,非自立,副詞可能,*,*,*,ため,タメ,タメ
の　助詞,連体化,*,*,*,*,の,ノ,ノ
* 12 13D 1/2 1.876101
ヒューマン　名詞,一般,*,*,*,*,ヒューマン,ヒューマン,ヒューマン
インターフェース　　名詞,一般,*,*,*,*,インターフェース,インターフェース,インターフェース
を　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ
* 13 14D 2/3 0.124090
開発　　名詞,サ変接続,*,*,*,*,開発,カイハツ,カイハツ
する　　動詞,自立,*,*,サ変・スル,基本形,する,スル,スル
の　名詞,非自立,一般,*,*,*,の,ノ,ノ
に　助詞,格助詞,一般,*,*,*,に,ニ,ニ
* 14 -1D 0/1 0.000000
役立ち　動詞,自立,*,*,五段・タ行,連用形,役立つ,ヤクダチ,ヤクダチ
ます　　助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス
.　記号,句点,*,*,*,*,.　,.　,.
EOS
```

上記出力結果の書式に関しては cabocha の公式インターネットサイトなどを参照のこと.

CaboCha では実行時のコマンドオプションとして '-f3' を与えると，解析結果を XML 形式で出力することもできる．出力例を次に示す.

出力結果：

```
<sentence>
 <chunk id="0" link="1" rel="D" score="0.047241" head="0" func="0">
  <tok id="0" feature="連体詞,*,*,*,*,*,この,コノ,コノ">この</tok>
 </chunk>
 <chunk id="1" link="6" rel="D" score="1.151904" head="3" func="4">
  <tok id="1" feature="名詞,一般,*,*,*,*,文,ブン,ブン">文</tok>
  <tok id="2" feature="助詞,連体化,*,*,*,*,の,ノ,ノ">の</tok>
  <tok id="3" feature="名詞,非自立,助動詞語幹,*,*,*,よう,ヨウ,ヨー">よう</tok>
  <tok id="4" feature="助動詞,*,*,*,特殊・ダ,体言接続,だ,ナ,ナ">な</tok>
 </chunk>
 <chunk id="2" link="3" rel="D" score="0.705414" head="5" func="6">
  <tok id="5" feature="名詞,形容動詞語幹,*,*,*,*,複雑,フクザツ,フクザツ">複雑</tok>
  <tok id="6" feature="助動詞,*,*,*,特殊・ダ,体言接続,だ,ナ,ナ">な</tok>
 </chunk>
 <chunk id="3" link="5" rel="D" score="0.694835" head="8" func="8">
  <tok id="7" feature="名詞,一般,*,*,*,*,係り,カカリ,カカリ">係り</tok>
  <tok id="8" feature="動詞,自立,*,*,一段,連用形,受ける,ウケ,ウケ">受け</tok>
 </chunk>
 <chunk id="4" link="5" rel="D" score="2.508135" head="9" func="10">
  <tok id="9" feature="名詞,一般,*,*,*,*,構造,コウゾウ,コーゾー">構造</tok>
  <tok id="10" feature="助詞,格助詞,一般,*,*,*,を,ヲ,ヲ">を</tok>
 </chunk>
 <chunk id="5" link="6" rel="D" score="1.335883" head="11" func="11">
  <tok id="11" feature="動詞,自立,*,*,五段・タ行,基本形,持つ,モツ,モツ">持つ</tok>
 </chunk>
 <chunk id="6" link="7" rel="D" score="2.262625" head="12" func="13">
  <tok id="12" feature="名詞,一般,*,*,*,*,文,ブン,ブン">文</tok>
  <tok id="13" feature="助詞,格助詞,一般,*,*,*,を,ヲ,ヲ">を</tok>
 </chunk>
 <chunk id="7" link="8" rel="D" score="1.993750" head="15" func="15">
  <tok id="14" feature="名詞,サ変接続,*,*,*,*,解析,カイセキ,カイセキ">解析</tok>
  <tok id="15" feature="動詞,自立,*,*,サ変・スル,基本形,する,スル,スル">する</tok>
 </chunk>
 <chunk id="8" link="9" rel="D" score="1.854513" head="16" func="17">
  <tok id="16" feature="名詞,非自立,一般,*,*,*,こと,コト,コト">こと</tok>
  <tok id="17" feature="助詞,格助詞,一般,*,*,*,が,ガ,ガ">が</tok>
 </chunk>
 <chunk id="9" link="13" rel="D" score="0.124090" head="18" func="19">
  <tok id="18" feature="動詞,自立,*,*,一段,基本形,できる,デキル,デキル">できる</tok
      >
  <tok id="19" feature="助詞,接続助詞,*,*,*,*,と,ト,ト">と</tok>
 </chunk>
 <chunk id="10" link="11" rel="D" score="1.869991" head="22" func="23">
  <tok id="20" feature="名詞,一般,*,*,*,*,人工,ジンコウ,ジンコー">人工</tok>
  <tok id="21" feature="名詞,一般,*,*,*,*,知能,チノウ,チノー">知能</tok>
  <tok id="22" feature="名詞,サ変接続,*,*,*,*,プログラム,プログラム,プログラム">プ
      ログラム</tok>
  <tok id="23" feature="助詞,連体化,*,*,*,*,の,ノ,ノ">の</tok>
 </chunk>
 <chunk id="11" link="12" rel="D" score="2.088981" head="24" func="25">
  <tok id="24" feature="名詞,非自立,副詞可能,*,*,*,ため,タメ,タメ">ため</tok>
  <tok id="25" feature="助詞,連体化,*,*,*,*,の,ノ,ノ">の</tok>
 </chunk>
 <chunk id="12" link="13" rel="D" score="1.876101" head="27" func="28">
  <tok id="26" feature="名詞,一般,*,*,*,*,ヒューマン,ヒューマン,ヒューマン">ヒュー
      マン</tok>
  <tok id="27" feature="名詞,一般,*,*,*,*,インターフェース,インターフェース,インタ
      ーフェース">インターフェース</tok>
  <tok id="28" feature="助詞,格助詞,一般,*,*,*,を,ヲ,ヲ">を</tok>
 </chunk>
 <chunk id="13" link="14" rel="D" score="0.124090" head="31" func="32">
  <tok id="29" feature="名詞,サ変接続,*,*,*,*,開発,カイハツ,カイハツ">開発</tok>
  <tok id="30" feature="動詞,自立,*,*,サ変・スル,基本形,する,スル,スル">する</tok>
  <tok id="31" feature="名詞,非自立,一般,*,*,*,の,ノ,ノ">の</tok>
  <tok id="32" feature="助詞,格助詞,一般,*,*,*,に,ニ,ニ">に</tok>
 </chunk>
 <chunk id="14" link="-1" rel="D" score="0.000000" head="33" func="34">
  <tok id="33" feature="動詞,自立,*,*,五段・タ行,連用形,役立つ,ヤクダチ,ヤクダチ">
```

```
        役立ち</tok>
65    <tok id="34" feature="助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス">ます</tok>
66    <tok id="35" feature="記号,句点,*,*,*,*,. ,. ,. ">. </tok>
67   </chunk>
68  </sentence>
```

上記出力結果の書式に関しては cabocha の公式インターネットサイトなどを参照のこと.

1.4 JUMAN：日本語形態素解析

JUMAN は，京都大学大学院情報学研究科で開発[4] された日本語形態素解析プログラムである．JUMAN に関する情報はインターネットサイト

http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN

で公開されている．

JUMAN を用いることで日本語の文章を解析して，文章の要素データを生成することができる．基本的に JUMAN はコマンドラインツールであり，日本語文章のテキストを標準入力から入力することで，文章の要素データを出力することができる．(次の例参照)

■ JUMAN の実行例

コマンドの書式： juman

この後，標準入力からの文書の入力が受け付けられ，処理の結果次のような文章の要素データが標準出力に出力される．

例. juman の実行

```
C:￥Users￥katsu>juman   Enter      ←juman コマンドの投入
これは日本語の文です.   Enter      ←日本語文の入力
これ これ これ 指示詞 7 名詞形態指示詞 1 * 0 * 0 NIL      ←解析結果の表示
は は は 助詞 9 副助詞 2 * 0 * 0 NIL
日本 にほん 日本 名詞 6 地名 4 * 0 * 0 "代表表記:日本/にほん 地名:国"
@ 日本 にっぽん 日本 名詞 6 地名 4 * 0 * 0 "代表表記:日本/にほん 地名:国"
語 ご 語 名詞 6 普通名詞 1 * 0 * 0 "代表表記:語/ご 漢字読み:音 カテゴリ:抽象物"
の の の 助詞 9 接続助詞 3 * 0 * 0 NIL
文 ぶん 文 名詞 6 普通名詞 1 * 0 * 0 "代表表記:文/ぶん 漢字読み:音 カテゴリ:抽象物"
です です だ 判定詞 4 * 0 判定詞 25 デス列基本形 27 NIL
. . . 特殊 1 句点 1 * 0 * 0 NIL
EOS
```

このように，文の構成要素（語）に切り離されて 1 行毎に出力される．各行の形式は空白区切りで，

表記　読み　原型　品詞　品詞細分類　活用型　活用型　意味情報

の順となっている．(詳しくは先のインターネットサイトを参照のこと)

JUMAN は標準入力からの読込みが**ファイルの終端 (EOF)** となった時点で終了する．OS のシェルから標準入力に対してファイルの終端を入力するには CTRL+Z と入力する．この操作で JUMAN が終了する．

1.5 KNP：日本語係り受け解析

KNP は，JUMAN によるの日本語文の形態素解析の結果を用いて，文の係り受け構造を解析するツールである．(開発元は JUMAN と同じである)
KNP はコマンドツールであり，標準入力から受け付けた JUMAN からの入力を解析して結果を標準出力に出力する．

次に示す jtext-s.txt のようなテキストファイルを JUMAN と KNP を用いて解析する例を示す．

サンプルテキスト：jtext-s.txt

```
1 この文のような複雑な係り受け構造を持つ文を解析することができると人工知能プログラムのためのヒューマンインター
      フェースを開発するのに役立ちます.
```

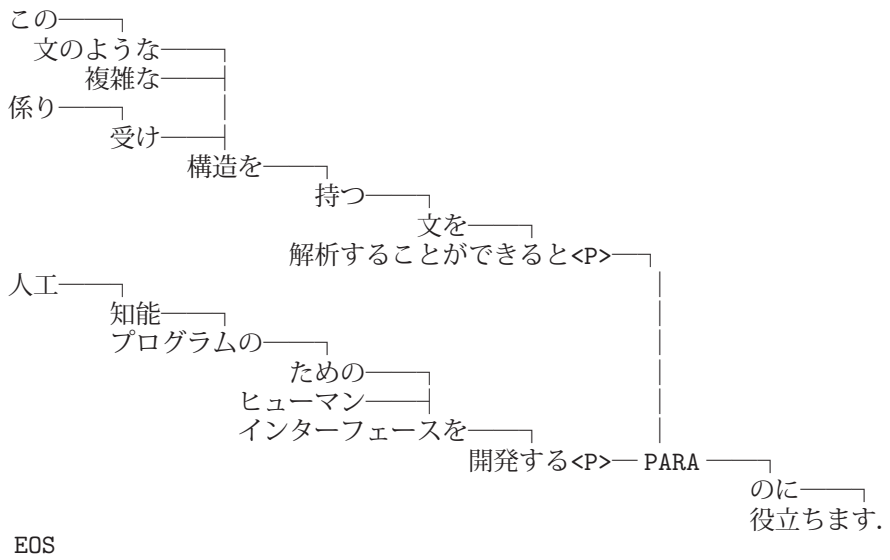
このテキストファイルを次のようなコマンドで処理をする．

[4]知能情報学専攻 知能メディア講座 言語メディア分野（工学部電気電子工学科）黒橋・河原研究室

Windows の場合： `type jtext-s.txt | juman | knp`

Mac/Linux の場合： `cat jtext-s.txt | juman | knp`

これにより，次のような結果（木構造）が標準出力に出力される．




```
EOS
```

KNP 実行時に ‘-tab’ オプションを与えると，解析結果がデータレコードとして出力される．（次の例参照）

例. Windows の場合の実行例

```
C:￥Users￥katsu>type jtext-s.txt | juman | knp -tab   Enter      ←コマンドの投入
# S-ID:1 KNP:4.11-CF1.1 DATE:2018/09/22 SCORE:-110.62014      ←解析結果の表示
* 1D <文頭><連体修飾><連体詞形態指示詞><係:連体><区切:0-4><正規化代表表記:この/ この>…
+ 1D <文頭><連体修飾><連体詞形態指示詞><係:連体><区切:0-4><正規化代表表記:この/ この>
この この この 指示詞 7 連体詞形態指示詞 2 * 0 * 0 "疑似代表表記 代表表記:この/ この" …
                :
          （途中省略）
                :
ます ます ます 接尾辞 14 動詞性接尾辞 7 動詞性接尾辞ます型 31 基本形 2 "代表表記:ます/…
. . . 特殊 1 句点 1 * 0 * 0 NIL <文末><英記号><記号><付属>
EOS
```

1.6 Open JTalk：テキスト読み上げ

Open JTalk は名古屋工業大学で開発された音声合成ソフトウェア[5] である．このソフトウェアは同大学が開発した hts_engine API を用いて構築されており，与えられた**辞書**と**音響モデル**に従って，日本語のテキストデータを読み上げた音声データ（WAV 形式データ）を生成する．このソフトウェアを計算機環境にインストールすると，OS のコマンド open_jtalk として実行できる．

Open JTalk のソースコード，日本語辞書，音響モデルなどがインターネットサイト

http://open-jtalk.sourceforge.net/

で公開されており入手できる．

■ open_jtalk **コマンドの基本的な使い方**

```
open_jtalk -x 辞書ディレクトリ  -m 音響モデル  -s サンプリング周波数  -r 読み上げ速度の比率
           -ow 出力ファイル     入力用テキストファイル
```

「辞書ディレクトリ」には辞書関連データを収めたディレクトリのパスを,「音響モデル」にはモデルデータのファイルのパスを指定する.「サンプリング周波数」には Hz 単位の数値を指定する.「読み上げ速度の比率」には，システムが標準としている読み上げ速度に対する比率を数値（小数点付き）で指定する.「入力用テキストファイル」は読み上げ対象のテキストデータのファイルのパスを指定するが，これは標準入力から与えることもできる．

処理が正常に終了すると「出力ファイル」に指定したパスに WAV 形式のデータがファイルとして生成される．出力ファイルのファイル名の末尾には拡張子 '.wav' を付けておくべきである．

1.6.1 処理の例

【Windows での例】

日本語文書が記述された入力ファイル jtest1.txt から，それを読み上げた結果の WAV 形式ファイル jtest1.wav を生成する方法の例を示す．Open JTalk のコマンドや辞書，音響モデルのファイルが次のようなディレクトリ階層に配置されているとする．

```
C:\OpJTalk
   |
   +- dic_utf_8              ←辞書関連のディレクトリ
   |   +- char.bin
   |   +- left-id.def
   |   +- matrix.bin
   |   +- pos-id.def
   |   +- rewrite.def
   |   +- right-id.def
   |   +- sys.dic
   |   +- unk.dic
   +- mei_angry.htsvoice       ←音響モデルファイル
   +- mei_bashful.htsvoice     ←音響モデルファイル
   +- mei_happy.htsvoice       ←音響モデルファイル
   +- mei_normal.htsvoice      ←音響モデルファイル
   +- mei_sad.htsvoice         ←音響モデルファイル
   +- open_jtalk.exe           ← Open JTalk 本体
```

この環境で音響モデル mei_normal.htsvoice に従って OpenJTalk コマンドを実行するには次のようなコマンドを投入する．

```
C:\OpJTalk\open_jtalk -x C:\OpJTalk\dic_utf_8 -m C:\OpJTalk\mei_normal.htsvoice
                                              -s 44100 -r 1.0 -ow jtest1.wav jtest1.txt
```

[5] http://software.web.nitech.ac.jp/「研究室のソフトウェア」（名古屋工業大学 産学官連携センター）

1.7 Julius：音声認識

Julius は音声入力デバイスや音声データファイルから入力された音声を認識するためのソフトウェアである．Julius は国内の大学[6] や IPA[7] によって開発，保守されており，関連情報（ソフトウェア本体，ドキュメント等）は OSDN サイトの Julius ページ http://julius.osdn.jp/ から入手できる．

Julius の機能は，各種プログラミング言語用の API という形（ライブラリという形）で利用することができる．また Julius は OS のコマンドとして実行することもでき，標準入出力やファイル，あるいは TCP/IP の通信を介して他のソフトウェアと連携させることができる．本書ではコマンドの基本的な使用方法を紹介し，音声入力デバイス（マイクからの入力）と音声データファイル（WAV 形式ファイル）から入力された音声を認識してテキストデータとして出力する過程を例示する．

■ 入手するパッケージ

先に挙げたインターネットサイトから**ディクテーションキット**をダウンロード[8] して展開する．（図 1）

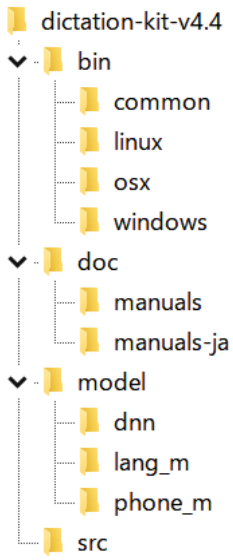


図 1: ディクテーションキットの中身（Ver.4.4）

このパッケージには Julius 本体をはじめ，音声認識に必要となるデータ（音響モデル，単語辞書，言語モデル）が含まれている．展開して出来上がったディレクトリ（図 1 の例では「dictation-kit-v4.4」フォルダ）はディスクの適切なディレクトリに配置しておく．

図 1 の例では「bin」フォルダ配下に Julius 関連の実行可能なプログラムファイル（各種 OS 用）が収められているので，使用する OS に合わせてコマンドサーチパスの設定をしておくのが良い．

以下，Ver.4.4 のディクテーションキットを前提として解説する．

■ Julius を利用するための基礎知識

Julius の動作を正しく理解して有効に利用するためには音声認識に関する基礎知識を有することが望ましいが，本書では Julius を最低限の範囲で動作させるために必要となる最小限の解説にとどめる．

Julius では HMM（隠れマルコフモデル）で状態遷移をモデル化する手法に基いて音声認識を行う．HMM としては GMM-HMM（Gaussian Mixture Model）と DNN-HMM（深層ニューラルネットワークモデル）の 2 種類のモデルから選択して利用することができ，Julius の起動時に必要となるモデルデータを指定する．これら 2 種類のモデルの違いとしては概ね次の通りである．

[6] 京都大学 河原研究室，奈良先端科学技術大学院大学 鹿野研究室，名古屋工業大学 Julius 開発チーム
[7] 独立行政法人情報処理推進機構（経済産業省所管）
[8] Ver.4.4 の場合，ZIP アーカイブ形式で 407MB，展開すると 830MB のサイズになることを予め留意すること．

1) GMM

認識精度はやや低いが実行時の CPU への負荷が低い.

2) DNN

CPU への負荷は大きいが，認識精度が高い.

どちらのモデルを使用するかは，Julius を利用する計算機環境と処理の目的から判断する.

1.7.1 動作テスト

ディクテーションキットに含まれるファイルを使用して Julius の動作をテストする方法を紹介する．パッケージのフォルダにはテスト実行用のバッチファイル[9] などが含まれており，それらを利用して音声認識のテスト（デモンストレーション）を行うことができる.

【Windows で動作テストを実行する例】

Windows 用の起動バッチファイルとして下記のようなファイルがパッケージに含まれている.

1) `run-win-dnn.bat` : DNN による音声認識
2) `run-win-dnncli.bat` : DNN による音声認識（クライアントモードで実行）
3) `run-win-gmm.bat` : GMM による音声認識

この内，1) のバッチファイルを起動する例を示す.

音声認識処理に先立って，マイクを計算機環境に接続して音声入力が機能することを確認する．この後，コマンドプロンプトウィンドウを開いてカレントディレクトリをディクテーションキットのディレクトリに移動してバッチファイルを起動する.

例.

`run-win-dnn` Enter ←バッチの起動

起動直後に Julius からのメッセージが多数表示された後，次のような表示となる.

```
Notice for feature extraction (01),
        ************************************************************
        * Cepstral mean and variance norm. for real-time decoding:  *
        * initial mean loaded from file, updating per utterance.    *
        * static variance loaded from file, apply it constantly.    *
        * NOTICE: The first input may not be recognized, since      *
        *         cepstral mean is unstable on startup.             *
        ************************************************************

Stat: adin_portaudio: audio cycle buffer length = 256000 bytes
Stat: adin_portaudio: sound capture devices:
  1 [MME: Microsoft サウンド マッパー - Input]
  2 [MME: マイク (2- USB PnP Audio Device]
  3 [MME: マイク (Realtek High Definition]
  6 [Windows DirectSound: プライマリ サウンド キャプチャ ドライバー]
  7 [Windows DirectSound: マイク (2- USB PnP Audio Device)]
  8 [Windows DirectSound: マイク (Realtek High Definition Audio)]
Stat: adin_portaudio: APIs: DirectSound MME
Stat: adin_portaudio: -- DirectSound selected
Stat: adin_portaudio: [Windows DirectSound: プライマリ サウンド キャプチャ ドラ
イバー]
Stat: adin_portaudio: (you can specify device by "PORTAUDIO_DEV_NUM=number"
Stat: adin_portaudio: try to set default low latency from portaudio: 0 msec
Stat: adin_portaudio: latency was set to 0.000000 msec
pass1_best: <input rejected by short input>
<<< please speak >>>
```

図 2: run-win-dnn 起動直後の表示

ウィンドウの最下行に “`<<< please speak >>>`” と表示されたら音声入力が受け付けられるので，マイクに向かって短く（10 秒程度）発話してみる．ここで,

「音声認識のテストをしています」

と発話した場合の認識例（コマンドプロンプトウィンドウの表示内容）を示す.

[9] 処理を実行するために必要な各種設定を記述したファイル.

```
pass1_best:  ＜input rejected by short input＞
pass1_best:  温泉 認識 の 、 テスト し て い ます
sentence1:  音声 認識 の 、 テスト を し て い ます 。
<<< please speak >>>
```

音声認識の結果が表示されている．

バッチファイルの内容について

ここで使用したバッチファイル run-win-dnn.bat の内容は次のようなものである．

```
.￥bin￥windows￥julius.exe -C main.jconf -C am-dnn.jconf -demo
                           -charconv utf-8 sjis -dnnconf julius.dnnconf
```

Julius 本体を起動するには多くのオプションを指定する必要があり，このバッチファイルではそのための処理が記述されている．バッチファイルの中で，次に示す記述は各種のオプション指定を記述するファイルであり，更に詳細なオプションが記述されている．

```
main.jconf
am-dnn.jconf
julius.dnnconf
```

参考） これらのファイルの内容を調べることで，オプション指定の方法に関するヒントが得られる．オプションの指定に関する詳細については公式のドキュメントを参照のこと．

1.7.2 WAV 形式音声ファイルの認識

WAV 形式の音声データファイルを入力として音声認識を行う例を示す．次に示すような筆者が読み上げた音声のデータを認識する過程を取り上げて説明する．

使用する音声データ

```
snd01.wav  :  「音声認識のテストをしています」
snd02.wav  :  「結果はどうでしょうか」
```

処理はディクテーションキットのディレクトリで行うことを前提とし，コマンドプロンプトウィンドウでバッチ処理を行う際のカレントディレクトリもディクテーションキットのディレクトリであることを前提とする．また OS は Windows を前提とする．

手順１：バッチファイルの作成

先の動作テストで使用したバッチファイルを参考にして次のような内容のバッチファイル to-file.bat を作成する．

```
.￥bin￥windows￥julius.exe -C myconf.jconf -C am-dnn_2.jconf
                    -dnnconf julius.dnnconf -charconv utf-8 sjis
```

手順２：設定ファイルの作成

バッチファイル to-file.bat にある myconf.jconf はディクテーションキットに含まれる設定ファイル main.jconf を参考にして次のように作成する．

設定ファイル：myconf.jconf

```
1  ## ログファイルの設定など
2  -logfile log.txt
3  -quiet
4
5  ## 入力ファイルの設定など
6  -input rawfile
7  -filelist flist.txt
8  -spsegment
9
10 ## 認識結果のファイル出力
11 -outfile
12
13 ## 単語2-gram,3-gramファイル
14 -d model/lang_m/bccwj.60k.bingram
15
16 ## 単語辞書ファイル
17 -v model/lang_m/bccwj.60k.htkdic
18
19 #### 探索パラメータ ####
20 -b 1500
21 -b2 100
22 -s 500
23 -m 10000
24 -n 30
25 -output 1        # 第2パスで見つかった文のうち出力する数 （文数）
26 -zmeanframe      # フレーム単位のDC成分除去を行う （HTKと同処理）
27
28 #### 短時間瞬時入力の棄却 ####
29 -rejectshort 800    # 指定ミリ秒以下の長さの入力を棄却する
```

ファイルからの音声データ入力に関する設定で重要な記述が 6, 7, 11 行目である．-input rawfile というオプションを与えることで WAV 形式ファイルからの入力となる．また -filelist **入力ファイル名リストのファイル名**　という記述によって，入力用の WAV 形式ファイルのファイル名 Julius に与えることができる．

手順３：入力ファイル名リストの作成

今回は「入力ファイル名リスト」のファイル名は `flist.txt`（設定ファイル 7 行目）で，内容は次のようなものである．

```
snd01.wav
snd02.wav
```

設定ファイルの 11 行目にある -outfile という記述は，認識結果をテキストファイルとして出力することを指定する．この際の出力ファイルの名前は，入力ファイルと同じファイル名で，拡張子が ‘.out’ となったものである．従って今回の場合は出力ファイルは，

```
snd01.out
snd02.out
```

の 2 つとなる．

手順４：バッチファイルの実行

実行用のバッチファイル `to-file.bat` をコマンドプロンプトのウィンドウで実行する．

以上の処理が正常に終了すると，`snd01.out`，`snd02.out` に認識結果が出力される．実行結果の例を次に示す．

認識結果 1：snd01.out

```
1 sentence1: 音声 認識 の 、 テスト を し て い ます 。
2 wseq1: <s> 音声+名詞 認識+名詞 の+助詞 、+補助記号 テスト+名詞 を+助詞 し+動詞 て+
      助詞 い+動詞 ます+助動詞 </s>
3 phseq1: sp_S | o_B N_I s_I e:_E | n_B i_I N_I sh_I i_I k_I i_E | n_B o_E | sp_S |
      t_B e_I s_I u_I t_I o_E | o_S | sh_B i_E | t_B e_E | i_S | m_B a_I s_I u_E |
      sp_S
4 cmscore1: 0.902 0.844 0.894 0.312 0.790 0.992 0.215 0.988 0.995 0.543 0.769 1.000
5 score1: 421.860229 (AM: 716.412537  LM: -294.552307)
```

認識結果 2：snd02.out

```
1 sentence1: 結果 は 、 どう でしょう か 。
2 wseq1: <s> 結果+名詞 は+助詞 、+補助記号 どう+副詞 でしょう+助動詞 か+助詞 </s>
3 phseq1: sp_S | k_B e_I q_I k_I a_E | w_B a_E | sp_S | d_B o:_E | d_B e_I sh_I o:_E
      | k_B a_E | sp_S
4 cmscore1: 0.591 0.972 0.925 0.425 0.848 0.980 0.733 1.000
5 score1: 509.048126 (AM: 665.216736  LM: -156.168594)
```

各出力ファイルにおいて，行頭が ‘sentence1:’ となる行が認識結果で，行頭が ‘wseq1:’ となる行が解析結果である．

■ その他

本書で紹介した内容は Julius の機能の部分的な紹介に過ぎない．利用者が作成したプログラムと Julius のプログラムを連携させるための方法の詳細に関しては，公式ドキュメントをはじめとする各種ドキュメントを参照のこと．

1.8 OpenSSL：暗号化，復号，電子署名

OpenSSL は通信の暗号化やメッセージの暗号化，ハッシュ生成のための関数ライブラリある．またそれらを用いた機能を実行するためのコマンド openssl も提供している．OpenSSL に関する情報はインターネットサイト

https://www.openssl.org/

で公開されている．OpenSSL がサポートする暗号化通信プロトコルは，

SSL 2.0, 3.0　　TLS 1.0, 1.1, 1.2　　DTLS 1.0, 1.2

である．また，文書を暗号化する際のアルゴリズムは，

Blowfish, Camellia, DES, RC2, RC4, RC5, SEED, IDEA, AES

が，ハッシュアルゴリズムとしては，

MD5, MD2, SHA-1, SHA-2, MDC-2

が利用できる．OpenSSL は公開鍵暗号方式の暗号化機能も提供しており，アルゴリズムとして，

RSA 暗号, DSA, Diffie-Hellman 鍵共有

が利用できる．

1.8.1 公開鍵暗号方式による暗号化の方法

公開鍵暗号方式は，メッセージ（ファイル）の暗号化と復号にそれぞれ異なる鍵を使用する暗号化方式であり，この形態から**非対称鍵暗号方式**とも呼ばれる．公開鍵暗号方式には**鍵の所有者**の存在が前提とされている．すなわち，暗号化処理を利用する者は，暗号化に使用する鍵である**公開鍵**と，復号のために使用する鍵である**秘密鍵**を作成して用意する．そして公開鍵を第三者に渡しておき，その第三者から鍵の所有者にメッセージやデータを送信する際に公開鍵を用いて暗号化してもらう．これによって，第三者から鍵の所有者に向けてのデータ送信の秘匿化が実現する．暗号化されたメッセージやデータを受け取った鍵の所有者は，秘密鍵を用いてそれを復号する．**秘密鍵**はその名が示すとおり，鍵の所有者のみが保持するものであり，決して第三者に開示してはならない．

OpenSSL には公開鍵，秘密鍵を生成するための機能と，それらを用いて暗号化と復号を行う機能が提供されており，openssl コマンドによりそれらの機能が利用できる．ここでは，RSA 暗号アルゴリズムを用いた公開鍵暗号方式の使用方法について説明する．

1.8.1.1 公開鍵と秘密鍵の生成

openssl コマンドに引数 genrsa を与えることで秘密鍵を生成することができる．

例. 秘密鍵の生成

```
openssl genrsa > secret.key
```

この結果，秘密鍵がファイル secret.key として作成される．鍵を生成する際に**鍵の長さ**（ビット長）を指定することができる．鍵は長い程安全である．

例. ビット長を指定した秘密鍵の生成

```
openssl genrsa 2048 > secret.key
```

この結果，2048 ビットの秘密鍵がファイル secret.key として作成される．

第三者に開示する**公開鍵**は秘密鍵を元にして作成する．

例. 秘密鍵 secret.key から公開鍵 public.key を作成する

```
openssl rsa -pubout < secret.key > public.key
```

このように openssl コマンドの引数に rsa と，オプション -pubout を与えて実行する．

1.8.1.2　公開鍵による暗号化

先の例に引き続き，作成した公開鍵 public.key を使用してメッセージデータ original.dat を暗号化して暗号データ encrypted.dat を生成する例を示す．

例. 公開鍵を用いた暗号化

```
openssl rsautl -encrypt -pubin -inkey public.key < original.dat > encrypted.dat
```

このように openssl コマンドの引数に rsautl と，オプション -encrypt, -pubin, -inkey を与えて実行する．オプション -inkey の後ろには公開鍵のファイル名を指定する．作成された暗号データ encrypted.dat の内容は可読性の無いものになっている．

1.8.1.3　秘密鍵による復号

先の例に引き続き，作成した秘密鍵 secret.key を使用して暗号データ encrypted.dat を復号して decrypted.dat を生成する例を示す．

例. 秘密鍵を用いた復号

```
openssl rsautl -decrypt -inkey secret.key < encrypted.dat > decrypted.dat
```

このように openssl コマンドの引数に rsautl と，オプション -decrypt, -inkey を与えて実行する．オプション -inkey の後ろには秘密鍵のファイル名を指定する．作成されたデータ decrypted.dat の内容は暗号化前のデータと同じ内容となっている．

1.8.2　公開鍵暗号方式による電子署名の作成

公開鍵暗号方式による暗号化は，第三者から鍵の所有者に向けてのデータの送信を秘匿化することが目的である．これとは別に，鍵の所有者が第三者に向けてデータを配信する場合に，配信者が鍵の所有者であることを保証し，かつ，配信されたデータが改竄されていないこと（真正性）を保証するための方法として電子署名の利用がある．

電子署名は，鍵の所有者が**配信するデータ**と**自身の秘密鍵**から作成するものであり，当該データを配信する際に添付するものである．

電子署名の利用の手順は次の通りである．

1. 鍵の所有者が秘密鍵を使って，配信対象のデータのための電子署名を作成する．
original.dat の真正性を保証するために電子署名 sign.dat を作成するには，openssl コマンドを用いて次のように処理する．

```
openssl dgst -sha1 -sign secret.key < original.dat > sign.dat
```

これで，original.dat の真正性を保証する電子署名 sign.dat が作成される．この例では，secret.key は秘密鍵である．

2. データと電子署名の両方を第三者に配信する．

3. 電子署名を用いてデータの真正性を検査する．(受信した第三者の作業)
openssl コマンドを用いて次のように処理する．

```
openssl dgst -sha1 -verify public.key -signature sign.dat < original.dat
```

この例では public.key は公開鍵である．データの真正性が確認できた場合は，

```
Verified OK
```

と表示され，そうでない場合は，

```
Verification Failure
```

と表示される．

この例のように，電子署名の生成やデータの真正性の検査をする際には openssl コマンドの引数には dgst を与える．また電子署名生成に用いるハッシュアルゴリズムを指定することができ，-sha1 の他にも -md5，-sha256，-sha512 などが指定できる．

2　ライブラリ

2.1　GMP/MPFR：任意多倍長演算

C や Java で標準的に扱える数値の型はビット長が固定されており，整数として扱える数値の桁数や，浮動小数点数として扱える数値の有効桁数（精度）には限界がある．従って，それを超える桁数の数値を扱う [10] には，特別なデータ構造を定義して，それを扱うための関数やメソッドを実装する必要がある．ここで紹介する GMP（GNU Multi-Precision Library）と MPFR は，任意精度の数値演算を実行するためのライブラリであり，C 言語と C++言語で利用できるが，それ以外の各種言語で利用するためのラッパーも用意されている．

GMP は任意精度の整数，有理数，浮動小数点数を扱うためのライブラリであり，インターネットサイト

https://gmplib.org/

でソースコードやドキュメントが公開されている．GMP 自体には各種の数学関数の値を求める機能は提供されておらず，GMP を用いて開発されたライブラリ MPFR として各種の数学関数が提供されている．MPFR に関しては，インターネットサイト http://www.mpfr.org/ でソースコードやドキュメントが公開されている．

GMP と MPFR は別々のライブラリであり，C/C++言語で利用する際にもインクルードするヘッダファイルやリンクするライブラリは別のものとして扱われる．本書ではこれらライブラリの使用方法を導入的に紹介する．詳しくは各ライブラリのドキュメントを参照のこと．

2.1.1　GMP

GMP には任意精度の整数，有理数，浮動小数点数の値を保持するオブジェクトのデータ型としてそれぞれ mpz_t, mpq_t，mpf_t がある．[11]

GMP を用いて行うことは概ね次の 3 種類の処理である．

1. オブジェクトの宣言と初期化
   mpz_t，mpq_t，mpf_t といった GMP のオブジェクトは宣言した後，初期化する必要がある．
2. データの変換
   C の各種データ（long, double, char* など）と GMP のオブジェクトの値を相互に変換する機能．
3. 算術演算など
   加減乗除，開平をはじめとする基本的な演算処理．
4. オブジェクトの開放
   不要になったオブジェクトの記憶域を開放する．

オブジェクトの初期化と値の変換に関する処理について重要なものを表 4〜5 に挙げる．

表 4: オブジェクトの初期化

対象の型	書き方	説明
mpz_t	mpz_init(x) mpz_inits(x1, x2, ···,(mpz_t *)0)	mpz_t 型の x を初期化する． mpz_t 型の x1,x2,··· を初期化する． 初期値は 0 となる．
mpq_t	mpq_init(x) mpq_inits(x1, x2, ···,(mpq_t *)0)	mpq_t 型の x を初期化する． mpq_t 型の x1,x2,··· を初期化する． 初期値は 0/1 となる．
mpf_t	mpf_init2(x, p)	mpf_t 型の x を p ビット長の精度で初期化する． 初期値は 0 となる．(p の型は mp_bitcnt_t)

[10] 情報セキュリティ関連の処理（暗号化や復号など）や，数理科学系分野における力学系の高精度シミュレーションなどを行う際に必須の機能である．

[11] mpz_t は整数を表す $\mathbb{Z}$ を，mpq_t は有理数を表す $\mathbb{Q}$ を命名の由来とする．

表 5: 値の変換

変換前のオブジェクト	mpz_t 型の x へ	mpq_t 型の x へ	mpf_t 型の x へ
long n	mpz_set_si(x, n)	mpq_set_si(x, n, d) x ← n / (unsigned long)d	mpf_set_si(x, n)
double n	mpz_set_d(x, n)	mpq_set_d(x, n)	mpf_set_d(x, n)
mpz_t n	mpz_set(x, n)	mpq_set_z(x, n)	mpf_set_z(x, n)
mpq_t n	mpz_set_q(x, n)	mpq_set(x, n)	mpf_set_q(x, n)
mpf_t n	mpz_set_f(x, n)	mpq_set_f(x, n)	mpf_set(x, n)
(char *)n	mpz_set_str(x, n, Base)	mpq_set_str(x, n, Base)	mpf_set_str(x, n, Base)

※ Base には int 型の整数で基数を指定する.

表 6: GMP の型から文字列 (char *)s への変換

変換前のオブジェクト	書き方
mpz_t n	mpz_get_str(s, Base, n)
mpq_t n	mpq_get_str(s, Base, n)
mpf_t n	mpf_get_str(s, &E, Base, Digit, n)

〔表 5,6 の補足〕

Base は基数を指定する整数で，int 型で与える．E は mp_exp_t 型の変数で，処理の結果得られた文字列の小数点の位置を表す数値（整数）が取得される．引数には E のアドレスを渡す.

Digit は size_t 型の整数値で，文字列として取得する数値の桁数を指定する．Digit の値として 0 を指定すると，最も正確な値が取られる最大の桁数が自動的に設定される.

■ mpf_t オブジェクトの精度と初期化について

mpf_t オブジェクトを初期化するに当たって，精度を暗黙値として設定する方法がある．具体的には関数 mpf_set_default_prec を使って，

```
mpf_set_default_prec(精度)
```

とすることで初期化のための精度を事前に設定しておくことができる.（精度の型は mp_bitcnt_t）
この関数呼び出しの後，複数の mpf_t オブジェクト a, b, c, ··· に対して mpf_inits 関数を使って

```
mpf_inits( a, b, c, ···, (mpf_t *)0 )
```

として一括して初期化することができる.

2.1.1.1 GMP で利用できる関数（一部）

■ 加算

- **mpz_t の加算**

  mpz_add_ui(y, x, n) : (mpz_t)y ← (mpz_t)x + (unsigned long)n
  mpz_add(y, x, n) : (mpz_t)y ← (mpz_t)x + (mpz_t)n

- **mpq_t の加算**

  mpq_add(y, x, n) : (mpq_t)y ← (mpq_t)x + (mpq_t)n

- **mpf_t の加算**

  mpf_add_ui(y, x, n) : (mpf_t)y ← (mpf_t)x + (unsigned long)n
  mpf_add(y, x, n) : (mpf_t)y ← (mpf_t)x + (mpf_t)n

■ 減算

- **mpz_t の減算**

  mpz_sub_ui(y, x, n) : (mpz_t)y ← (mpz_t)x - (unsigned long)n
  mpz_ui_sub(y, x, n) : (mpz_t)y ← (unsigned long)x - (mpz_t)n
  mpz_sub(y, x, n) : (mpz_t)y ← (mpz_t)x - (mpz_t)n

- **mpq_t の減算**

  mpq_sub(y, x, n) : (mpq_t)y ← (mpq_t)x - (mpq_t)n

- **mpf_t の減算**

  mpf_sub_ui(y, x, n) : (mpf_t)y ← (mpf_t)x - (unsigned long)n
  mpf_ui_sub(y, x, n) : (mpf_t)y ← (unsigned long)x - (mpf_t)n
  mpf_sub(y, x, n) : (mpf_t)y ← (mpf_t)x - (mpf_t)n

■ 乗算

- **mpz_t の乗算**

  mpz_mul_si(y, x, n) : (mpz_t)y ← (mpz_t)x * (long)n
  mpz_mul(y, x, n) : (mpz_t)y ← (mpz_t)x * (mpz_t)n

- **mpq_t の乗算**

  mpq_mul(y, x, n) : (mpq_t)y ← (mpq_t)x * (mpq_t)n

- **mpf_t の乗算**

  mpf_mul_ui(y, x, n) : (mpf_t)y ← (mpf_t)x * (unsigned long)n
  mpf_mul(y, x, n) : (mpf_t)y ← (mpf_t)x * (mpf_t)n

■ 除算

- **mpq_t の除算**

  mpq_div(y, x, n) : (mpq_t)y ← (mpq_t)x / (mpq_t)n

- **mpf_t の除算**

  mpf_div_ui(y, x, n) : (mpf_t)y ← (mpf_t)x / (unsigned long)n
  mpf_ui_div(y, x, n) : (mpf_t)y ← (unsigned long)x / (mpf_t)n
  mpf_div(y, x, n) : (mpf_t)y ← (mpf_t)x / (mpf_t)n

■ 比較

表 7 に値の大きさを比較する関数の一部を挙げる．

表 7: 値の比較： x¿n →正の整数，x=n → 0，x¡n →負の整数

x の型	long n	double n	mp*_t n (n は x と同じ型)
mpz_t x	mpz_cmp_si(x, n)	mpz_cmp_d(x, n)	mpz_cmp(x, n)
mpq_t x	mpq_cmp_si(x, n, d) (n/d との比較※)		mpq_cmp(x, n)
mpf_t x	mpf_cmp_si(x, n)	mpf_cmp_d(x, n)	mpf_cmp(x, n)

※ d は unsigned long

■ その他

表 8 にその他の関数（一部）を挙げる．

表 8: その他の関数

対象の型	正負判定 正:1, 零:0, 負:-1	正負反転 $y = -x$	絶対値 $y = \|x\|$
mpz_t x, y	int mpz_sgn(x)	mpz_neg(y, x)	mpz_abs(y, x)
mpq_t x, y	int mpq_sgn(x)	mpq_neg(y, x)	mpq_abs(y, x)
mpf_t x, y	int mpf_sgn(x)	mpf_neg(y, x)	mpf_abs(y, x)

※ 正負判定は関数の戻り値を見る

表 9: オブジェクトの記憶域を開放する関数

関数	説明
mpz_clear(mpz_t n) mpz_clears(a, b, ..., (mpz_t *)0)	mpz_t n の記憶域を開放する. mpz_t 型の a, b, ... の記憶域を開放する.
mpq_clear(mpq_t n) mpq_clears(a, b, ..., (mpq_t *)0)	mpq_t n の記憶域を開放する. mpq_t 型の a, b, ... の記憶域を開放する.
mpf_clear(mpf_t n) mpf_clears(a, b, ..., (mpf_t *)0)	mpf_t n の記憶域を開放する. mpf_t 型の a, b, ... の記憶域を開放する.

■ オブジェクトの開放

不要になったオブジェクトは表 9 に挙げるような関数を使用して記憶域を開放する.

ここで紹介するもの以外の関数も多数あるため，公式のドキュメントを参照のこと.

2.1.1.2 サンプルプログラム

■ 任意桁数の整数の計算

100 の階乗（100!）を求めるプログラムを bn_gmp01.c に示す.

プログラム：bn_gmp01.c

```
 1 #include    <stdio.h>
 2 #include    <gmp.h>
 3
 4 int main()
 5 {
 6     int     i;
 7     char    s[1024];
 8     mpz_t   n;              /* 任意精度整数の宣言 */
 9
10     mpz_init( n );          /* 初期化 */
11     mpz_set_si( n, 1 );     /* 1をセット */
12     for ( i = 2; i < 100; i++ ) {
13         mpz_mul_si( n, n, i );      /* n = n * i の計算 */
14     }
15     mpz_get_str( s, 10, n );        /* 文字列に変換 */
16     printf("100!=%s\n",s);          /* 文字列の表示 */
17
18     mpz_clear( n );         /* 記憶域の開放 */
19 }
```

2.1.1.3 プログラムの実行に際して

GMP を利用するアプリケーションプログラムを作成するに当たって，ヘッダファイル gmp.h を読み込む必要がある．具体的にはソースプログラムの冒頭に，

```
#include <gmp.h>
```

と記述する必要がある．また，ソースプログラムを翻訳，結合する際には，GMP のライブラリの結合をコンパイラ

に指示しなければならないことが多い．例えば，GNU の C コンパイラ gcc を用いて先のプログラム bn_gmp01.c を翻訳，結合するには，

```
gcc bn_gmp01.c -o bn_gmp01 -lgmp
```

などとする．(’-lgmp’ オプションは GMP ライブラリの結合を意味する)

C++言語でも C 言語の場合と同様に GMP を利用することができるが，GNU の C++コンパイラ g++ を用いて翻訳，結合するには，’-lgmp’ オプションに代わって ’-lgmpxx’ オプションを指定する．

プログラム bn_gmp01.c を翻訳して実行すると次のように表示される．

```
100!=933262154439441526816992388562667004907159682643816214685929638952175999932299
156089414639761565182862536979208272237582511852109168640000000000000000000000
```

■ 有理数の計算と浮動小数点数の扱い

次のような有理数の総和を計算するプログラムを bn_gmp02.c に示す．

$$\sum_{k=1}^{99} \frac{k}{k+1}$$

プログラム：bn_gmp02.c

```
#include    <stdio.h>
#include    <gmp.h>

int main()
{
    int     i;
    char    s[1024];
    mpq_t   q, q2;      /* 任意精度有理数の宣言 */
    mpf_t   f;          /* 任意精度浮動小数点数の宣言 */
    mp_exp_t    e;      /* 指数データ（表示用） */

    mpq_inits( q, q2, (mpq_t *)0 ); /* 有理数の初期化 */
    for ( i = 1; i < 100; i++ ) {
        mpq_set_si( q2, i, i+1 ) ;  /* q2 = i/(i+1) の計算 */
        mpq_add( q, q, q2 );        /* q = q + q2 の計算 */
    }
    mpq_get_str( s, 10, q );            /* 文字列に変換 */
    printf("sum(rational)=%s\n\n",s);   /* 文字列の表示 */

    mpf_init2( f, 1024 );   /* 浮動小数点数fを精度1024ビットで初期化 */
    mpf_set_q( f, q );      /* 有理数qを浮動小数点数fに変換 */
    mpf_get_str( s, &e, 10, 300, f );   /* 文字列に変換 */
    printf("sum(fp)=%s[exp:%d]\n",s,e); /* 文字列の表示 */

    /* 記憶域の開放 */
    mpq_clears( q, q2, (mpq_t *)0 );
    mpf_clear( f );
}
```

このプログラムを翻訳して実行すると次のように表示される．

```
sum(rational)=26441486463932955749791371769814508277948 9/27888150091884990865813523
57412492142272

sum(fp)=94812622482360379739194882324341746842091027873291548346823466043412780442 4
67449503394312231076879586447048627099919040514235665097996140748715452520600393511
32228064356229896564858249837199638786618606365966389602829741849614390770239074224
14750975798421354587658616633908101293972409274649548741705[exp:2]
```

’sum(rational)=’ に続いて計算結果が有理数（分数）の形で表示されている．また’sum(rational)=’ に続いて浮動小数点数で表現した数値が表示されている．最後の ’[exp:2]’ は，小数点が先頭から 2 桁目の右の位置に存在し

ていることを意味している．

2.1.2 MPFR

MPFR は GMP を用いて構築されたライブラリであり，GMP が提供していない高度な数学関数が多数実装されている．MPFR の扱い方は基本的には GMP に倣ったものとなっており，ライブラリの利用は概ね次の 3 種類の処理に分類することができる．

1. オブジェクトの宣言と初期化
   MPFR のデータ型 mpfr_t のオブジェクトを宣言した後，それらを初期化する．
2. データの変換
   C の各種データ（long, double, char* など）や GMP の各種オブジェクトと値を相互に変換する．
3. 演算処理
   算術演算，数学関数などの演算．

■ mpfr_t オブジェクトの初期化

mpfr_t のオブジェクト x がある場合，これを 512 ビット長の精度を持つものとして初期化するには mpfr_init2 関数を用いて，

```
mpfr_init2( x, 512 )
```

とする．第 2 引数には整数で精度を与えるが，より正確には mpfr_prec_t 型の値として与える．

複数の mpfr_t のオブジェクト m1, m2,··· を一度に初期化するには mpfr_inits2 関数を用いて，

```
mpfr_inits2( 512, m1, m2,···, (mpfr_ptr)0 )
```

とする．この関数は最初の引数に精度を指定し，最後の引数に (mpfr_ptr)0 を与える．

■ 値の変換

mpfr_t のオブジェクトと，他の型のオブジェクトや変数との間で値を変換する関数の一部を表 10 に挙げる．

表 10: 各種の型の値 n と mpfr_t x の間で変換を行う関数（一部）

変換元	x ← n	n ← x
long n	mpfr_set_si(x, n, Rnd)	n = mpfr_get_si(x, Rnd)
double n	mpfr_set_d(x, n, Rnd)	n = mpfr_get_d(x, Rnd)
long double n	mpfr_set_ld(x, n, Rnd)	n = mpfr_get_ld(x, Rnd)
mpz_t n	mpfr_set_z(x, n, Rnd)	mpfr_get_z(n, x, Rnd)
mpq_t n	mpfr_set_q(x, n, Rnd)	
mpf_t n	mpfr_set_f(x, n, Rnd)	mpfr_get_f(n, x, Rnd)
char *n	mpfr_set_str(x, n, Base, Rnd)	mpfr_get_str(n, &E, Base, Digit, x, Rnd)

〔**表 10 の補足**〕

Base は基数を指定する整数で，int 型で与える．E は mpfr_exp_t 型の変数で，処理の結果得られた文字列の小数点の位置を表す数値（整数）が取得される．引数には E のアドレスを渡す．Digit は size_t 型の数値で，文字列として取得する数値の桁数を指定する．Digit に 0 を与えると，信頼できる範囲の最大限の桁数で文字列データが得られる．

Rnd には mpfr_rnd_t 型の値で**丸め**（Rounding Mode）を指定する．具体的には表 11 のようなものが定義されており，それを指定する．

表 11: 丸め（Rounding Mode）

Mode	説明
MPFR_RNDN	最近接丸め．（**推奨**）
MPFR_RNDA	最近接丸め．0 から遠い方向に丸める．
MPFR_RNDZ	0 に近い方向に丸める．（切り捨て：truncation）
MPFR_RNDU	$+\infty$ の方向に丸める．（切り上げ：rounding up, ceiling）
MPFR_RNDD	$-\infty$ の方向に丸める．（切り下げ：rounding down, floor）

（IEEE-754 2008 準拠）

表 12: オブジェクトの記憶域を開放する関数（MPFR）

関数	説明
mpfr_clear(mpfr_t n)	mpfr_t n の記憶域を開放する．
mpfr_clears(a, b, ..., (mpfr_ptr *)0)	mpfr_t 型の a, b, ... の記憶域を開放する．

■ オブジェクトの開放

不要になったオブジェクトは表 12 に挙げるような関数を使用して記憶域を開放する．

ここで紹介するもの以外の関数も多数あるため，公式のドキュメントを参照のこと．

2.1.2.1 サンプルプログラム

π から $\frac{\pi}{6}$ を求め，更に $\sin\left(\frac{\pi}{6}\right)$ と $\cos\left(\frac{\pi}{6}\right)$ を求め，それを用いて $\sqrt{\sin^2\left(\frac{\pi}{6}\right)+\cos^2\left(\frac{\pi}{6}\right)}$ を求める

プログラム bn_mpfr01.c を示す．

プログラム：bn_mpfr01.c

```
 1 #include    <stdio.h>
 2 #include    <mpfr.h>
 3
 4 int main()
 5 {
 6     char    st[2048];
 7     mpfr_t  s, c, p, p6, s2, c2, l, norm;
 8     mpfr_exp_t  e;
 9
10     mpfr_init2( p, 512 );                   /* 初期化(1): 1つだけ初期化する例 */
11     mpfr_inits2( 512, p6, s, c, s2, c2,     /* 初期化(2): */
12                  l, norm, (mpfr_ptr)0 );    /*       複数まとめて初期化する例 */
13
14     mpfr_const_pi( p, MPFR_RNDN );          /* π（円周率） */
15     mpfr_div_si( p6, p, 6, MPFR_RNDN );     /* π/6 */
16
17     mpfr_get_str( st, &e, 10, (size_t)0, p, MPFR_RNDN );
18     printf("Pi=%s(point:%d)\n\n",st,e);
19
20     mpfr_sin( s, p6, MPFR_RNDN );           /* sin(π/6) */
21     mpfr_get_str( st, &e, 10, (size_t)0, s, MPFR_RNDN );
22     printf("sin(Pi/6)=%s(point:%d)\n\n",st,e);
23
24     mpfr_cos( c, p6, MPFR_RNDN );           /* cos(π/6) */
25     mpfr_get_str( st, &e, 10, (size_t)0, c, MPFR_RNDN );
26     printf("cos(Pi/6)=%s(point:%d)\n\n",st,e);
27
28     mpfr_sqr( s2, s, MPFR_RNDN );           /* s2 = sin(π/6)**2 */
29     mpfr_sqr( c2, c, MPFR_RNDN );           /* c2 = cos(π/6)**2 */
30     mpfr_add( l, s2, c2, MPFR_RNDN );       /* l = s2 + c2 */
```

```
    mpfr_sqrt( norm, l, MPFR_RNDN );         /* norm = sqrt( l ) */
    mpfr_get_str( st, &e, 10, (size_t)0, norm, MPFR_RNDN );
    printf("sqrt( sin(Pi/6)**2 + cos(Pi/6)**2 )=%s(point:%d)\n",st,e);

    /* 記憶域の開放 */
    mpfr_clears( s, c, p, p6, s2, c2, l, norm, (mpfr_ptr)0 );
}
```

2.1.2.2 プログラムの実行に際して

MPFR を利用するアプリケーションプログラムを作成するに当たって，ヘッダファイル mpfr.h を読み込む必要がある．具体的にはソースプログラムの冒頭に，

```
#include <mpfr.h>
```

と記述する必要がある．また，ソースプログラムを翻訳，結合する際には，MPFR のライブラリの結合をコンパイラに指示しなければならないことが多い．例えば，GNU の C コンパイラ gcc を用いて先のプログラム bn_mpfr01.c を翻訳，結合するには，

```
gcc bn_mpfr01.c -o bn_mpfr01 -lmpfr -lgmp
```

などとする．'-lmpfr' オプションは MPFR ライブラリの結合を意味する．MPFR は GMP を用いて構築されているので，'-lgmp' オプションも必要となる．

C++言語でも C 言語の場合と同様に MPFR を利用することができる．

プログラム bn_mpfr01.c を翻訳して実行すると次のように表示される．

```
Pi=314159265358979323846264338327950288419716939937510582097494459230781640628620899862
803482534211706798214808651328230664709384460955058223172535940812848115(point:1)

sin(Pi/6)=50000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000(point:0)

cos(Pi/6)=86602540378443864676372317075293618347140262690519031402790348972596650845440
0018540573093378624287837813070707703351514984972547499476239405827756047186834(point:0)

sqrt( sin(Pi/6)**2 + cos(Pi/6)**2 )=100000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000(point:1)
```

2.1.3 MPFR で利用できる数学関数（一部）

MPFR で利用できる数学関数の一部を表 13 に挙げる．詳しくは公式ドキュメントを参照のこと．

表 13: MPFR の数学関数（一部）

関数	書き方	関数	書き方
$y = \sqrt{x}$	mpfr_sqrt(mpfr_t y, mpfr_t x, Rnd)	$y = x^n$	mpfr_sqrt(mpfr_t y, mpfr_t x, mpfr_t n, Rnd)
$y = \log_e(x)$	mpfr_log(mpfr_t y, mpfr_t x, Rnd)	$y = \log_2(x)$	mpfr_log2(mpfr_t y, mpfr_t x, Rnd)
$y = \log_{10}(x)$	mpfr_log10(mpfr_t y, mpfr_t x, Rnd)	$y = e^x$	mpfr_exp(mpfr_t y, mpfr_t x, Rnd)
$y = 2^x$	mpfr_exp2(mpfr_t y, mpfr_t x, Rnd)	$y = 10^x$	mpfr_exp10(mpfr_t y, mpfr_t x, Rnd)
$y = \sin(x)$	mpfr_sin(mpfr_t y, mpfr_t x, Rnd)	$y = \cos(x)$	mpfr_cos(mpfr_t y, mpfr_t x, Rnd)
$y = \tan(x)$	mpfr_tan(mpfr_t y, mpfr_t x, Rnd)	$y = \sec(x)$	mpfr_sec(mpfr_t y, mpfr_t x, Rnd)
$y = \csc(x)$	mpfr_csc(mpfr_t y, mpfr_t x, Rnd)	$y = \cot(x)$	mpfr_cot(mpfr_t y, mpfr_t x, Rnd)
$y = \sin^{-1}(x)$	mpfr_asin(mpfr_t y, mpfr_t x, Rnd)	$y = \cos^{-1}(x)$	mpfr_acos(mpfr_t y, mpfr_t x, Rnd)
$y = \tan^{-1}(x)$	mpfr_atan(mpfr_t y, mpfr_t x, Rnd)		
$y = \sinh(x)$	mpfr_sinh(mpfr_t y, mpfr_t x, Rnd)	$y = \cosh(x)$	mpfr_cosh(mpfr_t y, mpfr_t x, Rnd)
$y = \tanh(x)$	mpfr_tanh(mpfr_t y, mpfr_t x, Rnd)	$y = sech(x)$	mpfr_sech(mpfr_t y, mpfr_t x, Rnd)
$y = csch(x)$	mpfr_csch(mpfr_t y, mpfr_t x, Rnd)	$y = \coth(x)$	mpfr_coth(mpfr_t y, mpfr_t x, Rnd)
$y = \sinh^{-1}(x)$	mpfr_asinh(mpfr_t y, mpfr_t x, Rnd)	$y = \cosh^{-1}(x)$	mpfr_acosh(mpfr_t y, mpfr_t x, Rnd)
$y = \tanh^{-1}(x)$	mpfr_atanh(mpfr_t y, mpfr_t x, Rnd)		
$y = \Gamma(x)$	mpfr_gamma(mpfr_t y, mpfr_t x, Rnd)	$y = \zeta(x)$	mpfr_zeta(mpfr_t y, mpfr_t x, Rnd)
$y = \pi$	mpfr_const_pi(mpfr_t y, Rnd)		

3 処理環境

3.1 Microsoft Visual Studio / Visual C++

Microsoft Visual Studio（以後 VS と略す）は Microsoft 社が開発している統合開発環境（IDE）である．VS 下では C/C++/C#，Visual Basic，Python をはじめとする各種のプログラミング言語でソフトウェア製品を開発できる．これら言語の内，C++ としては，同社が開発する Visual C++（以後 VC と略す）を使うことができる．VC は最適化性能の高い C 言語処理系であり，Windows 環境下の標準的な C/C++コンパイラである．

VS / VC は Microsoft 社のインターネットサイトから入手することができる．具体的には，Visual Studio のインストールパッケージを入手して起動し，インストーラの指示に従って作業すると VS と VC の両方が計算機環境にインストールされる．本書では Visual Studio Community 2017 の版に基づいて使用方法などを解説する．

3.1.1 Visual C++

C 言語処理系は基本的に**コンパイラ**，**ライブラリ**，**インクルードファイル**といったファイル群から成る．コンパイラは翻訳処理の際にインクルードファイルを読み込み，結合処理の際にライブラリ内の関数などを読み込む.（**動的リンクライブラリ**／DLL は実行時に結合される） 従って，ライブラリやインクルードファイルのディレクトリを環境変数に設定しておき，翻訳と結合の際にコンパイラが参照できるようにしておく必要があるが，それら設定作業はインストール時に行われる．

3.1.1.1 コマンドツールとしてのコンパイラ

VS / VC をインストールすると**開発者コマンドプロンプト**が使える．これは Windows 標準のコマンドプロンプトに VC 用の環境設定を施したものである．例えば，Visual Studio 2017 で開発者コマンドプロンプトを起動するバッチファイルは下記のパスである．

例. VS2017 での開発者コマンドプロンプト起動バッチファイル

```
C:￥Program Files (x86)￥Microsoft Visual Studio￥2017￥
                    Community￥Common7￥Tools￥VsDevCmd.bat
```

このバッチファイルを用いてコマンドプロンプトを起動すると，コンパイラ利用のための環境設定が施される．

開発者コマンドプロンプトを起動した例を次に示す．

例. 開発者コマンドプロンプトの起動

```
**********************************************************************
** Visual Studio 2017 Developer Command Prompt v15.7.1
** Copyright (c) 2017 Microsoft Corporation
**********************************************************************

C:￥Program Files (x86)￥Microsoft Visual Studio￥2017￥Community>
```

この後，コンパイラのコマンド cl.exe を使うことができる．

■ 翻訳と実行の例

次のような C 言語のプログラム test00.c を翻訳して実行する例を示す．

プログラム：test00.c

```
1  #include    <stdio.h>
2
3  int main()
4  {
5      int a = 1, b = 2, c;
6      c = a + b;
7      printf("%d + %d = %d\n",a,b,c);
8  }
```

開発者コマンドプロンプトで次のようにして翻訳処理を行う.

例. 翻訳処理

```
C:￥Users￥katsu> cl test00.c  Enter     ←翻訳と結合の処理
Microsoft(R) C/C++ Optimizing Compiler Version 19.14.26433 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

test00.c

Microsoft (R) Incremental Linker Version 14.14.26433.0
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:test00.exe
test00.obj
```

これで，オブジェクトファイル（結合前のファイル）test00.obj と実行形式ファイル test00.exe が生成された.
test00.exe を実行する例を次に示す.

例. 実行

```
C:￥Users￥katsu> test00  Enter     ←実行
1 + 2 = 3     ←結果表示
```

■ cl.exe の使用方法

cl.exe はオプション ‘/?’ を付けて起動すると各種オプションの使用方法（ヘルプ）を表示する.

例. ヘルプの表示

```
C:￥Users￥katsu>cl /?  Enter     ←ヘルプの表示
Copyright (C) Microsoft Corporation.  All rights reserved.

                    C/C++ COMPILER OPTIONS

                          -最適化-

/O1 最大限の最適化（スペースを優先）     /O2 最大限の最適化（速度を優先）
/Ob<n> インライン展開（既定値 n=0）     /Od 最適化を無効にする（既定）
/Og グローバルな最適化を有効にする       /Oi[-] 組み込み関数を有効にする
                          :
                    （以下省略）
                          :
```

3.1.2 Visual Studio

Visual Studio は統合開発環境（IDE）であり，構築対象のソフトウェアのためのファイル群を**プロジェクト**という単位で管理する．Visual Studio 2017 を起動した例を図 3 に示す．

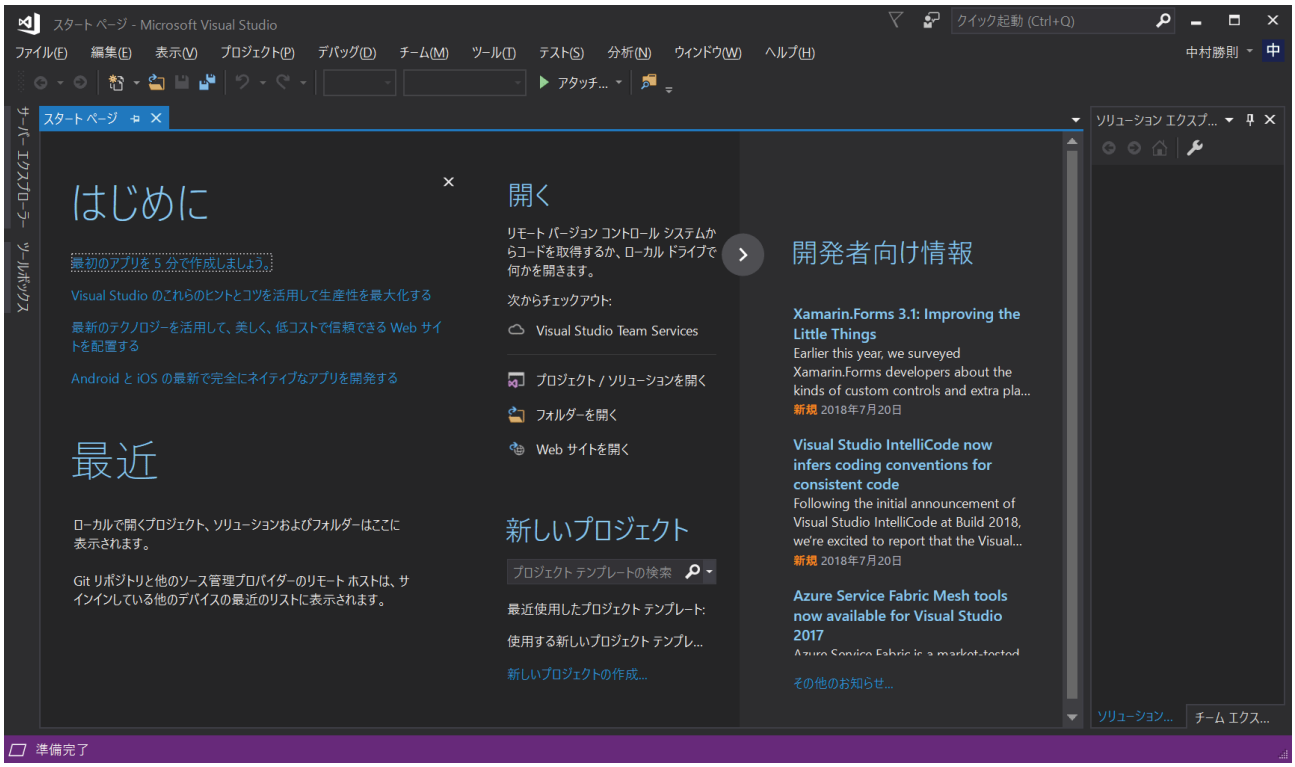


図 3: Visual Studio のウィンドウ

3.2 MinGW / MSYS2

MinGW（**Min**imalist **G**NU for **W**indows）は GNU の GCC [12] をはじめとするソフトウェアツール群を Windows 環境に移植したものである．また，MSYS（**M**inimal **SYS**tem）は UNIX の基本的なコマンドツール群を Windows 環境に移植したものである．Windows 環境で UNIX（POSIX）に準拠した開発ツール群を利用する場合，MinGW と MSYS の両方をインストールすると良い．

MinGW，MSYS は共に Cygwin [13] から派生したツールセットであり，公式サイトは下記の通りである．

ツールセット	公式サイトの URL
MinGW	http://www.mingw.org/
MSYS	http://www.msys2.org/

3.2.1 導入とパッケージ管理の方法

MSYS にはパッケージ管理のための機能が備わっており，それによって MSYS 導入後に MinGW を導入することができる．MSYS の公式インターネットサイトからインストーラ（図 4）を入手する．

図 4: MSYS のインストーラ

これを起動して導入作業を行う．インストーラの指示に従って作業を進めるが，MSYS のインストール先のディレクトリを選択できる．標準的なインストール作業ではインストール先はデフォルトで「C:￥msys64」となる．

インストール作業が終了すると MSYS のシェルである bash が利用できる．シェルを起動するためのアイコンが Windows のスタートメニューに登録されているのでそれによってシェルを起動する．パッケージ管理などは MSYS 標準の bash で，MinGW のツール群を使用する場合は MinGW32（もしくは 64）の bash を使用する．

3.2.1.1 pacman によるパッケージ管理

MSYS は pacman というツール（コマンド）でパッケージを管理する．このコマンドの起動時にオプションを付けることで，様々な管理操作ができる．

■ インストールされているパッケージの確認： `pacman -Q`

例.

```
$ pacman -Q   Enter      ←コマンドの発行
bash 4.4.019-3
bash-completion 2.8-1
bsdcpio 3.3.2-1
bsdtar 3.3.2-1
bzip2 1.0.6-2
      :
   (以下省略)
      :
```

[12] 各種プログラミング言語のコンパイラ・コレクションとして有名である．
[13] Windows 上に実現する UNIX（POSIX）API 互換レイヤーである．公式インターネットサイトは https://www.cygwin.com/

■ **パッケージのインストール：** `pacman -S　パッケージ 1　パッケージ 2　…`

MSYS 用の各種パッケージは**リポジトリ**として公開されており，オンラインでパッケージを入手してインストールすることができる．始めて MSYS を導入する際は，リポジトリから基本的な開発ツールを導入する．この作業は下記のコマンド操作による．

```
pacman -S base-devel msys2-devel
```

このコマンドを発行した後，未インストールのものを確認してインストールする．(処理が終了するまでしばらく待つ)
次に，MinGW のツール群をインストールするために下記のコマンド操作を行う．

```
pacman -S mingw-w64-x86_64-toolchain
```

先と同様に，処理が終了するまでしばらく待つ．
これらが終われば，後は個別にツール（例えば vim，cmake，winpty など）をインストールする．

■ **パッケージをリポジトリから検索：** `pacman -Ss　キーワード`

指定したキーワードを含むパッケージをリポジトリから検索する．例えば GUI のライブラリである wxWidgets の存在を検索するには次のようにする．

例. リポジトリから wxWidgets を検索

```
$ pacman -Ss wxwidgets  Enter    ←検索開始
mingw32/mingw-w64-i686-wxPython 3.0.2.0-7
    A wxWidgets GUI toolkit for Python (mingw-w64)
mingw32/mingw-w64-i686-wxWidgets 3.0.4-1
    A C++ library that lets developers create applications for Windows, Linux
    and UNIX (mingw-w64)
mingw64/mingw-w64-x86_64-wxPython 3.0.2.0-7
    A wxWidgets GUI toolkit for Python (mingw-w64)
mingw64/mingw-w64-x86_64-wxWidgets 3.0.4-1
    A C++ library that lets developers create applications for Windows, Linux
    and UNIX (mingw-w64)
```

■ **パッケージデータベースの更新：** `pacman -Sy`

■ **インストール済みパッケージの更新：** `pacman -Su`

まとめて `pacman -Syu` として実行しても良い．

■ **指定したパッケージの削除：** `pacman -R　パッケージ`

依存関係も含めて削除するには「`pacman -Rs　パッケージ`」とする．

4 Python での応用例

4.1 MeCab との連携

natto-py パッケージを導入することで Python から MeCab を呼び出して日本語文章の形態素解析ができる．natto-py パッケージについての情報はインターネットサイト

https://github.com/buruzaemon/natto-py
https://pypi.python.org/pypi/natto-py

を参照のこと．また，natto-py とは別に mecab-python3，mecab-python-windows （Windows 用）というパッケージもあり，下記のインターネットサイトから入手することができる．

https://pypi.org/project/mecab-python3/
https://pypi.org/project/mecab-python-windows/

ここでは mecab-python3，mecab-python-windows の使用方法を紹介する．

4.1.1 mecab-python3/mecab-python-windows

このモジュールは Python 処理系から MeCab を呼出して文章の形態素解析を行う．使用に際しては，次のようにしてモジュールを読み込む．

```
import MeCab
```

次に形態素解析のために Tagger クラスのオブジェクトを生成する．（下記参照）

例. モジュールの読込みと Tagger オブジェクトの生成

```
>>> import MeCab   Enter      ←モジュールの読込み
>>> m = MeCab.Tagger()   Enter      ← Tagger オブジェクト m を生成
```

この例では Tagger クラスのインスタンス m が生成されている．この m に対して parse メソッドを実行することで解析結果が得られる．

例. “私は大阪府に住んでいます．”の解析

```
>>> r = m.parse('私は大阪府に住んでいます．')   Enter      ←文章解析
>>> print( r )   Enter      ←結果の表示
私      名詞,代名詞,一般,*,*,*,私,ワタシ,ワタシ
は      助詞,係助詞,*,*,*,*,は,ハ,ワ
大阪    名詞,固有名詞,地域,一般,*,*,大阪,オオサカ,オーサカ
府      名詞,接尾,地域,*,*,*,府,フ,フ
に      助詞,格助詞,一般,*,*,*,に,ニ,ニ
住ん    動詞,自立,*,*,五段・マ行,連用タ接続,住む,スン,スン
で      助詞,接続助詞,*,*,*,*,で,デ,デ
い      動詞,非自立,*,*,一段,連用形,いる,イ,イ
ます    助動詞,*,*,*,特殊・マス,基本形,ます,マス,マス
.       記号,句点,*,*,*,*,.  ,.  ,.
EOS
```

この例のように，parse メソッドの引数に解析対象の文章の文字列を与える．parse メソッドは解析結果を文字列で返す．この文字列は ‘￥n’ で区切られたレコードの形式であり，各行のフォーマットは

単語　￥t　属性 1，属性 2，…￥n

である．また ’EOS’ は解析の終端を意味する．（「1.2 MeCab(和布蕪)：テキスト形態素解析」を参照のこと）

4.1.2 MeCab の解析結果の取り扱いについて

MeCab の解析結果は文字列で得られるが，実際の形態素解析処理においては使いやすいデータ構造の形に変換する方が良い．

次のプログラム MeCabEx.py は，MeCab モジュールの機能を応用して，形態素解析の結果をリストとして取得するものである．

プログラム：MeCabEx.py

```
# coding: utf-8
import MeCab

###############################################################################
#   mecab-python準拠の機能                                                    #
###############################################################################
class Tagger( ):
    # コンストラクタ
    def __init__(self):
        self.M = MeCab.Tagger()

    # 文章解析
    def parse(self,s):
        self.R = self.M.parse(s)
        s1 = self.R.split('\n')
        s2 = []
        for m in s1:
            s3 = m.split('\t')
            if s3[0] == 'EOS':
                break
            else:
                s4 = [s3[0]]
                for n in s3[1].split(','):
                    s4.append(n)
                s2.append(s4)
        self.A = s2
        return( self.R )

    # 解析結果表示
    def show(self):
        for m in self.A:
            msg = m[0]+'\t'+','.join(m[1:])
            print(msg)

    # 列の取り出し
    def col(self,n):
        r = []
        for m in self.A:
            if len(m) <= n:
                r.append('')
            else:
                r.append(m[n])
        return( r )

###############################################################################
#   モジュールの実行テスト                                                    #
###############################################################################
if __name__ == '__main__':
    m = Tagger()
    s = '私は大阪府に住んでいます．'
    print( '（元の文章）',s,'\n' )
    m.parse('私は大阪府に住んでいます．')
    print('（解析結果）')
    m.show()
    print('\n(単語リスト)')
    w = m.col(0)
    print( w )
    print('\n（再構成）',''.join(w))
```

MeCabEx モジュールの使用方法

MeCabEx モジュールは，MeCab モジュールと同様の方法で使用することができ，解析結果をリストとして得ることもできる．解析処理の例を次に示す．

例. モジュールの読込みと Tagger オブジェクトの生成

```
>>> import MeCabEx  Enter      ←モジュールの読込み
>>> m = MeCabEx.Tagger()  Enter      ← Tagger オブジェクト m を生成
```

MeCab モジュールの場合と同様の処理ができる．(下記参照)

例. 文書の形態素解析 (1)

```
>>> r = m.parse('私は大阪府に住んでいます．')  Enter      ←文章解析
>>> print( r )  Enter      ←結果の表示
私      名詞, 代名詞, 一般,*,*,*, 私, ワタシ, ワタシ
は      助詞, 係助詞,*,*,*,*, は, ハ, ワ
大阪    名詞, 固有名詞, 地域, 一般,*,*, 大阪, オオサカ, オーサカ
府      名詞, 接尾, 地域,*,*,*, 府, フ, フ
に      助詞, 格助詞, 一般,*,*,*, に, ニ, ニ
住ん    動詞, 自立,*,*, 五段・マ行, 連用タ接続, 住む, スン, スン
で      助詞, 接続助詞,*,*,*,*, で, デ, デ
い      動詞, 非自立,*,*, 一段, 連用形, いる, イ, イ
ます    助動詞,*,*,*, 特殊・マス, 基本形, ます, マス, マス
.       記号, 句点,*,*,*,*,. ,. ,.
EOS
```

これに加えて，解析結果をリストの形として得ることができる．(下記参照)

例. 文書の形態素解析 (2)

```
>>> m.col( 0 )  Enter      ← 0 番目の列（単語列）の取得
['私','は','大阪','府','に','住ん','で','い','ます','．']      ←結果表示
>>> m.col( 1 )  Enter      ← 1 番目の列（品詞列）の取得
['名詞','助詞','名詞','名詞','助詞','動詞','助詞','動詞','助動詞','記号']   ←結果表示
```

このように col メソッドを使用して，MeCab からの出力の特定の列を取り出すことができる．col メソッドの引数には列のインデックスを与える．

このプログラムでは，parse メソッドを実行すると解析結果を Tagger オブジェクト内に保持しており，show メソッドを使用することで再度表示することができる．解析結果は次回の parse メソッドの実行まで保持され，その間 show, col メソッドは同じ処理結果となる．解析結果は Tagger オブジェクトの A プロパティに保持されている．

例. Tagger オブジェクトの A プロパティ

```
>>> m.A  Enter      ← A プロパティの参照
[['私','名詞','代名詞','一般','*','*','*','私','ワタシ','ワタシ'],
 ['は','助詞','係助詞','*','*','*','*','は','ハ','ワ'],
 ['大阪','名詞','固有名詞','地域','一般','*','*','大阪','オオサカ','オーサカ'],
 ['府','名詞','接尾','地域','*','*','*','府','フ','フ'],
 ['に','助詞','格助詞','一般','*','*','*','に','ニ','ニ'],
 ['住ん','動詞','自立','*','*','五段・マ行','連用タ接続','住む','スン','スン'],
 ['で','助詞','接続助詞','*','*','*','*','で','デ','デ'],
 ['い','動詞','非自立','*','*','一段','連用形','いる','イ','イ'],
 ['ます','助動詞','*','*','*','特殊・マス','基本形','ます','マス','マス'],
 ['.','記号','句点','*','*','*','*','.','.','.']]
```

4.2 日本語テキストの読み上げ（Open JTalk と PyAudio の応用）

Open JTalk と PyAudio モジュールを併用することで，テキスト読み上げの機能を実現することができる．ここではプログラムの 1 例を示す．

■ 実装の概要

サンプルプログラム kspeech.py は，与えたテキスト（文字列型データ）を読み上げる関数 speak を実装するものである．このプログラムはモジュールとして他の Python プログラムに読み込んで使用することができる．他のプログラムから利用する場合はこの kspeech.py を

```
import kspeech
```

として読み込んで，

```
kspeech.speak(文字列データ)
```

と記述することで与えた文字列を読み上げることができる．

このプログラム（speak 関数）は，与えた文字列データを一旦テキストファイル kspeech_tmp.txt として保存した後に open_jtalk コマンドを起動してそれを WAV 形式ファイル kspeech_tmp.wav に変換する．更にその WAV 形式ファイルを PyAudio モジュールを使用して再生する．

プログラム：kspeech.py

```
# coding: utf-8
#############################################################################
#   kspeech.py                                                              #
#   0.5版        2017/08/13                                                 #
#############################################################################
# 必要なモジュール
import subprocess
import wave
import pyaudio

# 作業ファイル
ftxt = 'kspeech_tmp.txt'    # 読み上げ作業ファイル
fout = 'kspeech_tmp.wav'    # 作業用音声ファイル
#=========================================================================#
#   PyAudioによる再生                                                     #
#=========================================================================#
# WAVファイルの再生
def playwave():
    # 再生用コールバック関数
    def sndplay(in_data, frame_count, time_info, status):
        buf = wf.readframes(frame_count)
        return( buf, pyaudio.paContinue )
    # 音声ファルのオープン
    wf = wave.open(fout, 'rb')
    # 属性情報の取得
    ch = wf.getnchannels()      # チャネル数
    qb = wf.getsampwidth()      # 量子化ビット数（バイト数）
    fq = wf.getframerate()      # サンプリング周波数
    # PyAudioオブジェクトの生成
    p = pyaudio.PyAudio()
    # ストリームの生成
    stm = p.open( format=p.get_format_from_width(qb),
                  channels=ch, rate=fq, output=True,
                  stream_callback=sndplay )
    # コールバック関数による再生を開始
    stm.start_stream()

#=========================================================================#
#   open_jtalkコマンドによる読み上げ                                      #
#=========================================================================#
#----- open_jtalkコマンド -----
cmd0 = 'C:\OpJTalk\open_jtalk'              # コマンド本体
```

```
43 cmd1 = '-x C:\OpJTalk\dic_utf_8'             # 辞書ディレクトリ
44 cmd2 = '-m C:\OpJTalk\mei_normal.htsvoice'  # 音響モデル
45 cmd3 = '-s 44100 -r 1.0 -ow '   # サンプリング周波数と読み上げ速度など
46 # 実行用コマンド
47 cmd = cmd0+' '+cmd1+' '+cmd2+' '+cmd3+' '+fout+' '+ftxt
48
49 #----- 読み上げ -----
50 def speak(s):
51     # 読み上げ作業ファイル作成
52     f = open(ftxt,'w',encoding='sjis')
53     f.write(s)
54     f.close()
55     # open_jtalkのプロセス生成
56     sp = subprocess.Popen(cmd,shell=False,
57                 stdin=None, stdout=None, stderr=None)
58     # プロセス実行
59     sp.wait()   # open_jtalk終了待ち
60     playwave()  # 再生
61
62 #=================================================================#
63 #       kspeech.pyパッケージ実行テスト                             #
64 #=================================================================#
65 if __name__ == '__main__':
66     print('--- kspeech.py: begin ---')
67     while True:
68         s = input('テキスト入力:(終了は\'exit\')  ')
69         if s == 'exit':
70             print('終了します…')
71             break
72         else:
73             speak(s)
74     print('--- kspeech.py: end ---')
```

このプログラムは Python インタプリタで直接実行することもでき，その場合は，65 行目以降の部分が実行され，キーボードから入力された文字列を読み上げる．

4.3 Python と SWI-Prolog の連携

PySwip モジュールの利用により，Python と SWI-Prolog を連携することができる．PySwip に関する情報はインターネットサイト

https://github.com/yuce/pyswip

から得ることができる．

4.3.1 基本的な使用方法

PySwip モジュールが提供する Prolog クラスのオブジェクトを生成すると，それに対応する SWI-Prolog 処理系が 1 つ起動して連携が始まる．

例. Prolog オブジェクトの生成

```
>>> from pyswip import Prolog  Enter      ← Prolog クラスの読込み
>>> pr = Prolog()  Enter      ← Prolog オブジェクト pr の生成
```

この例では，Prolog クラスの pr オブジェクトを生成している．この処理により SWI-Prolog 処理系が起動し，pr オブジェクトを介して SWI-Prolog 処理系に各種の処理を依頼することができる．

例. 述語の SWI-Prolog 処理系への登録

```
>>> pr.assertz('human(taro)')  Enter
>>> pr.assertz('human(jiro)')  Enter
>>> pr.assertz('human(hanako)')  Enter
```

この例では，assertz メソッドを用いて pr に 3 つの述語を登録している．SWI-Prolog への問い合わせ（節の単一化）には query メソッドを使用する．

例. SWI-Prolog 処理系への問い合わせ

```
>>> q = pr.query('human(X)')  Enter      ←問い合わせ処理
>>> list(q)  Enter      ←単一化の結果をリストに変換
[{'X': 'taro'}, {'X': 'jiro'}, {'X': 'hanako'}]      ←単一化の結果
```

query メソッド実行により，単一化の結果が返される．この例では結果をリストに変換している．この処理により Prolog の変数 への単一化が辞書オブジェクトとして得られる．単一化の全ての変数の組み合わせが，それぞれの辞書オブジェクトを要素とするリストとして得られる．

SWI-Prolog 処理系に Prolog プログラムのソースを読み込ませるには consult メソッドを使用する．Prolog のプログラム test02.pl を読み込む例を示す．

Prolog のプログラム：test02.pl

```
1  human(taro).
2  human(hanako).
3
4  dog(pochi).
5  dog(nana).
6
7  deceased(nana).
8
9  cat(ruru).
10 cat(tama).
11
12 animal(A) :- human(A).
13 animal(A) :- dog(A).
14 animal(A) :- cat(A).
15
16 mortal(A) :- animal(A), not(deceased(A)).
```

例. Prolog のプログラム test02.pl の読込みと問い合わせ

```
>>> from pyswip import Prolog  Enter      ← Prolog クラスの読込み
>>> pr = Prolog()  Enter      ← Prolog オブジェクト pr の生成
>>> pr.consult('test02.pl')  Enter      ← Prolog プログラムの読込み
>>> q = list( pr.query('cat(A)') )  Enter      ←問い合わせ処理（ ?- cat(A) ）
>>> q  Enter      ←結果の確認
[{'A': 'ruru'}, {'A': 'tama'}]
>>> q = list( pr.query('mortal(A)') )  Enter      ←問い合わせ処理（ ?- mortal(A) ）
>>> q  Enter      ←結果の確認
[{'A': 'taro'}, {'A': 'hanako'}, {'A': 'pochi'}, {'A': 'ruru'}, {'A': 'tama'}]
```

索 引