P A D manual

(pad2ps 3.1j)

吉田 誠一 (seiichi@muraoka.info.waseda.ac.jp)

1996 年 11 月 2 日

はじめに

pad2ps シリーズは、プログラムの PAD 図を描くためのツールである。

PAD (Problem Analysis Diagram) とはプログラムの流れや構造を図示したもので、同様なものにフローチャートがある。

フローチャートは世界的に有名で、広く一般にも広まっているが、それはフローチャートの書きやすさが原因であろう。ところがこのフローチャートは、プログラムの処理の **流れ** を考える上では大いに役に立つのだが、アルゴリズム、特に C 言語のような構造化プログラミングでのアルゴリズムを表す、という目的には向いていない。その理由は次のとおりである。まず第一に、フローチャートを理解するには、始めから順に流れに沿って追っていかないといけない、ということがある。これは、大きなプログラムになった場合、全体を理解する上で障害となる。第二に、フローチャートは構造を表すことができないので、同じアルゴリズムでも、例えばたった 1 つのループでも、人によって描き方が大きく異なるということもある。逆に言えば、2 つの図の形が同じように見えても、片方は分岐でもう一方はループになっていたということもある。これはアルゴリズムを図示する上で致命的な欠点である。

一方、PAD はアルゴリズムの構造を表すための図である。フローチャートとの大きな違いは、反復や条件分岐などの **構造** を明確に表すことができる点である。この PAD を描くためには自分自身でアルゴリズムを理解していないといけないため、フローチャートのように手軽に書く訳にはいかない。しかし、アルゴリズムを他者に示す時には、フローチャートに比べて、その処理と構造をより容易に理解してもらうことができる。

結論として、フローチャートは GOTO のある BASIC のような言語のプログラムを図示するのに向いているが、C のような構造化プログラミング言語を表すのには、この PAD の方が向いていると言える。特に、本やレポートなどでアルゴリズムの紹介をする際や、後の保守のために自身のプログラムの図をとっておこう、というような場合が、この PAD が最も役に立つ場面である。

pad2ps シリーズは、このような PAD 図をドローツールを使わずに、テキストベースで簡単に描くためのツールである。元となるファイルは、**PADEL**(PAD Expression Language) という C 言語に似た文法に従って書く。また、C, C++, Bourne shell, C shell, Java, AWK のプログラムの PAD 図を直接プログラムから描かせることもできる。出力形式は PostScript、PDF、LaTeX、Encapsulated PostScript (EPS) の 4 種類である。

この **pad2ps** シリーズの製作にはたくさんの方が御協力下さった。倉持 聡氏には LaTeX のスタイルファイル **pad.sty** を使わせて頂いただけでなく、**pad2tex** の開発中にもデバッグ、改良などをして頂いた。加藤 淳也氏には日本語化に際してなど、初期の頃の製作全般に渡っていろいろと協力して頂いた。井口 誠氏には **pad2ps** シリーズの英語版作成、マニュアル等の英訳をして頂いた。小副川 尚氏、藤井 大輔氏、三上 浩士氏にはいろいろと有益な助言を頂いた。それらの中にはまだ実現していないこともあり、今後の課題である。その他にも、実にたくさんの方々からバグ情報、修正、改良案などを頂いた。改めて感謝の意を表したい。

1996 年 11 月 2 日

吉田 誠一

目 次

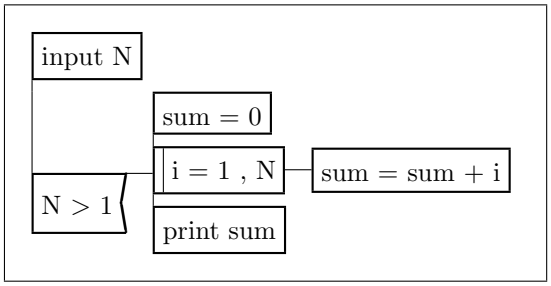


図 1: 1 から N までの足し算

1 PAD 図を描いてみよう

この章では、**pad2ps** シリーズを使って実際に PAD 図を描いてみる時の、コマンドの操作方法、及び出力ファイルを利用する上での注意点などを説明する。

1.1 まずはエディタを使って

pad2ps シリーズは PAD をテキストとして書いて、それを自動で図に変換するものであるから、まずはエディタで PAD のソースファイルを作成しなければいけない。PAD のソースは **PADEL**(PAD Expression Language) の文法に従って書く。**PADEL** については、第 2 章で説明するので、とりあえずここでは 1 つ例を挙げておこう。例えば、図 1 の PAD 図を描くためには、次のようなファイルを用意する。

もし貴方が C 言語のプログラムを知っているならば、そっくりであることに気づくだろう。**pad2ps** シリーズで PAD 図を描くためには、このような C 言語もどきのファイルを作ってやればよい。

```
1 から N までの足し算
{
    input N

    if(N > 1){
        sum = 0
        for(i = 1 , N){
            sum = sum + i
        }

        print sum
    }
}
```

では、このファイルを `foo.pad` という名前で保存しておこう。

1.2 PostScript にする

まずは PostScript 形式にしてみよう。そのためには、**pad2ps** コマンドを使って、次のようにする。

```
% pad2ps foo.pad > foo.ps
```

PostScript 形式に変換した場合は、それ自身で完成した 1 つの作品となる。そのため、いろいろな装飾を施すことが可能なようになっている。具体的には、パラメータ変数を指定する。どのようなパラメータ変数があり、どうなるかは、第 4 章を参照していろいろと試してみてほしい。

1.3 PDF にする

PostScript の表示はできなくても、PDF の表示ができる環境は多い。PDF 形式にする場合は、**pad2pdf** コマンドを使って、次のようにする。

```
% pad2pdf foo.pad > foo.pdf
```

1.4 LaTeX にする

LaTeX 形式で出力させるには、**pad2tex** コマンドを使う。

```
% pad2tex foo.pad > foo.tex
```

LaTeX 形式の場合、これだけでも既に dvi ファイルにして見ることができる。

```
% jlatex foo.tex
% xdvi foo.dvi
```

しかし、C 言語のプログラムを変換させる場合のようにファイルが大きい場合は、PAD 図が右にはみ出してしまうことが多いだろう。そのような時は無理に LaTeX に変換せず、PostScript に変換する方がきれいな結果を得ることができるだろう。どうしても LaTeX にしたければ、**pad2tex** で LaTeX のコードに変換してから、手動で \\ を挿入して適当に改行させる必要がある。

だがやはり、LaTeX に変換させるのは主に小さな、1 ページ以内に収まるような PAD 図を描き、LaTeX で書いた本やレポートの中にはめ込む場合であろう。その時は、変換した `foo.tex` の中から、`\begin{figure}[h]` と`\end{figure}` で囲まれた部分、またはその中の `\padseq{%` や`\fbox{%` などから `}` までの一連のコマンド群を抜き出し、貴方の本やレポートの適当な場所に埋め込んでやればよい。このマニュアルの中の PAD 図もそのようにして作ったものである。

また、先頭の `\documentstyle` 文で、**pad.sty** [1]をロードするように指定する。必要ならそれ以外のコマンドもコピーしておく。例えば、`\fboxsep=10pt` という行をコピーしないと、PAD 図を囲む枠との間隔が狭くなってしまう。

ところで、**pad2tex** を使って LaTeX 形式にする場合には、

```
/* pad2ps: texmode = latex */

\LaTeX の命令を使った例
{
    /* pad2ps: message = $\sqrt{2}$ を求める */
    $ x = 1 $
    while ( $ | x^{2} - 2 | > 0.0001 $) {
        $ \displaystyle x = x - \frac{x^{2} - 2}{2 x} $
    }
}
```

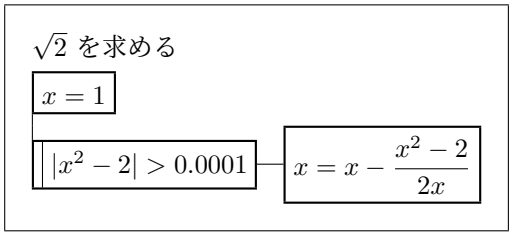


図 2: LaTeX の命令を使った例

のように、パラメータ変数 **texmode** に latex を指定することで LaTeX の命令を使うこともできる。この例では図 2 のような PAD 図が描かれる。

ちなみに、**pad2tex** の出力する LaTeX コードはデフォルトでは 4 カラムごとにインデントされているが、これは `-tab` オプションで変更できる。インデントが不要ならば、`-tab 0` とすればよい。

1.5 Encapsulated PostScript(EPS) にする

EPS 形式に変換する場合というのは、大きな、複数ページに渡るような PAD 図を描いて、それを縮小して LaTeX のレポートなどに並べる、というような時だろう。

EPS 形式にする場合は、**pad2eps** コマンドを使って次のようにする。

```
% pad2eps foo.pad foo
```

pad2ps、**pad2tex** のようなコマンドと違って、入力ファイル名だけでなく、出力ファイル名のヘッダを指定する。もし **pad2ps** をつかって PostScript 形式にした時に複数ページになったとしたら、できる EPS ファイルも複数になる。そのファイル名は、指定したヘッダに `foo.1.eps`、`foo.2.eps`、··· のように、ページ番号と拡張子 `.eps` がついたものとなる。もしヘッダを指定しなかった場合は、`.1.eps` のようなファイル名のファイルができてしまい、`ls` では現れなかったりするので注意してほしい。尚、必ず出力ファイル名を指定するため、標準入力から読み込ませる場合には`-stdin` オプションを使って、

```
% cat foo.pad | pad2eps -stdin foo
```

のようにしなければいけない。

さて、EPS 形式の PAD 図を LaTeX のレポートなどで使用するには、まず先頭の`\documentstyle` 文で **epsf.sty** をロードするようにし、そして EPS ファイルを読み込みたい場所で次のようにする。

```
\epsfile{file=foo.1.eps,width=5cm,height=7cm}
```

EPS ファイルを取り込んだ例は第 4.15 節の図 3 にある。

[1]倉持 聡氏作 **pad.sty** 1.1j。

2 PADELの文法について

この章では **PADEL**(PAD Expression Language) の文法について説明する。

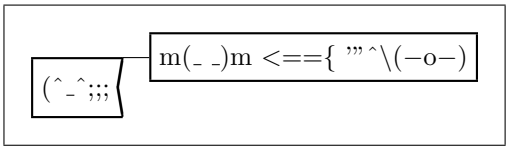
PAD 図を描くためには、まずテキストファイルとして PAD のソースファイルを作ってやり、それを第 1 章で述べたように **pad2ps** シリーズのコマンドを使って変換する。この PAD のソースは **PADEL** という文法に従って書くが、この書き方は C 言語のプログラムとよく似ている。であるから、C 言語を多少なりとも知っていれば、PADEL で PAD 図を描くのは思ったより簡単であろう。

ただ、注意しないといけない点もいくつかある。特に大事なことは、改行コードが重要な意味を持っている、ということである。以下の節で、改行する、といわれたら必ず改行し、また余計なところでは改行はするべきではない。改行コードが重要になったのは、文字列に自由度を持たせたことが原因である。それに比べて、空白はかなり自由に使うことができる。特に、文頭の空白は無視されるので、好きなようにインデントして構わない。

文字列については、他の言語に比べて格段に自由度が高い。ブレース、括弧などは自由に使用して構わないし、左右の対応がとれていなくても問題ない。" と ' についても同様で、" が 1 つだけしかなくても構わない。また、\ のような記号も、英数字と全く同様に書くことができる。これは、たとえ記号であろうとも、なるべく PADEL に書いた文字をそのまま図の中に書けるようにした方が、**pad2ps** シリーズの使用目的に合っているのではないか、という考えによるものである。よって、

```
if( (^_^;;; )
    m(_ _)m <==={ '"^\(-o-)
```

というイラストのような PADEL を書いても、



のように、書いたままの PAD 図を描くことができる。

また、1.4 節の **pad2tex** のところで述べたように、パラメータ変数 **texmode** に latex を指定した場合はそのまま出力されるのではなく、LaTeX によって LaTeX の命令として処理される。この場合も、上記の方針により、わざわざ \ を \\ などとする必要がなく、LaTeX のコマンドをそのまま PADEL に書くことができる。

2.1 PADの構成とタイトルについて

PADEL での PAD の一般的な書き方は、次のようなものである。これは n の階乗を求める関数である。

```
factorial(n)
{
    f = 1
    for(i = 1, n)
        f = f * i
    return f
}
```

このように、PADEL での 1 つの PAD は、まず始めにタイトルを書き、命令群を{ } で囲む、という形になっている。{ } の中は空でも構わない。これは C 言語の関数定義と同様である。また、1 つの C 言語のプログラムの中に複数の関数を書いても構わないように、1 つの PADEL ファイルの中に複数の PAD を書くことができる。

ここで、いくつか注意すべきことがあるので、述べておこう。

まず、PAD のタイトルは必ず 1 行で書かなければいけない。また、タイトルを書いたら必ず改行しなければいけない。よって、C のように

```
factorial(n){
    f = 1
    for(i = 1, n)
        f = f * i
    return f
}
```

と書いてしまったらエラーとなる。これは不必要な制限のように思えるかもしれないが、逆にブレースも含めて、任意の文字をタイトルに使用できる、という利点がある。その点については、この章の始めの方で詳しく述べているので、そちらを参照してほしい。

ところで、先程は PADEL の命令群が { } で囲まれている、と述べたが、正確に言えば、{(左ブレース) だけの行が 1 つの PAD の始まりを表し、}(右ブレース) だけの行が 1 つの PAD の終りを意味している、ということになる。決して、{ や } が単独で意味を持っている訳ではないので、間違えないようにしてほしい。従って、改行を省略してはいけない。

タイトルに使用する文字はほとんど何でもよく、PADEL に書いた文字列がそのまま出力される。但し、いくつか例外がある。下記のようなタイトルだけはつけることができない。

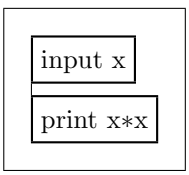
- { や } 1 文字だけのタイトル
- /* や // で始まるタイトル

また、PAD のタイトルは省略することもできる。

2.2 文を書く

まずは例を載せておこう。

```
input x
print x*x
```



このように、ふつうにシーケンシャルな処理を行うような時は、1 つ 1 つの処理をふつうの文として、1 行ずつ書いて行けば良い。

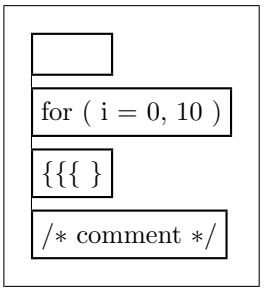
ふつうの文を書く上での注意点は、必ず 1 行に 1 つだけ処理を書く、ということである。途中で改行してはいけないし、また 2 つの処理を 1 行に書いてもいけない。

文に使用する文字はほとんど何でもよい。たいていはそのまま書くだけでよいだろう。しかし、中には特殊な場合があり、そのような時は例外として、特別な書き方をしなければいけない。特別な場合とは、次のような場合である。

- 空白のボックスを描きたい時
- for、while などの予約語で始まる時
- { で始まる時
- } 1 文字だけの時
- }else 、}while 、}catch で始まる時
- L1: のようなラベルの省略形と見なされる文字列を書きたい時
- /* や // で始まる時

これらの場合には、{ と } を使って、次のように書く。この時、{ から } までは 1 行で書かなければいけない。

```
{        }
{ for ( i = 0, 10 ) }
{ {{{ } }
{ /* comment */ }
```



ちなみに、{ と } とで囲まれた文字は、どんなものでも出力できる。そこで、すべての文を { と } とで囲んだとしても別に構わない。

2.3 空行による間隔調整

PADEL では、空行を書くことによって、縦方向にある一定のスペースを空けることができる。

空行は PAD の内部だけでなく、PAD と PAD の間にある場合にも同様にスペースを空ける。但し、最初の PAD より前にある場合、つまりファイルの先頭だけは例外で、ここにある空行は無視される。

空行がこのような意味を持っているため、下手な位置に空行を入れてしまうとエラーになることがある。

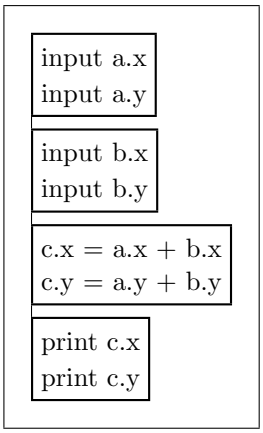
実際に PAD 図を描いてみて、もっと微妙な調整をしたいと思ったら、パラメータ変数 **vsp** を設定する。詳しくは、第 4.18 節を見てほしい。

2.4 ブロック

PAD 図では 1 つの処理が 1 つのボックスで描かれるが、時には複数の処理をまとめて、1 つのボックスで描いた方が分かりやすいことがある。このように、複数行をまとめて 1 つのボックスで描くようにしたものをブロックと呼んでいる。

ブロックを描くためには、ひとまとめにする一連の命令群を、{(左ブレース) のみの行と、}(右ブレース) のみの行で囲む。例を挙げておこう。これはベクトルの足し算である。

```
{
    input a.x
    input a.y
}
{
    input b.x
    input b.y
}
{
    c.x = a.x + b.x
    c.y = a.y + b.y
}
{
    print c.x
    print c.y
}
```



ブロックの中にはたいていの文字列を書くことができ、それらはそのまま PAD 図の中に出力される。但し、例外として書くことのできない文字列もある。次のようなものである。

- { や } 1 文字だけの行
- /* や // で始まる文字列

また、ブロックの中に空行を入れると、きちんと 1 行分のスペースを空けることができる。

2.5 反復処理を描く

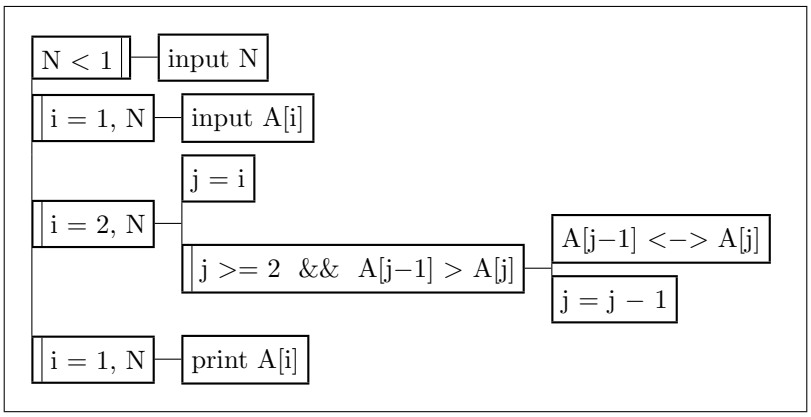
反復には、先に判定し、真ならば内部を実行するということを繰り返す前判定反復と、先に内部を実行し、その後で判定して、真ならばまた繰り返すという後判定反復の 2 種類がある。また、前判定の中には、回数を指定し、その回数だけ反復する、というものもある。PADEL では、前判定反復を表すのに **while** を、後判定を表すのに **do** ～ **while** を、そして回数指定の反復には **for** を使う。これらの書式については、次の例を見れば分かるだろう。

```
do{
    input N
}while(N < 1)
```

```
for(i = 1, N){
    input A[i]
}

for(i = 2, N){
    j = i
    while(j >= 2  &&  A[j-1] > A[j]){
        A[j-1] <-> A[j]
        j = j - 1
    }
}

for(i = 1, N){
    print A[i]
}
```



少々長くなってしまったが、これは N 個のデータをソートして出力するアルゴリズムである。

まずデータ数 N を入力する。この時、入力した値が 1 より小さければ不正なので、再入力を促す。このように、まず何かを行ってから判定するのが後判定である。

次に、N 個のデータを入力する。このような、回数指定の反復は、**for** を使って書く。その後で、ソーティングの際に、1 つ前の値と大小を比較し、前の値の方が大きければ交換して、また繰り返す、という操作をしている。このような場合は前判定なので、**while** を使っている。ところで、**for** 文と **while** 文は共に前判定で、出力される PAD 図の形が同じであるが、一応回数指定の反復が **for**、その他の前判定反復が **while**、というように使い分けておくことを勧める。

さて、最後にいくつか注意すべき点、というか、可能な書式について述べておこう。

まず、反復部が 1 命令のみの場合、`{ }` で囲む必要はない。つまり、

```
for(i = 1, N)
    input A[i]
```

といった書き方が可能である。もちろん `{ }` で囲んだとしても何も問題はない。さらに、

```
for(i = 1, N){
}
```

のように、反復部がなくても構わない。但し、その場合は必ず `{ }` で囲まなければいけない。

ところが、C 言語の真似をして、

```
for(i = 1, N)  input A[i]
```

と書くとエラーになってしまうので注意してほしい。これは、 for(i = 1, N) という反復命令と、 input A[i] という命令と、2 命令を 1 行に書いたことになり、1 行 1 命令の原則に反しているのである。もちろん、

```
for(i = 1, N){  input A[i]  }
```

と書いても駄目である。

また、{ の前や、**do** ～ **while** 文の } の後ろに改行コードを入れても構わない。

```
do
{
    input N
}
while(N < 1)

for(i = 1, N)
{
    input A[i]
}
```

但し、他の箇所では改行はしてはいけない。

また、空白は適当に入れて構わない。

```
for  (  i = 1, N  )  {
    input A[i]
}
```

条件式には、任意の文字列を書くことができる。

2.6 条件分岐を描く

条件分岐を描くための命令は、**if** ～ **else** と **switch** の 2 つである。一般に、**if** ～ **else** はふつうの条件分岐を描くために使い、**switch** は多重条件分岐を描くために使う。

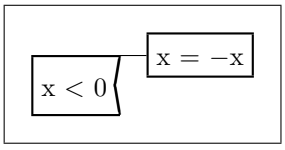
どちらとも、条件式には任意の文字列を書くことができる。

2.6.1 ふつうの条件分岐

ふつうの条件分岐には、**if** ～ **else** 文を使う。ただ、1 つ注意してほしいのは、**if** ～ **else** を使えば必ずふつうの条件分岐になるという訳ではないことだ。つまり、**if** ～ **else** を使っても、書き方によって多重条件分岐になることがあるのである。詳しくは 2.6.2 節に述べてあるので、そちらを参照してほしい。

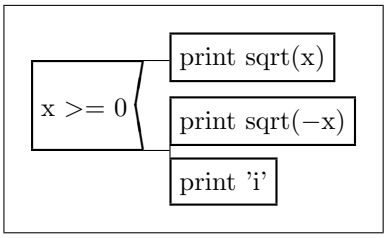
まず、もっとも簡単な条件分岐は、次のような **if** しかないものである。例は、絶対値を求めるプログラムである。

```
if(x < 0)
    x = -x
```



else を使えば、条件が成り立たないときの処理も書くことができる。例として、平方根を出力するプログラムを載せておく。負の数の時には虚数単位 i を出力するようにしてある。

```
if(x >= 0)
    print sqrt(x)
else{
    print sqrt(-x)
    print 'i'
}
```



ふつうの条件分岐についてはこれくらいである。

最後に、**if** ～ **else** 文で可能な書き方、不正な書き方をいくつか例を挙げておく。

まず、最初の例にもある通り、真部や偽部が 1 命令のみの場合は、`{ }` で囲まなくても構わない。但し、

```
if(x < 0)  x = -x
if(x < 0){  x = -x  }
```

という書き方は、共に不正である。さらに、真部、偽部のどちらも中身を省略しても構わない。よって、次のような書き方が可能である。

```
if(x >= 0){
}else{
    x = -x
}

if(x < 0){
}else{
}
```

また、`{` の前や `}` の後に改行コードをいれても構わない。空白も適当にいれることができる。よって、

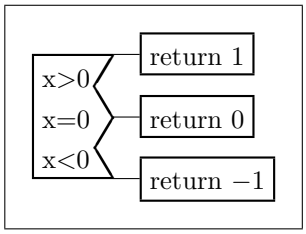
```
if  (  x >= 0  )
{
    print sqrt(x)
}
else
{
    print sqrt(-x)
    print 'i'
}
```

といった書き方も可能である。

2.6.2 多重条件分岐

多重条件分岐には、ふつう **switch** 文を使う。まずは例をみてほしい。これは符号を返す関数である。

```
switch{
    case x>0:
        return 1
    case x=0:
        return 0
    case x<0:
        return -1
}
```



このように、分岐先が 3 つ以上になるのが多重分岐である。それぞれの条件式は、**case** を使って書く。

もし貴方が C 言語を知っていれば、次のような書き方をしたくなるかもしれない。

```
switch(n%3){
    case 0:
        print ３の倍数
    case 1:
        print あまり１
    case 2:
        print あまり２
}
```

このような書き方も可能である。これは、

```
switch{
    case n%3=0:
        print ３の倍数
    case n%3=1:
        print あまり１
    case n%e=2:
        print あまり２
}
```

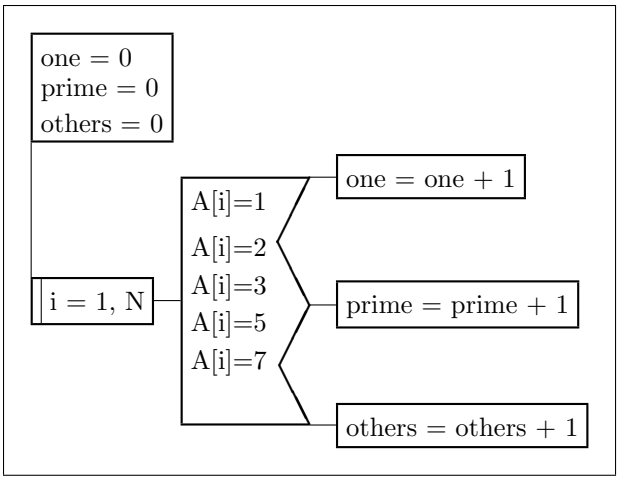
と書いたのと同じ結果となる。

また、複数の条件式を並べて書くこともできる。さらに、どの条件式にも当てはまらない場合の処理を、**default** を使って書くことができる。**default** の時は、PAD 図の中には何も文字は書かれない。では、例を見てみよう。これは、与えられた数列の中の数を 1、1 桁の素数、その他に分類してそれぞれ個数を数えるプログラムである。

```
{
    one = 0
    prime = 0
    others = 0
}

for(i = 1, N){
    switch(A[i]){
        case 1:
            one = one + 1
        case 2:
```

```
            case 3:
            case 5:
            case 7:
                prime = prime + 1
            default:
                others = others + 1
        }
    }
```

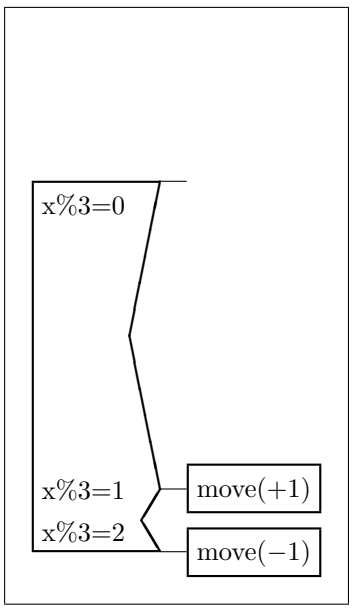


case や **default** の条件式の中身を省略し、その条件に当てはまる場合には何もしないことを表すこともできる。その場合には、条件式の後に空行のみを入れる。例えば、

```
switch{
    case x%3=0:

    case x%3=1:
        move(+1)
    case x%3=2:
        move(-1)
}
```

と書いて、



という PAD 図を描くことができる[2]。

switch 文でも、{ の前で改行することが可能である。また、空白も適当に入れて構わない。この辺りのことについては、反復やふつうの条件分岐と同じである。また、**case** や **default** などの分岐ラベルは、きちんと 1 行に書かなければいけない。よって、

```
switch{
    case n > 0  &&
         m > 0:
                print n*m > 0
    case n < 0: print n < 0
}
```

のような書き方は不正である。

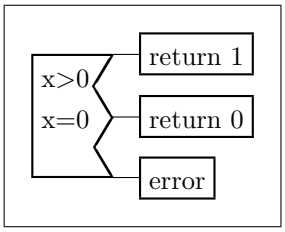
ここで、**if** ∼ **else if** ∼ **else** を使って多重条件分岐を書く場合について述べておこう。次の例を見てほしい。

```
if(x>0)
    return 1
else if(x=0)
    return 0
else
    error
}
```

これは **switch** を使って、

```
switch{
    case x>0:
        return 1
    case x=0:
        return 0
    default:
        error
}
```

と書いたのと全く同じ結果となる。

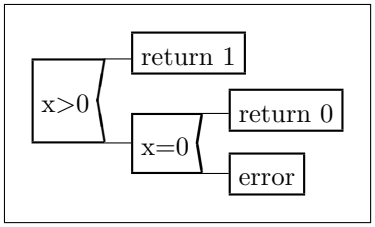


但し、次のように、**else if** の間に改行をいれてしまうと、これは **if** ∼ **else if** ∼ **else** とは見なされなくなってしまう。

```
if(x>0)
    return 1
else
if(x=0)
    return 0
else
    error
```

[2]但し、**pad2tex** を使って LaTeX 形式に変換した場合、ページレイアウト次第では縦に伸びることがある。

この場合の出力結果は次のようになる。ただの **if** ～ **else** 文のネストとなっている。



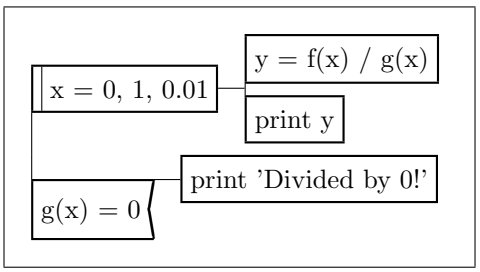
このように、**if** 文を使う時には注意が必要である。思っていたのと違った形の PAD 図が描かれた時には、ここに書いてあることを思い出して、もう一度見直してみてほしい。逆に、これらをうまく使うと、場合によって出力の形を変えてみることも可能になる。

2.7 例外処理

ある命令群を実行し、エラーになったら指定したエラー処理命令群を実行する、という例外処理は **try** ～ **catch** を使って記述する。書式は次の例を見ればだいたい分かるだろう。

```
try{
    for(x = 0, 1, 0.01){
        y = f(x) / g(x)
        print y
    }
}catch(g(x) = 0){
    print 'Divided by 0!'
}
```

この場合の出力結果は次のようになる。



結果は、単なる **if** 文と同じ形になっている。これは、そもそも PAD 自身に例外処理の概念がないためである。

try ～ **catch** 文での改行コードの入れ方や { } の使い方は、反復や条件分岐の場合とほぼ同じである。**try** や **catch** の中身が 1 行であれば、{ } で囲まなくても良い。また、{ や } の前後に改行コードを入れても良いし、空白も適当に入れて構わない。つまり、

```
try
    print sty
catch(str = NULL)
    print 'NULL!'

try
{
    print sty
}
```

```
catch (str = NULL) {
    print 'NULL!'
}
```

といった書き方ができる。一方、

```
try{  print sty  }
catch(str = NULL)  print 'NULL!'
```

という書き方は不正となる。

2.8 ラベルと参照

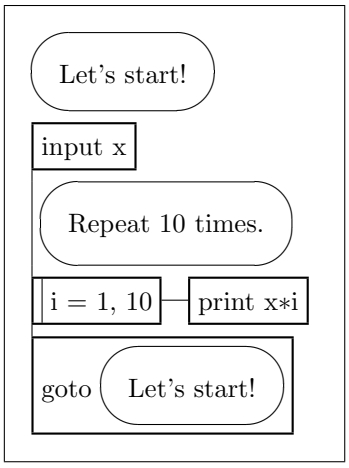
PADEL では、PAD 図の中にラベルを描くことができる。PAD 図ではラベルは円形に描かれる。ラベルを描くためには、**label** 命令を使う。ラベル名はどのようなものでも構わない。つまり、空白や { } なども自由に使うことができる。

また、そのラベルを参照するには、**refer** 命令を使う。参照するラベルも、PAD 図では円形に描かれる。**refer** 文でも、任意のラベル名を使うことができる。参照するラベルの存在はチェックしていないので、存在しないラベル名を書くこともできる。

では例を挙げておこう。

```
label Let's start!
input x
label Repeat 10 times.
for (i = 1, 10)
    print x*i
refer (Let's start!) {
    goto
}
```

この結果は次のようになる。



この例を見ても分かるように、ラベルは **label** 命令の後に記述するだけである。1 行に書かないといけないこと以外に特に注意することもない。

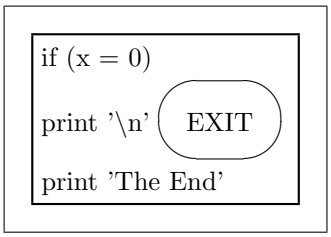
refer 文による参照については、もう少し詳しい解説が必要だろう。**refer** 文では、まず参照先のラベル名を ( ) で囲んで書き、その後で { } で囲んで文を書く。よって、例えば "goto ラベル" として、このラベルを円の中に書きたい時には、次のように 3 行に渡って書く。

```
refer (LABEL) {
    goto
}
```

この書き方は既に述べた反復や条件分岐などとほぼ同じであり、改行コードや空白の入れ方についてはここでは省略する。但し、{ } の中身の扱いは反復や条件分岐とは異なるので注意が必要である。**refer** 文では、{ } の中身はブロック文と同じで、単なる文字列として扱われる。例えば

```
refer (EXIT) {
    if (x = 0)
        print '\n'
    print 'The End'
}
```

の結果は



となる。

また、ラベルには省略形がある。C 言語のラベルと同じで、英数字及び _ (アンダースコア) を組み合わせたラベル名の後ろに : (コロン) をつけただけの行はラベルの省略形と見なされる。PADEL では更に日本語も含めることができる。例えば、

```
LABEL1:
```

は

```
label LABEL1
```

と同じ結果になる。但し、1 行 1 命令の原則より、: (コロン) の後に続けて別の PADEL の命令を書いてはならず、必ず改行しなければいけない。空白は、ラベル名の前、ラベル名と : の間、: の後のいずれにも入れて構わないが、ラベル名が空白を含んではいけない。また、省略形のラベル名には `for` などの予約語は使えない。

2.9 コメント

他のプログラム言語と同様に、PADEL にもコメントを書くことができる。コメントの内容は出力される PAD 図の中には書かれない。

コメントの書き方は C、または C++のコメントと同じである。つまり、

```
/* コメント */
// コメント
```

の 2 通りの方法が可能である。

但し、コメントを書く場所は任意ではないので注意が必要である。PADEL では、コメントはコメントのみの行として独立していなければいけない。つまり、1 行に PADEL の命令とコメントを共に書くことはできないのである。正確に言うと、`/*` や `//` の前には空白以外の文字はあってはならず、また `*/` の後には、改行コードとの間に空白以外の文字があってはならない。よって、

```
for ( i = 0, N )        /* N までの和 */
    sum = sum + i
```

のような書き方は不正である。ちなみに、C ライクなコメントの場合、途中に改行コードが入るのは構わない。

ところで、コメントは本来のコメントとしての役割以外に、パラメータ変数を指定するのにも使われる。パラメータ変数を使うと、PAD 図の仕上がりをよりいっそうきれいに装飾することができる。パラメータ変数については、第 4 章で詳しく述べているので、そちらを参照してほしい。

3　プログラムから直接PAD図を描く

この章では、**pad2ps** シリーズを使って C 言語等の各種プログラムの PA D 図を描かせる方法について説明する。他人の作ったプログラムはえてして読みにくいものであるが、**pad2ps** シリーズを使えば格段に理解しやすくなる。

ところで、プログラムの PAD 図を描かせる場合は、**pad2ps** シリーズのコマンドは内部から各言語ごとのパーサを呼び出して、いったん PADEL の文法に合わせたテキストファイルを作成した後、改めてそれを読みだして PAD 図を描く、という 2 段階の処理を行っている。もしプログラムの PAD 図をそのまま描くのではなく、自分で修正したい時には、プログラムを PADEL のファイルに変換し、それを修正するようにすれば、元のプログラムには手をつけなくて済む。各種プログラムを PADEL に変換するには、`mkpad` コマンドを使う。例えば C 言語なら、

```
% mkpad -c foo.c > foo.pad
```

として、この `foo.pad` を修正し、その後 `pad2ps` シリーズで変換する。

ちなみに、`pad2ps` シリーズでプログラムから直接 PAD 図を描く場合は、エラーになった時に出力される行番号は元のプログラムの行番号ではなく、内部で呼び出したパーサの出力した PADEL ファイルの行番号である。よって、エラーの時は `mkpad` コマンドを使って PADEL に変換し、その PADEL ファイルを使ってエラーチェックをするようにしないと行番号が一致しない。

3.1　C

C 言語のプログラムから PAD 図を直接描くには、

```
% pad2ps -c foo.c > foo.ps
```

とする。但し、拡張子が `.c` のファイルを変換する場合には `-c` オプションは省略できる。

C 言語のパーサは、**c2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -c foo.c > foo.pad
```

とする。この場合は `-c` オプションは省略できない。

C 言語のプログラムは、各関数ごとに PAD 図を描いていく。各関数名がそれぞれの PAD のタイトルになる。また、関数の外にある大域変数はすべて読み捨てられる。関数内でも、`-O` オプションを指定すると、初期化していない変数宣言は読み捨てることができる。

デフォルトでは、文末の ; (セミコロン) は PAD 図には出力しないが、これは`-semicolon` オプションで出力させることもできる。

C 言語のプログラムを変換する上で最も注意すべきなのは、`#ifdef` などのプリプロセッサによる条件コンパイルを制御する部分である。あるプログラム `foo.c` の PAD 図を描かせるとしよう。この `foo.c` をコンパイルする時、もしも

```
% gcc foo.c -DJAPANESE
```

と、`-DJAPANESE` を指定しているとすれば、**pad2ps** シリーズで PA D 図を描かせる時も、

```
% pad2ps foo.c -D JAPANESE > foo.ps
```

のように、-D オプションを使って定義してやる必要があるだろう。この時、指定形式が C コンパイラとは少し違うので注意して欲しい。

ところが、C コンパイラはわざわざ指定しなくても、その環境や機械ごとにデフォルトで定義するものがいくつかある。また、**pad2ps** シリーズは　#include <stdio.h>　などを無視するので、ヘッダファイル内で定義されているものも見落としてしまう。そこで、定義すべきシンボルはすべて -D オプションで指定してやるべきである。その際、-ifdef オプションを使って、

```
% pad2ps foo.c -ifdef
```

としてやれば、定義することでプログラムが変化するシンボルの一覧を出力してくれるので、その中から適当に定義してやればよい。

C 言語のプログラムは、ほとんどのものの PAD 図を描かせることができるだろう。現在変換できないのは、マクロを使っていて見かけ上 C 言語の文法に合わなくなってしまった場合のみである。特に

```
#define NUMBER(X)  { int a; for( a='0' ; a<=X ; a++) putchar(a);}
    ⋮
puts("Print Numbers");
NUMBER('9')
putchar('\n');
```

のように、; (セミコロン) で終了していなくてエラー、という場合が多い。そのような時は、仕方ないので手で修正してやるしかない。エラーにならなくても、たまたまうまくいっただけで、PAD 図の形が変になっている、という可能性もある。結局、マクロを大いに利用しているプログラムを変換する場合には、よく注意するしかない。

3.2 C++

C++のプログラムから PAD 図を直接描くには、

```
% pad2ps -c++ foo.C > foo.ps
```

とする。但し、拡張子が .C, .cc, .cpp のファイルを変換する場合には -c++ オプションは省略できる。

C++のパーサは、**c2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -c++ foo.C > foo.pad
```

とする。この場合は -c++ オプションは省略できない。

C++のプログラムを変換する上での注意は、C の場合と同じであるので、3.1 節を参照してほしい。

C++の場合、クラスの定義部は読み捨てられる。よって、デフォルトではクラス定義の内部に書き込んだインライン関数も一緒に読み捨てられてしまう。インライン関数の PAD 図も描くためには、-inline オプションを使う。その場合、インライン関数の関数名にそのクラス名を自動的に補って出力する。

3.3 Bourne shell

Bourne shell のプログラムから PAD 図を直接描くには、

```
% pad2ps -sh foo.sh > foo.ps
```

とする。但し、拡張子が `.sh` のファイルを変換する場合には `-sh` オプションは省略できる。

Bourne shell のパーサは、**sh2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -sh foo.sh > foo.pad
```

とする。この場合は `-sh` オプションは省略できない。

Bourne shell のプログラムは一般に 1 つの PAD として描かれ、その PAD にはタイトルはつかない。但し、`function` を使って関数を定義している場合は、全体のプログラムの PAD を描いた後で、それらの関数の PAD を描いていく。その場合、各関数名がそれぞれの PAD のタイトルになる。

Bourne shell のプログラムでは、

```
cat << EOF
    ⋮
EOF
```

のような書き方をすることがある。この場合は、`cat << EOF` と `EOF` で囲まれた出力される部分全体が四角の枠に囲まれて、1 つの PAD ボックスとして描かれる。

Bourne shell のプログラムは、ほぼ完全に PAD 図を描くことができる。

3.4 C shell

C shell のプログラムから PAD 図を直接描くには、

```
% pad2ps -csh foo.csh > foo.ps
```

とする。但し、拡張子が `.csh` のファイルを変換する場合には `-csh` オプションは省略できる。

C shell のパーサは、**csh2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -csh foo.csh > foo.pad
```

とする。この場合は `-csh` オプションは省略できない。

C shell のプログラムは一般に 1 つの PAD として描かれ、その PAD にはタイトルはつかない。

C shell のプログラムでは、

```
cat << EOF
    ⋮
EOF
```

のような書き方をすることがある。この場合は、`cat << EOF` と `EOF` で囲まれた出力される部分全体が四角の枠に囲まれて、1 つの PAD ボックスとして描かれる。

C shell のプログラムは、ほぼ完全に PAD 図を描くことができる。

3.5 Java

Java のプログラムから PAD 図を直接描くには、

```
% pad2ps -java foo.java > foo.ps
```

とする。但し、拡張子が .java のファイルを変換する場合には -java オプションは省略できる。

Java のパーサは、**java2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -java foo.java > foo.pad
```

とする。この場合は -java オプションは省略できない。

Java のプログラムは、各メンバ関数ごとに PAD 図を描いていく。各関数名がそれぞれの PAD のタイトルになる。

-O オプションを指定すると、アルゴリズムに関係ない部分を出力しないようにする。具体的には、初期化していない変数宣言を読み捨てる。

デフォルトでは、文末の ; (セミコロン) は PAD 図には出力しないが、これは-semicolon オプションで出力させることもできる。

Java のプログラムは、ほぼ完全に PAD 図を描くことができる。

3.6 AWK

AWK のプログラムから PAD 図を直接描くには、

```
% pad2ps -awk foo.awk > foo.ps
```

とする。但し、拡張子が .awk のファイルを変換する場合には -awk オプションは省略できる。

AWK のパーサは、**awk2pad** である。これを直接起動してプログラムを PADEL に変換するには、

```
% mkpad -awk foo.awk > foo.pad
```

とする。この場合は -awk オプションは省略できない。

AWK のプログラムは、BEGIN、END、または各パターンごとに PAD 図を描いていく。BEGIN、END、または各パターンがそれぞれの PAD のタイトルになる。

AWK のプログラムは、ほぼ完全に PAD 図を描くことができる。

3.7 その他の言語

残念ながら、バージョン 3.1j ではこれ以外の言語には対応していない。だが、今後さらに対応する言語を増やしていく予定である。現在候補に挙がっているのは、Tcl/Tk, Pascal, Fortran, Perl 等である。これらは比較的 C 言語と構造が似ているので、次のバージョンで対応する可能性がある。

ところで、すべてのプログラムを変換できなくても、たいていのプログラムを変換する程度であれば、sed や awk などの文字列処理言語を使って、比較的簡単に変換ツールが作れるだろう。**pad2ps** シリーズの作者は主に C を使ってプログラムを組んでいるので、他の言語を使っている諸氏は、変換ツールを自作してみるのもいいだろう。もしできたら、作者に送って頂けると幸いである。

4　パラメータ変数について

パラメータ変数は、PAD 図の体裁を整えたり、装飾を施したりして、きれいに仕上げるためのコマンドである。

パラメータ変数は PADEL のコメントとして記述するか、

```
/* pad2ps: centering = on */
```

またはコマンドラインから -P オプションで指定する。

```
% pad2ps foo.pad -P centering=on > foo.ps
```

一般にパラメータ変数は指定する場所とは無関係だが、**message**、**comment**、**newpage**、**vsp** の 4 つは指定する場所に依存したパラメータである。

LaTeX 形式で出力させた場合、有効にならないパラメータ変数もある。

4.1　総合タイトルをつけたい

パラメータ **title** を使って、総合タイトルをつけることができる。次のように、引数にタイトルの文字列を指定する。

```
/* pad2ps: title = This is a pad title. */
```

デフォルトでは、ファイル名を指定して起動した場合はファイル名がタイトルとなり、ファイル名を指定しなかった場合はタイトルはつかないことになっている。

4.2　メッセージを書く

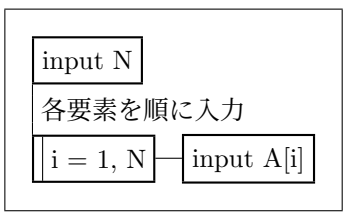
パラメータ **message** を使って、PAD 図の中にメッセージを書くことができる。

これは場所に依存するパラメータである。メッセージを書きたい場所で **message** を使う。引数には書く内容を指定する。例えば、

```
{
    input N

    /* pad2ps: message = 各要素を順に入力 */
    for(i = 1, N)
        input A[i]
}
```

と書くと、



という PAD 図になる。

メッセージはコメントやパラメータというよりも、ふつうの文が枠で囲まれていないだけ、と考えたほうがよい。つまり、メッセージを書くことができるのは、ふつうの文を書くことのできる場所に限るし、また、メッセージが書いてあればそれだけで文が 1 つ書いてあるのと同等になる。よって、

```
/* pad2ps: message = 符号によって 1 を足す、引く */

{
    for(i = 1, N){
        if( A[i] > 0 )
            /* pad2ps: message = 正ならば 1 を足す */
            m = m + 1
        else
            /* pad2ps: message = 0 か負ならば 1 を引く */
            m = m - 1
    }
}
```

は不正で、以下のように書かなければいけない。

```
{
    /* pad2ps: message = 符号によって 1 を足す、引く */

    for(i = 1, N){
        if( A[i] > 0 ) {
            /* pad2ps: message = 正ならば 1 を足す */
            m = m + 1
        } else {
            /* pad2ps: message = 0 か負ならば 1 を引く */
            m = m - 1
        }
    }
}
```

4.3 コメントを書く

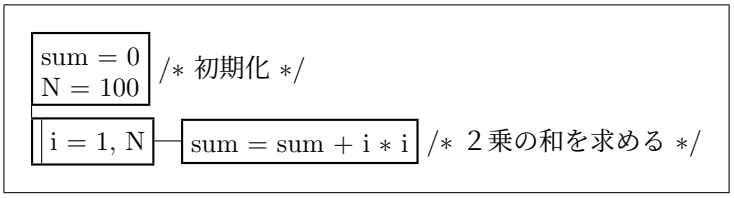
コメントをつけたい PADEL 命令の次の行で、パラメータ **comment** を使ってコメントの内容を指定する。ここでいうコメントとは、2.9 節で述べたような PADEL ファイルの中のコメントではなく、出力される PAD 図の中のコメントである。

コメントは対象となる PAD ボックスの右側に書かれる。例えば、

```
{
    {
        sum = 0
        N = 100
    }
    // pad2ps: comment = /* 初期化 */

    for(i = 1, N)
        sum = sum + i * i
        // pad2ps: comment = /* 2 乗の和を求める */
}
```

と書くと、



という PAD 図になる。

コメントを書ける場所は、対象となっている文またはブロックの次の行のみである。その文またはブロックとコメントの間には、空行も入れてはいけない。これは、コメントがメッセージと違って、独立した 1 つの文とはみなされず、対象となる文やブロックに付属するものとして扱われているからである。よって、上記の例では、ループを { } (ブレース) で囲まなくても構わない。

また、ループに対してコメントをつける時は、上記の例のように、ネストの一番内側の、繰り返される命令の次の行に書く。もしも

```
for(i = 1, N)
/* pad2ps: comment = 2 乗の和を求める */
{
    sum = sum + i * i
}
```

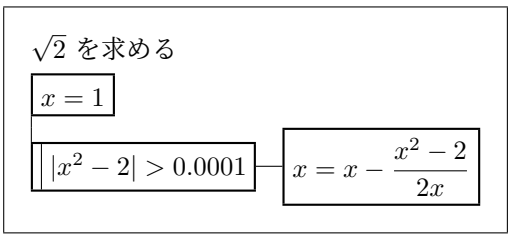
と書いた場合はエラーとなる。

4.4 PADEL に LaTeX の命令を書く

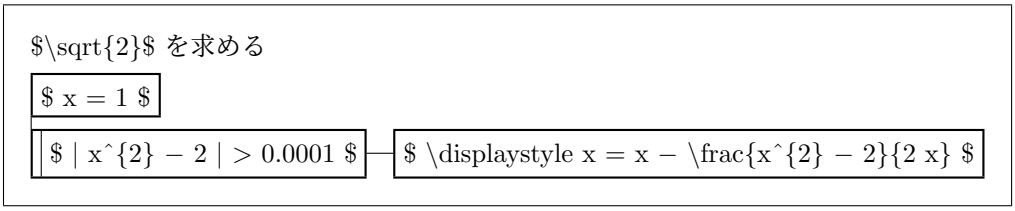
pad2tex で LaTeX 形式に変換する場合、パラメータ変数 **texmode** を latex または verbatim のいずれかに設定することで、PADEL に書かれた文字列を LaTeX の命令として LaTeX に処理させるか、PADEL に書いたとおりに出力するかを選択できる。デフォルトは verbatim で、PADEL の文字がそのまま PAD 図に出力される。latex を指定した場合、

```
/* pad2ps: texmode = latex */
{
    /* pad2ps: message = $\sqrt{2}$ を求める */
    $ x = 1 $
    while ( $ | x^{2} - 2 | > 0.0001 $) {
        $ \displaystyle x = x - \frac{x^{2} - 2}{2 x} $
    }
}
```

と書いて、次の PAD 図を描くことができる。



逆に、verbatim を指定した場合の結果は次のようになる。



4.5 中央、または左端に揃える

パラメータ **centering** を on または off にすることによって、基準線を中央とするか左端とするかを選択できる。on の時は中央に揃えて出力し、off の時は左に揃えて出力する。デフォルトは on である。

4.6 PAD 図の配置形式を変更する

パラメータ **locate** を topdown、bottomup、middle のいずれかに設定することで、ループや条件分岐の時、どこで高さを合わせるかの選択ができる。デフォルトは middle である。

4.7 ページ番号を出力する

ページ番号を出力するには、パラメータ **page** に開始ページを指定する。デフォルトではページ番号は出力しないようになっているので、たとえ 1 ページ目から順に番号を振る場合でも、出力させるためには 1 を指定する必要がある。

4.8 図番号を出力する

図番号を出力するには、パラメータ **figure** に開始番号を指定する。ページ番号と同様に、デフォルトでは番号がつかない。

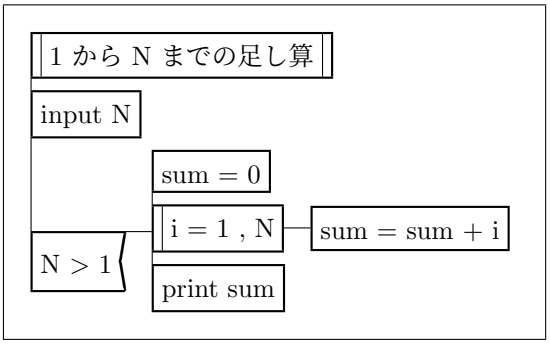
4.9 図題の出力位置を変更する

パラメータ **figure_locate** を top、bottom のいずれかに設定することによって、各 PAD 図のタイトルや図題を PAD 図の上に書くか下に書くかを選択できる。top なら PAD 図の上に、bottom なら下に図題を出力する。デフォルトでは、図番号を出力する場合は bottom、図番号なしでタイトルのみの場合は top である。

4.10 各 PAD 図のタイトルの出力形式を変更する

パラメータ **titlebox** を on または off にすることによって、各 PAD 図のタイトルの出力形式を選択できる。off の時は、各 PAD 図のタイトル (C 言語のプログラムなら関数名) を各 PA D 図の図題として扱い、on の時は、PAD 図のタイトルを左右が二重線になった PAD ボックスとして、PAD の中に描く。デフォルトは off である。

titlebox を on にした時の出力結果は、次のようになる。



4.11 ページに枠をつけたい

パラメータ **pageframe** を on にすることで、ページ全体の枠を描くことができる。デフォルトでは off になっている。

4.12 各 PAD 図を枠で囲みたい

パラメータ **padframe** を on にすることで、各 PAD 図を囲む枠を描くことができる。デフォルトでは on になっているので、この枠が気に入らない場合は off に設定する。

尚、この枠はパラメータ **centering** が on の時は各 PAD 図の大きさに合わせた大きさで描かれるが、off の時は横幅はページ幅と同じになる。

4.13 各 PAD 図をページの先頭から描き始めたい

パラメータ **slice** を on にすることで、各 PAD ごとに改ページして、PAD 図をページの先頭から描き始めることができる。この結果、1 ページに 1 つの PAD が描かれることになる。

デフォルトでは off になっているので、すべての PAD 図が連続して描き並べられる。

4.14 全体を拡大・縮小したい

pad2ps シリーズでは、PAD 図を縦横ともに自在に拡大・縮小することができる。横方向の倍率は **hrate**、縦方向の倍率は **vrate** で設定する。小数で指定可能である。デフォルトはそれぞれ 1.0 である。

但し、**hrate** をあまり大きく設定すると、折り曲げても入り切らなくなってしまい、自動で **hrate** が再設定されてしまう。逆に、できるだけ大きく拡大したい時には、大きな値を設定しておけば、限界まで拡大させることができる。もちろんパラメータ **poster** を on にして、折り曲げをしないようにした場合は、いくらでも大きく広げることができる。

一方、**vrate** の方はいくらでも大きく設定できる。

4.15 無理に折り曲げずに、大きな PAD 図を描きたい

pad2ps シリーズでは、1 枚の紙に入り切らないような大きな PAD 図の場合、各 PAD ボックスを折り曲げて、なんとか 1 枚の紙に収まるように調節して出力する。それでも入り切らない場合

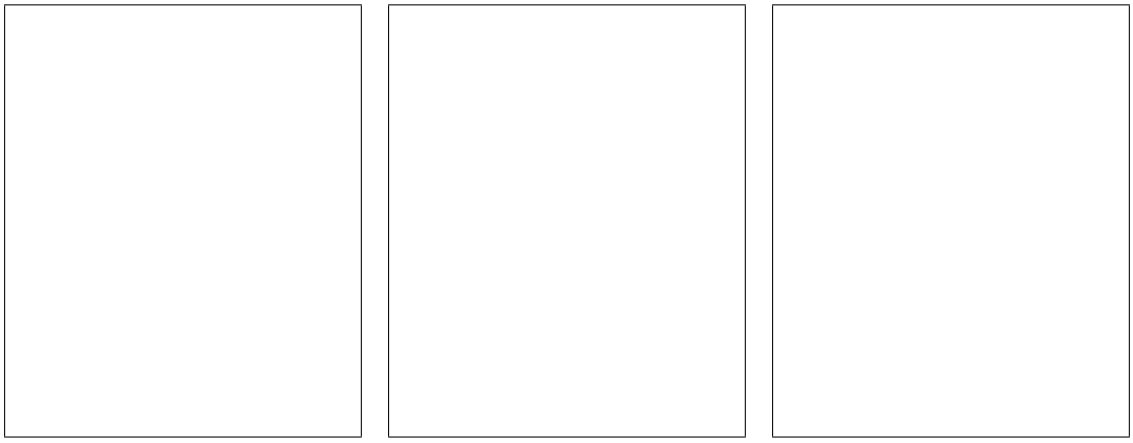


図 3: アッカーマン関数の PAD 図のポスター

は、パラメータ **hrate** を自動設定し、縮小して 1 枚の紙に収める。

もしも貴方がこの結果を好まず、横方向にも複数の紙に渡るような、大きな PA D 図を描きたいと思うのならば、パラメータ **poster** を on にする。この場合は、できた PostScript を印刷して、床の上に並べて見るとよいだろう。

もしも PAD 図のポスターのようなものを作りたいのならば、**poster** を on に設定しておいて、**hrate**、**vrate** を使って拡大してやると良い。

例として、アッカーマン関数の PAD 図を 3 倍の拡大率で描いてみた (図 3)。元のファイルは `ack.pad` である。**poster** を on にし、できた PAD 図をこのように横に並べれば、大きな PAD 図を得ることができる。

4.16 PAD を折り曲げる幅を指定したい

pad2ps シリーズは、PAD 図が横にはみだす場合、各 PAD ボックスを折り曲げて、1 枚の紙に収まるように調節する。その幅は、**pad2ps** シリーズが適当に計算してくれるので、ふつうは我々は関知する必要はない。

しかし、場合によっては PAD の文字列の長さに制限をもたせたい時もあるかもしれない。また、PAD 図があまり横に広がっていると、折り曲げた結果、ほとんどの PA D ボックスが縦長になりすぎて、かえって見にくくなる場合がある。このような時には、文字列の長さの下限を設定しておいて、折り曲げすぎを防ぐ必要があろう。

このように、文字列の長さの最大、最小を設定するには、パラメータ **max_length**、**min_length** を使用する。デフォルトでは、それぞれ無限大、0 になっている。これらのパラメータを設定すると、**pad2ps** シリーズはその条件を満たす範囲で折り曲げを行うようになる。

4.17 改ページをする

改ページするには、改ページしたい場所でパラメータ **newpage** を指定する。これは引数を持たないパラメータである。

4.18 PAD ボックスどうしの縦の間隔を調節する

PAD ボックスどうしは適当な間隔をとって描かれるが、場合によってはこれを変更したいことがあるかもしれない。また、ページの境に PAD ボックスがかかってしまった場合も、間隔を手動で調節し、ボックスが 2 ページにまたがらないようにした方がよいだろう。このような問題は、たいていは空行を使って間隔を空ければ解決するのだが、場合によっては、もっと微妙な調節をしたい時もあるだろう。そのような時には、パラメータ **vsp** を使う。また、最初の PAD 図を描くまで、つまり PADEL の先頭にある空行だけは無視されてしまうので、そこではこの **vsp** を使うしかない。

vsp には、引数としてあけるスペースの大きさを指定する。ちなみに空行は、デフォルトでは PAD の外部では 400、内部では 120 という大きさのスペースを空けている。また、**vsp** の引数には負の数を指定することもできるので、PAD ボックスの間隔をつめたり、間隔をなくしたり、上の PAD と重ねたりすることもできる。間隔を 0 にするには、引数に −120 を指定する。それ以下の数を指定すると、上の PAD と重なってしまう。

4.19 ページの大きさを変更する

表のようなパラメータ変数によって、ページの上下左右の端の座標を指定することが可能である。これによってページの大きさが変更できる。

パラメータ	意味	範囲	デフォルト
page_top	上端	0 − 10000	9800
page_bottom	下端	0 − 10000	0
page_left	左端	0 − 9000	500
page_right	右端	0 − 9000	8500

このうち、**page_left**、**page_right** は **centering** を off に設定している時のみ有効である。on の時は無視される。また、**page_bottom** を指定しても、ページ番号の出力位置は変更できない。

4.20 フォントサイズを変更する

4.20.1 ふつうのフォントのサイズを変更する

フォントサイズを変更する際には、高さを **fontsize** で、幅を **fontwidth** でそれぞれ指定する。デフォルトは 200、100 である。

日本語フォントの場合、高さは英数字と同じで **fontsize** であるが、幅は **jfontwidth** で別に指定する。デフォルトは 200 である。

4.20.2 特別なフォントのサイズを変更する

総合タイトル、各 PAD 図のタイトル、及び各 PAD 図の図題には、ふつうのフォントより少しおおきなフォントが使われる。その大きさは、表のようなパラメータに、ふつうのフォントに対する倍率を設定することで変更できる。

パラメータ	対象	デフォルト
title_rate	総合タイトル	2.0
padtitle_rate	各 PAD 図のタイトル	1.5
figure_rate	図題	1.5

padtitle_rate と **figure_rate** との違いは、パラメータ **figure** を指定せず、図番号が出力されない時が前者、図番号も出力するようにした場合が後者、ということである。

4.21 スペースの大きさを変更する

PAD ボックスの内部や、PAD どうしの間など、スペースを空ける時には表のような変数の大きさのスペースがとられている。

パラメータ	デフォルト
pad_vspace	400
vspace	200
hspace	200
if_vspace	50
box_hspace	125
if_rightspace	200
sideline_space	50
padframe_vspace	180
padframe_hspace	300

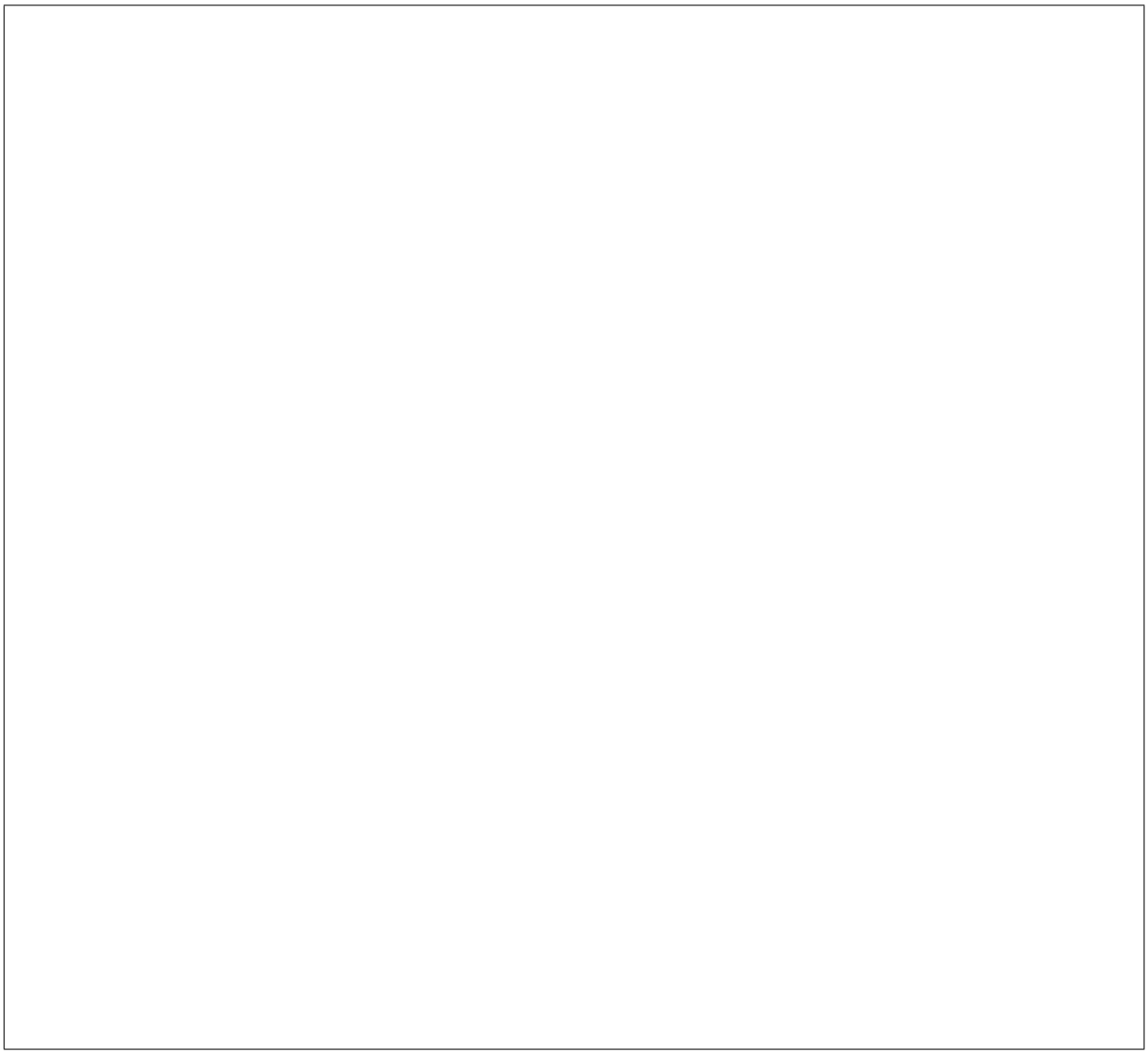
4.22 線幅を変更する

PAD 図を描く時の線の幅も、表のようなパラメータ変数を設定することで変更可能である。

パラメータ	対象	デフォルト
linewidth	PAD ボックスどうしをつなぐ線	16
padlinewidth	PAD ボックスを描く線	16
framelinewidth	PAD 図を囲む枠、ページ枠の線	8

A　pad2ps シリーズのモジュール体系

pad2ps シリーズはたくさんのモジュールから構成されていて、それぞれが作業を分担して一連の処理を行なっている。これらの各モジュールの体系を図で示しておこう。



索引