

超入門 Cfep/asir (MacOS X)

高山信毅

2006年(平成18年), 3月12日版 (cfep 1.1). 2008-09-26, 2009-09-19, 2012-08-28 修正
コメントは takayama@math.kobe-u.ac.jp まで

目次

第 1 章	電卓としての利用	5
1.1	キー操作と用語の復習	5
1.2	Cfep/Asir の起動法と電卓的な使い方	6
1.3	エラーメッセージ	11
第 2 章	変数とプログラム	15
2.1	変数	15
2.2	くりかえし	18
2.3	実行の中止	20
2.4	エンジン再起動	21
2.5	ヘルプの利用	22
第 3 章	グラフィック	25
3.1	ライブラリの読み込み	25
3.2	線を引く関数	25
3.3	円を描く関数を作ってみよう	28
第 4 章	For 文による数列の計算	33
4.1	超入門, 第 2 の関門: 漸化式でできる数列の計算	33
4.2	円を描く数列	34
第 5 章	cfep 上級編	37
5.1	T _E X によるタイプセット (実験的)	37
5.2	選択範囲のみの実行	37
5.3	エンジンを起動しない	38
5.4	OpenGL インタプリタ	39
5.5	asir 以外の計算エンジンの利用	41

第1章 電卓としての利用

神戸大学の教育用計算機環境が MacOS X に変更されるのに伴い、筆者が教材として利用していた Windows で動作する 10 進 Basic が利用できなくなった。Cfep/asir はその代用として、2006 年初頭から開発を進めているシステムである。10 進 Basic の優れている点の一つは、丁寧な入門解説が付属していることである。“Cfep/asir 超入門”はこの解説にすこしでも近付こうと努力してみた。Asir の入門テキストに“Asir ドリル”があるが、この超入門では“Asir ドリル”の一章およびその先の入門の内容を丁寧に (少々くどく) 説明した。

この節では MacOS X での cfep/asir の起動法、電卓風、Basic 風の使い方を説明する。ファイルの保存等 MacOS X の共通の操作方法にはほとんどふれていないが、cfep/asir は MacOS X 標準のファイルの保存等を用いているので、このような部分では他のソフトウェアと利用方法は同一である。初心者の方は適当な本やガイドを参照されたい。

1.1 キー操作と用語の復習

キーボード、マウスの操作の用語。

1. **Command** キーや **ALT** キーや **SHIFT** キーや **CTRL** キーは他のキーと一緒に押すことで始めて機能するキーである。これらだけを単独に押してもなにもおきない。以後 **SHIFT** キーをおしながら他のキーを押す操作を **SHIFT**+**キー** と書くことにする。command キー、alt キー、ctrl キーについても同様である。
2. **SHIFT**+**a** とすると大文字の A を入力できる。
3. **BS** とか **DEL** と書いてあるキーを押すと一文字前を消去できる。
4. 日本語キーボードの場合 **** (バックスラッシュ) は **ALT**+**¥** で入力できる。
5. **SPACE** キーは空白を入力するキーである。計算機の内部では文字は数字に変換されて格納および処理される。文字に対応する数字を文字コードと呼ぶ。文字コードにはいろいろな種類のものがあるが、一番基礎的なのはアスキーコード系であり、アルファベットや数字、キーボードに現れる記号などをカバーしている。漢字はアスキーコード系では表現できない。**A** のアスキーコードは 65 番である。以下 **B** が 66, **C** が 67, と続く。空白のアスキーコードは 32 番である。日本語入力の状態で入力される空白は“全角空白”と呼ばれており、アスキーコード 32 番の空白 (半角空白) とは別の文字である。全角空白がプログラムに混じっているとエラーを起こす。asir ではメッセージやコメント等に日本語が利用可能であるが、慣れるまでは英字モードのみを利用することをお勧めする。右上の言語表示が



となっている状態で cfep/asir に入力しよう。

6. (シングルクオート) と (バッククオート) は別の文字である。プログラムを読む時に注意。また、プログラムを読む時は 0 (ゼロ) と o (おー) の違いにも注意。
7. マウスの操作には次の三種類がある。
 - (a) クリック: 選択するとき, 文字を入力する位置 (キャレットの位置) の移動に用いる。マウスのボタンをちょんとおす。
 - (b) ドラッグ: 移動, サイズの変更, 範囲の指定, コピーのときなどに用いる。マウスのボタンを押しながら動かす。
 - (c) ダブルクリック: プログラムの実行, open(ファイルを開く) をするために用いる。マウスのボタンを2回つづけてちょんちょんとおす。ダブルクリックをしたアイコンは白くなったり形状が変わることがおおい。ダブルクリックしたらしばらく待つ。計算機が忙しいときは起動に時間がかかることもあり。むやみにダブルクリックを繰り返すとその回数だけ起動されてなお遅くなる。

1.2 Cfep/Asir の起動法と電卓的な使い方

cfep のアイコン (いのぶた君)



をダブルクリックすると図 1.1 のように cfep/asir が起動する。以下 cfep/asir を単に asir とよぶ。

図 1.1 の入力窓に計算したい式やプログラムを入力して “始め” ボタン



をおすと実行を開始する。式の計算やプログラムの実行が終了すると, 新しいウインドウ OutputView が開き結果がそのウインドウに表示される。“始め” ボタンをおして実行を開始することを計算機用語では “入力の評価を始める” という。

出力小窓にはシステムからのいろいろな情報が出力されるが, 内容は中上級者向けのものが多い。ファイルメニュー



図 1.1: cfep/asir の起動画面



から”保存”や”別名で保存”を実行すると入力窓の内容をファイルとして保存できる。出力小窓の内容や OutputView の内容は保存されないので注意してほしい。

cfep/asir を完全に終了するには cfep メニュー



の“cfep を終了”を実行する。

さて図 1.1 では $3 \times 4 + 1$ の計算をしている。

Asir における計算式は普通の数式と似ていて、足し算は $+$ 、引き算は $-$ と書く。かけ算と割算は \times や \div がキーボードにないという歴史的理由もあり、それぞれ $*$ と $/$ で表現する。累乗 P^N は P^N のように $^$ 記号を用いて表す。

式の終りを処理系 (asir) に教える (示す) のに ; (セミコロン) を書かないといけない。文末の “.” のような役割を果たす。またかけ算の記号 $*$ の省略はできない。

例 1.1 以下の左の計算式を asir では右のようにあらわす。

$2 \times (3 + 5^4)$	$2*(3+5^4);$
$\{(2 + \frac{2}{3}) \times 4 + \frac{1}{3}\} \times 2 + 5$	$((2+2/3)*4+1/3)*2+5;$
$AX + B$	$A*X+B;$
$AX^2 + BX + C$	$A*X^2+B*X+C;$
$\frac{1}{X-1}$	$1/(X-1);$

計算の順序は括弧も含めて普通の数式の計算と同じである。ただし数学ではかっことして, [,], {, } などがつかえるが asir では (,) のみ。[,] や {, } は別の意味をもつ。上の例のように (,) を何重にもつかってよい。この場合括弧の対応関係がわかりにくい。括弧の対応を調べたい範囲をマウスでドラッグして選択し、



ボタンをおすことにより括弧の対応を調べることができる。図 1.2 の例では $(1+2*(3+4))$ と書くべきところを $(1+2*(3+4)$ と書いておりエラーが表示されている。

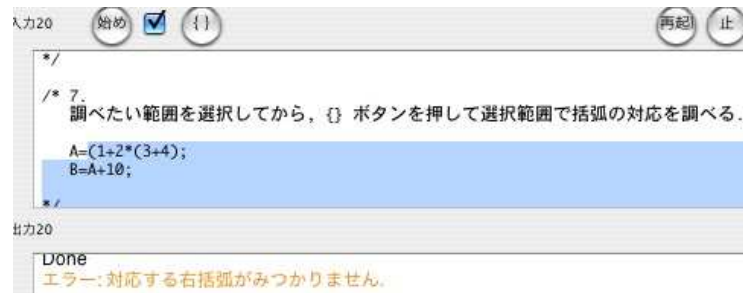


図 1.2: 括弧の対応

質問 “Basic 風の使い方を説明する” と書いてありましたが, Basic って何ですか?

答え コンピュータに仕事をさせるには最終的にはプログラム言語 (計算機への仕事の手順を指示するための人工言語) を用いる. ワープロ等もプログラム言語で記述されている. Basic は最も古いプログラム言語の一つであり, 初心者にはやさしく, かつ計算機の仕組みやプログラム言語の理解にも有用である. Basic は高校の数学の教科書等にも登場する. 著者はいままで “10 進 BASIC” を初心者向け教材として活用していたが, “10 進 BASIC” が MacOS X で動作しないため cfep を開発した. Asir 言語もプログラム言語であり Basic とよく似ているが, C 言語にもっと近い.

質問 MacOS X って何ですか?

答え —まだ書いてない.

Asir は数の処理のみならず, \sqrt{x} や三角関数の近似計算, 多項式の計算もできる. 左の数学的な式は asir では右のように表す.

π (円周率)	@pi
$\cos x$	cos(x)
$\sin x$	sin(x)
$\tan x$	tan(x)
\sqrt{x}	x^(1/2)

三角関数の角度にあたる部分の x はラジアンという単位を用いて表す. 高校低学年の数学では角度を度 (degree) という単位を用いて表すが, 数学 3 以上では角度はラジアンという単位で表す.

90 度 (直角) が $\pi/2$ ラジアン, 180 度が π ラジアン. 一般に d 度は $\frac{d}{180}\pi$ ラジアンである.

単位ラジアンをもちいると微分法の公式が簡潔になる. たとえば x がラジアンであると $\sin x$ の微分は $\cos x$ である.

$\sin(x)$ や $\cos(x)$ の近似値を求めるにはたとえば

```
deval(sin(3.14));
```

と入力する. これは $\sin(3.14)$ の近似値を計算する. $\sin \pi = 0$ なので 0 に近い値が出力されるはずである. 実際 0.00159265 を出力する. deval (evaluate and get a result in double number precision の略) は 64 bit の浮動小数点数により近似値計算する. 64 bit の浮動小数点数とは何かの説明は超入門の範囲外であるが, 計算機は有限の記憶領域 (メモリ) しか持たないので, 小数も有限桁しか扱えないと覚えておこう. 64bit は扱える桁数を表している. 詳しくは “asir ドリル” を参照して欲しい.

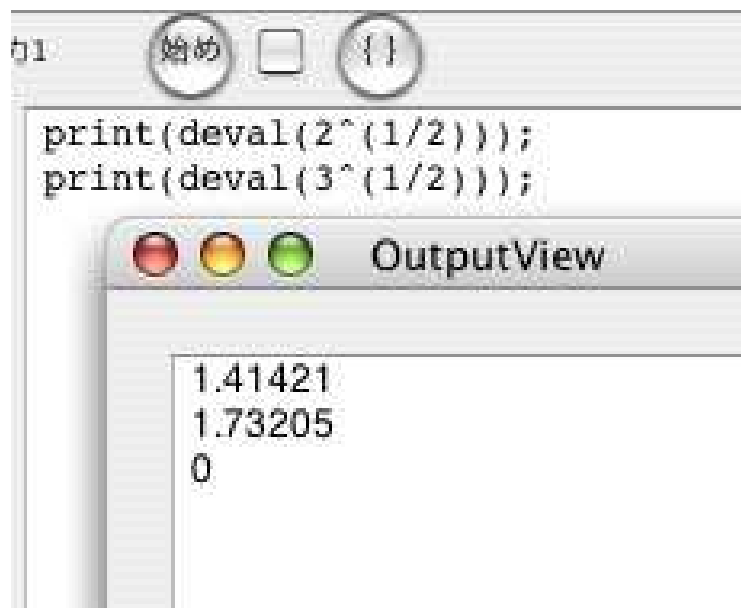


図 1.3: 平方根の計算

例 1.2 $\sqrt{2}$, $\sqrt{3}$ の近似値を計算しなさい.

入力

```
print(deval(2^(1/2)));
print(deval(3^(1/2)));
```

出力は図 1.3 をみよ.

上の例のように、セミコロン ; で区切られた一連の命令のあつまりはもっとも単純な asir プログラムの例である。一連の命令は始めから順番に実行される。print(式等); は“式等”の値を計算して値を画面に表示する。

さて出力の 1.41421 (ひとよひとよにひとみごろ) は $\sqrt{2}$ の近似値なので、print(deval(2^(1/2))); の実行結果である。さて出力の 1.73205 (ひとなみに おごれや) は $\sqrt{3}$ の近似値なので、print(deval(3^(1/2))); の実行結果である。最後の 0 はなんなのであろうか? 実はこれは最後の print 文の戻している値である。むつかしい? 別の例で説明しよう。

入力

```
1+2;
2+3;
3+4;
```

この時出力は (OutputView への表示は)

```
7
```

となる。cfep/asir ではとくに print 文をかかない限り最後の文の計算結果 (評価結果) しか出力しない。いまの場合は $3 + 4$ の結果 7 を出力している。

問題 1.1 1. 2^8 , 2^9 , 2^{10} , の値を計算して答えを表示するプログラムを書きなさい。

2. 2 の累乗はパソコンの性能説明によく登場する. たとえば検索システム google にキーワード “512 メモリ 搭載” を入力したところ “ビデオメモリを 256M から 512M に倍増させ” など, 数多くの記事がヒットする. このような記事を (意味がわからなくても)10 件あつめてみよう. 512 以外の 2 の累乗でも同じことを試してみよう.

3. (中級) 2 の累乗がパソコンの性能説明によく登場する理由を論じなさい.

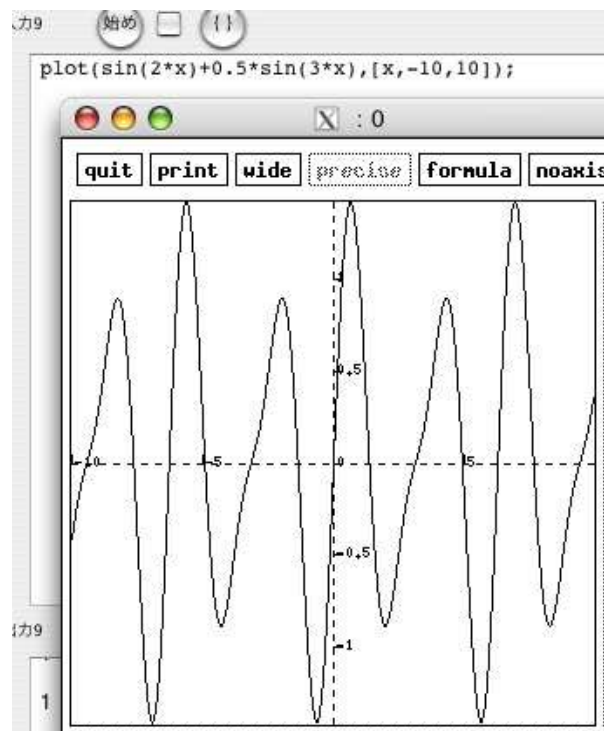


図 1.4: 関数のグラフ

発展学習 X11 環境が動いていれば, `plot(f);` 命令で x の関数 f のグラフを描ける. x の範囲を指定したいときはたとえば

`plot(f, [x, 0, 10])` と入力すると, x は 0 から 10 まで変化する.

入力例

```
plot(sin(x));
plot(sin(2*x)+0.5*sin(3*x), [x, -10, 10]);
```

問題 1.2 いろいろな関数のグラフを描いてあそんでみよう. 数学の知識を総動員して計算機の描く形がどうしてそうなのか説明を試してみよう.

1.3 エラーメッセージ

入力にエラーがあると, エラーメッセージが表示される.



図 1.5: 文法エラー

図 1.5 では `2+4=` と入力している。最後に `=` を書く表現は `asir` の文法では許されていないので、“文法エラー”と指摘されている。

大体これでわかってきていいじゃない、とこちらがおもっていてもプログラム言語は一切融通がきかない。

なお

```
error([41,4294967295,parse error,[asir_where,[[toplevel,1]]]])
```

の部分は上級者向けの情報なのでとりあえず無視してもらいたい。



図 1.6: エラー行

図 1.6 では

```
print( 2^7 );
print( 2^8 );
print( deval(2^(1/2)));
print( deval(3^(1/2)));
```

と入力している。3行目は右括弧がひとつ足りなくて `print(deval(2^(1/2)));` が正しい入力である。エラー行の3行目にカーレットが自動的に移動しているはずである。なおプログラムの入力ウイ

ンドー内でマウスをクリックすると、せっかく自動移動したキャレットの位置が変わってしまう。プログラムの入力ウインドーのタイトルバーをクリックするとよい。なおこの例では



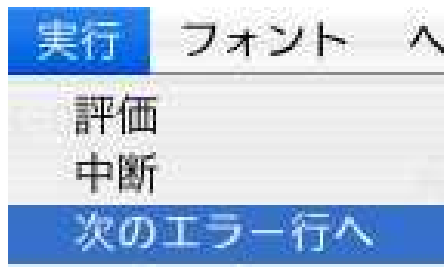
ボタンをもちいてもすぐエラーの場所がわかる。

注意: 表示された行はエラーの発生位置であるが、エラーの原因はその前の方の行にあることも多い。たとえば

```
1+2  
2+3;
```

と入力するとエラー行は 2 行目であるが、原因は 1 行目で ; を書き忘れたことである。

エラー行が複数表示された場合はそれらの中のどこかにエラーがある。複数あるエラー行に順番にジャンプしていくには、**実行** メニューから **次のエラー行へ** を選択する。



問題 1.3 エラーを生じる式またはプログラムを 5 つ作れ。

第2章 変数とプログラム

2.1 変数

変数に数値等を記憶しておける。変数名は大文字で始まる。なお後述するように asir では多項式計算ができるが小文字で始まる文字列は多項式の変数名として利用される。

2 の累乗を表示する次のプログラムを考えよう。

```
print( 2^1 );
print( 2^2 );
print( 2^3 );
print( 2^4 );
print( 2^5 );
print( 2^6 );
print( 2^7 );
print( 2^8 );
```

このプログラムは変数 X を用いて次のように書いておけば 2 の累乗だけでなく 3 の累乗を表示するのに再利用できる (図 2.1)。

```
X = 2;
print( X^1 );
print( X^2 );
print( X^3 );
print( X^4 );
print( X^5 );
print( X^6 );
print( X^7 );
print( X^8 );
```

3 の累乗を表示するには $X=2$ の行を $X=3$ に変更すればいいだけである。

アルファベットの 大文字 ではじまる英数字の列が asir の変数である。つまり, X, Y, Z はもちろんのこと, Sum とか $Kazu$ とか $X1$ など 2 文字以上の英数字の列の組み合わせが変数名として許される。

変数を含んだ式をプログラム中で自由につかうこともできる。たとえば

```
X = 2;
A = 1;
print( 2*X^2 -A );
```

を実行すると 7 が表示される。

```

カ3
X = 2;
print( X^1 );
print( X^2 );
print( X^3 );
print( X^4 );
print( X^5 );
print( X^6 );
print( X^7 );
print( X^8 );

```

```

2
4
8
16
32
64
128
256
0

```

図 2.1: 変数の利用

このような例をみると、変数の機能は中学数学でならう文字式と似ていると思うだろう。超入門としてはこれでほぼ正しい理解であるが、よりステップアップしていくには、次のことを強く記憶しておこう。

変数とは計算機に数値等を保存しておくメモリ上の場所の名前である。

さて、超入門、第一の関門である。

= 記号は次のような形式でつかう:

変数名 = 式;

これはまず右辺の式を計算しそのあとその計算結果を左辺の変数に代入せよという意味。= 記号は右辺を計算してその結果を左辺へ代入せよという 命令 だと思って欲しい。

たとえば、 $x=1$ は x が 1 に等しいという意味ではなく、1 を変数 x に代入せよという意味である。

ここでいいたいことは、

= 記号の意味が数学での意味と違うよ!

ということである。これで混乱する入門者も多いのでプログラム言語によっては“2 を変数 x に代入せよ”を $x:=2$ と書く場合もある (たとえばプログラム言語 Pascal)。

次のプログラムは $2, 2^2, 2^4, 2^8$ を計算して表示する。


```

X=2;
print(X);
X = X*X;
print(X);
X = X*X;
print(X);
X = X*X;
print(X);

```

出力が図 2.2 のようになる理由を説明しよう。まず 1 行目で変数 X に 2 が代入される。次に 3 行目ではま

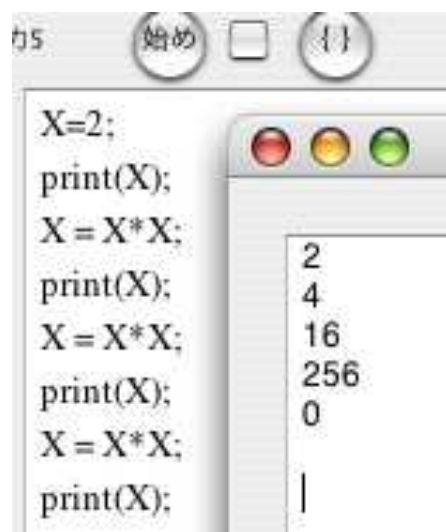


図 2.2: 変数の利用

ず右辺の式を計算する。この場合 X の値は 2 であるので、 2×2 で結果は 4 である。この計算が終わった後結果の 4 が変数 X に代入される。5 行目では右辺の式は 4×4 なので、その計算結果の 16 が左辺の変数 X に代入される。

発展学習 Asir は多項式計算もできる。実は Asir は計算機で記号的に数式を処理するための数式処理システムでもある。

1. 小文字ではじまる記号は多項式の変数である。数学とちがって変数の名前は一文字とはかぎらない。たとえば $rate$ と書くと、 $rate$ という名前の多項式の変数となる。たとえば $x2$ と書くと、 $x2$ という名前の多項式の変数となる。 x かける 2 は $x*2$ と書く。
2. $fctr(poly)$ は $poly$ を有理数係数の範囲で因数分解する。 $fctr$ は $factor$ の短縮表現である。

図 2.3 の $fctr$ の出力の最初は $x^2 + 2xy + y^2$ が $1^1 \times (x + y)^2$ と因数分解されることを意味している。図 2.3 の $fctr$ の出力の 2 番目は $x^2 - 1$ が

$$1^1 \times (x - 1)^1 \times (x + 1)^1$$

と因数分解されることを意味している。

```

カ1  始め □ {}
print( fctr(x^2+2*x*y+y^2) );
print( fctr(x^2-1) );
print( fctr(x^3-1) );
print( fctr(x^4-1) );
fctr(x^5-1);
Output
[[1,1],[x+y,2]]
[[1,1],[x-1,1],[x+1,1]]
[[1,1],[x-1,1],[x^2+x+1,1]]
[[1,1],[x-1,1],[x+1,1],[x^2+1,1]]
[[1,1],[x-1,1],[x^4+x^3+x^2+x+1,1]]

```

図 2.3: 因数分解

2.2 くりかえし

くりかえしや判断をおこなうための文が asir には用意されている. この文をもちいると複雑なことを実行できる. まず一番の基礎であるくりかえしの機能をためしてみよう.

例 2.1 図 2.1 のプログラムは次のように繰り返し機能 — for 文 — を用いて簡潔に書ける.

```

X = 2;
for (I=1; I<=8; I++) {
    print( X^I );
}

```

実行結果は図 2.4 をみよ.

```

カ2  始め □ {}
X = 2;
for (I=1; I<=8; I++) {
    print( X^I );
}
2
4
8
16
32
64
128
256
0

```

図 2.4: for 文

繰り返し関連の表現の意味を箇条書にしてまとめておこう.

1. `for (K=初期値; K<=終りの値; K++) {ループの中で実行するコマンド};` はあることを何度も繰り返したい時に用いる. for ループと呼ばれる. “ $K \leq N$ ” は, “ $K \leq N$ か” という意味である. 似た表現に, “ $K > N$ ” があるが, これは “ $K \geq N$ か” という意味である. “ $K < N$ ” は, “ $K < N$ か” という意味である.

2. ++K や K++ は K を 1 増やせという意味である。K = K+1 と書いてもよい。同じく、--K や K-- は K を 1 減らせという意味である。

```
X = 2;
for (I=1; I<=8; I++) {
    print("2の"+rtostr(I)+"乗は ",0);
    print( X^I );
}
```

2の1乗は 2
2の2乗は 4
2の3乗は 8
2の4乗は 16
2の5乗は 32
2の6乗は 64
2の7乗は 128
2の8乗は 256
0

図 2.5: for 文

for のあとの {, } の中には複数の文 (命令) を書ける。

```
X = 2;
for (I=1; I<=8; I++) {
    print("2 の"+rtostr(I)+"乗は ",0);
    print( X^I );
}
```

この例では日本語を含むので前の節で述べたように日本語空白をプログラム本体にいれないようにして、注意深くプログラムを入力してもらいたい。実行結果は図 2.5 をみよ。print("2 の"+rtostr(I)+"乗は ",0); の部分を簡単に説明しておこう。まず最後の 0 は出力のあと改行しない、つまり次の print 文の出力をそのまま続けよという意味。" でかこまれた部分は文字列と呼ばれている。これはそのまま表示される。rtostr(I) は数字 I を文字列表現に変換しなさい、という意味 (超入門としては難しい?)。あと文字列に対して + を適用すると文字列が結合される。

雑談 (江戸時代の数学の本にあった問題の改題)

殿様: このたびの働きはあっぱれであった。褒美はなにがよいか?

家来: 今日は 1 円, 明日は 2 円, 明後日は 4 円と, 前日の 2 倍ずつ, これを 4 週間続けてくださるだけで結構でございます。

殿様: なんともささやかな褒美じゃのう。よしよし。

さて, 家来はいくら褒賞金をもらえるだろう? これもまた 2 の累乗の計算である。Cfep/asir で計算してみよう。

例 2.2 for による繰り返しを用いて \sqrt{x} の数表をつくろう。

```
for (I=0; I<2; I = I+0.2) {
    print(I,0); print(" : ",0);
    print(deval(I^(1/2)));
}
```

出力結果

```
0 : 0
0.2 : 0.447214
0.4 : 0.632456
0.6 : 0.774597
0.8 : 0.894427
1 : 1
1.2 : 1.09545
1.4 : 1.18322
1.6 : 1.26491
1.8 : 1.34164
2 : 1.41421
```

`print(A)` は変数 `A` の値を画面に表示する。 `print(文字列)` は文字列を画面に表示する。 `print(A,0)` は変数 `A` の値を画面に表示するが、表示したあとの改行をしない。空白も文字である。したがって、たとえば `A=10; print(A,0); print(A+1);` を実行すると、`1011` と表示されてしまう。 `A=10; print(A,0); print(" ",0);print(A+1);` を実行すると、`10 11` と表示される。

ところで、この例では条件が $I < 2$ なのに $I = 2$ の場合が表示されている。実際に `asir` 上で実行してみるとこうなるが、理由を知るには、浮動小数の計算機上での表現についての知識が必要である (“`asir` ドリル” を参照)。とりあえず、

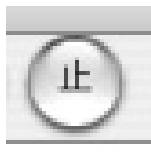
整数や分数の計算は `Asir` 上で正確に実行されるが、小数についてはそうでない。

と覚えておこう。

問題 2.1 あたえられた 10 進数を 2 進数へ変換するプログラムを作れ。ヒント: $A \div B$ の余りは `A%B` で計算できる。

2.3 実行の中止

実行中の計算やプログラムの実行を中止したい時は中止ボタン



をクリックする。

図 2.6 では 10^{100} 回の `Hello` の出力の繰り返しを中止している。
`cfep` は開発途上のシステムのため



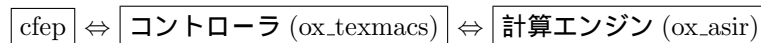
図 2.6: 実行の中止

```
[control] control function_id is 1030
[control] control_reset_connection.
Sending the SIGUSR1 signal to 1226: Result = 0
In ox103_reset: Done.
515
Done
```

このような開発者専用のメッセージも出力されるが、とりあえずこのようなメッセージがでたら中止が成功したということである。

2.4 エンジン再起動

Cfep/asir では次のように 3 つのプロセスが互いに通信しながら動作している。



計算エンジン (計算サーバ) を再起動したり別のものにとりかえたりできる。

エンジン再起動ボタン



をクリックすると、現在利用している計算エンジンを停止し、新しい計算エンジンをスタートする。選択範囲のみを実行するモードでないかぎり利用上で中止との違いはあまりないが、再起動のときのメッセージ



にもあるように、別の計算エンジンを起動することも可能である。この例では unix shell も起動できる。

また、“実行”メニューから“エンジンを自動スタートしない”モードを選んでも場合に計算エンジンを手動でスタートするには、このボタンを用いる。

発展学習 cfep は Cocoa FrontEnd view Process の略である。cfep は Objective C という言語および xcode 2 という開発環境を用いて Cocoa というフレームワークのもとで開発されている。cfep の Objective C のプログラムの一部をみてみよう。

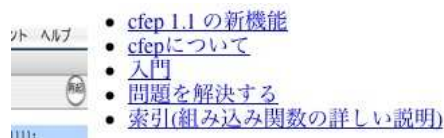
```
for (i=0; i<oglCommSize; i++) {
    gc = [oglComm objectAtIndex: i];
    [self execute: gc];
}
```

asir と同じような for 文があるね。

2.5 ヘルプの利用

Cfep/asir での“関数”とは数学の関数のように引数を与えると計算して値をもどし、かつある仕事(表示等)をする手続きの集まりである。例えば print, deval, sin, fctr 等は関数である。関数を自分で定義することも可能である。これについては後の説明および“asir ドリル”を参照。

あらかじめ定義済みの関数を“組み込み関数”とよぶ。組み込み関数の詳しい説明を調べるには“cfep のヘルプ”から



の“索引”を選び、索引

1. 使用説明書のフォルダをfinderで開く



検索したい言葉をスポットライトの窓へ入力。
検索のヒント: PDF文書のみから検索したい場合は“検索語 kind:pdf”と入力。

2. cfep の操作説明(まだ書いてない)
3. [Risa/Asir マニュアル](#)
4. [Risa/Asir 実験的機能マニュアル](#)
5. [Asir マニュアル](#), [Asir-contrib](#), [実験的関数マニュアル](#)も含むより詳しい関数一覧(ネ

の“Risa/Asir マニュアル”を選び、“Risa/Asir マニュアル”の最初のページの関数一覧から調べたい関数を探す。たとえば fctr (因数分解用の関数)はこの一覧の中にある。

- [lsay](#)
- [i](#)
- [subst.psubst](#)
- [diff](#)
- [ediff](#)
- [res](#)
- [fctr.sqfr](#)
- [ufctrhint](#)
- [modfctr](#)
- [ptozp](#)

検索には spotlight の活用も有益であろう。索引

1. 使用説明書のフォルダをfinderで開く



検索したい言葉をスポットライトの窓へ入力。
検索のヒント: PDF文書のみから検索したい場合は "検索語 kind:pdf" と入力。

2. cfep の操作説明 (まだ書いてない)
3. [Risa/Asir マニュアル](#)
4. [Risa/Asir 実験的機能マニュアル](#)
5. [Asir マニュアル](#), [Asir-contrib](#), [実験的関数マニュアル](#) も含むより詳しい関数一覧 (ネ

の “使用説明書のフォルダを finder で開く” を選ぶと使用説明書のフォルダが開くので、ここを spotlight で検索するといろいろな発見があるであろう。ちなみに、この超入門や asir ドリルはこのフォルダの pdf フォルダの中にある。(なおここからの spotlight 検索は何故か遅いので、メニューバーの spotlight からの検索の方がいいかもしれない。)

第3章 グラフィック

3.1 ライブラリの読み込み

Asir 言語で書かれている関数定義の集合がライブラリである。ライブラリを読み込むには `import` コマンドまたは `load` コマンドを用いる。マニュアルに記述されている関数でライブラリの読み込みが前提となっているものも多い。たとえば、線を引くコマンド `glib_line(0,0,100,100);` を実行しても、“`glib_line` が定義されていません” というエラーが表示される。グラフィックコマンドのライブラリ読み込むコマンド

```
import("glib3.rr");
```

を実行しておくくと図 3.1 のように線を描画する。

Asir-contrib プロジェクトにより集積されたライブラリの集合体が `asir-contrib` である。Asir-contrib を読み込んでしまうと、ほとんどの関数について `import` が必要かどうか気にする必要はなくなるが、大量のライブラリを読み込むために時間がかかるのが欠点である。`asir-contrib` は **実行** メニューから読み込める。



3.2 線を引く関数

```
例 3.1  import("glib3.rr");
        glib_line(0,0, 100,100);
        glib_flush();
```

図 3.1 が描画結果である。 y 座標は画面が下へいくほど大きくなる。図 3.2 を参照。左上の座標は $(0,0)$ 、右下の座標が $(400,400)$ 。 `glib_line` で $(0,0)$ から $(100,100)$ へ線を描画。 `glib_flush` は画面を更新するはたらきがある。 `flush` しないと、描画結果が画面での表示に反映しない場合がある。

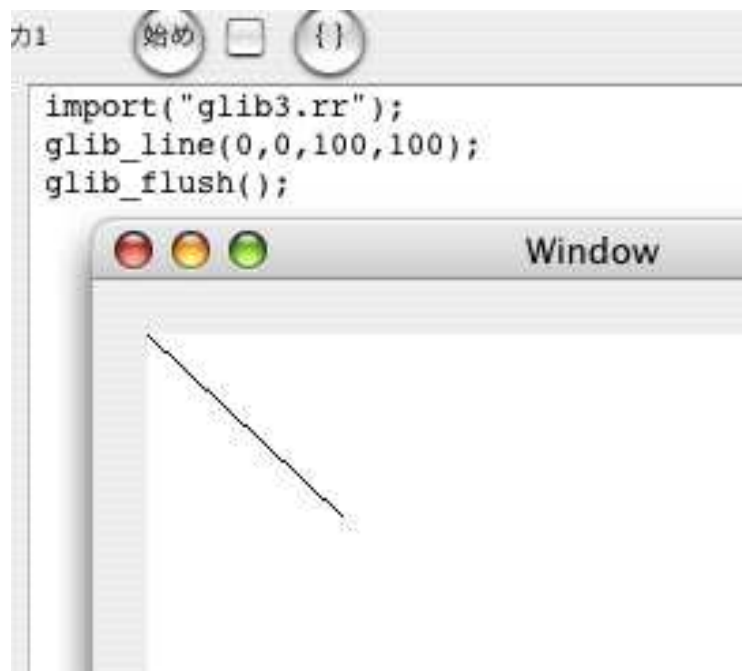


図 3.1: ライブラリのロード

`glib3.rr` をロードすることにより, 次の関数が使えるようになる.

<code>glib.window(X0,Y0,X1,Y1)</code>	図を書く window のサイズを決める. 画面左上の座標が $(X0, Y0)$, 画面右下の座標が $(X1, Y1)$ であるような座標系で以下描画せよ. ただし x 座標は, 右にいくに従いおおきくなり, y 座標は <u>下</u> にいくに従い大きくなる (図 3.2).
<code>glib.clear()</code>	全ての OpenGL オブジェクトを消去し, 描画面面をクリアする.
<code>glib.putpixel(X,Y)</code>	座標 (X, Y) に点を打つ.
<code>glib.set_pixel_size(S)</code>	点の大きさの指定. 1.0 が 1 ピクセル分の大きさ.
<code>glib.line(X,Y,P,Q)</code>	座標 (X, Y) から座標 (P, Q) へ直線を引く
<code>glib.remove_last()</code>	一つ前の OpenGL オブジェクトを消す.



図 3.2: 座標系

色を変更したいときは, | 記号で区切ったオプション引数 `color` を使う. たとえば,

```
glib_line(0,0,100,100|color=0xff0000);
```

と入力すると、色 0xff0000 で線分をひく。ここで、色は RGB の各成分の強さを 2 桁の 16 進数で指定する。16 進数については“asir ドリル”を参照。この例では、R 成分が ff なので、赤の線をひくこととなる。なお、関数 `glib_putpixel` も同じようにして、色を指定できる。16 進数を知らない人用に、色とその 16 進数による表現の対応表をあげておく。

0xffffffff	白
0xffff00	黄
0xff0000	赤
0x00ff00	緑
0x0000ff	青
0x000000	黒

(あとは試して下さい)

さて、図 3.2 で見たようにコンピュータプログラムの世界では、画面の左上を原点にして、下へいくに従い、 y 座標が増えるような座標系をとることが多い。数学のグラフを書いたりするにはこれでは不便なことも多いので、`glib3.rr` では、

```
Glib_math_coordinate=1;
```

を実行しておくとも画面の左下が原点で、上にいくに従い y 座標が増えるような数学での座標系で図を描画する。

例 3.2 2 次関数 $y = x^2 - 1$ のグラフを書いてみよう。

```
import("glib3.rr");
Glib_math_coordinate=1;
glib_window(-2,-2, 2,2);

glib_line(-2,0,2,0 | color=0x0000ff);
glib_line(0,-2,0,2 | color=0x0000ff);
for (X=-2.0; X< 2.0; X = X+0.1) {
    Y = X^2-1;
    X1 = X+0.1;
    Y1 = X1^2-1;
    glib_line(X,Y, X1,Y1);
}
glib_flush();
```

実行結果は図 3.3. —プログラムの解説はまだ書いてない。

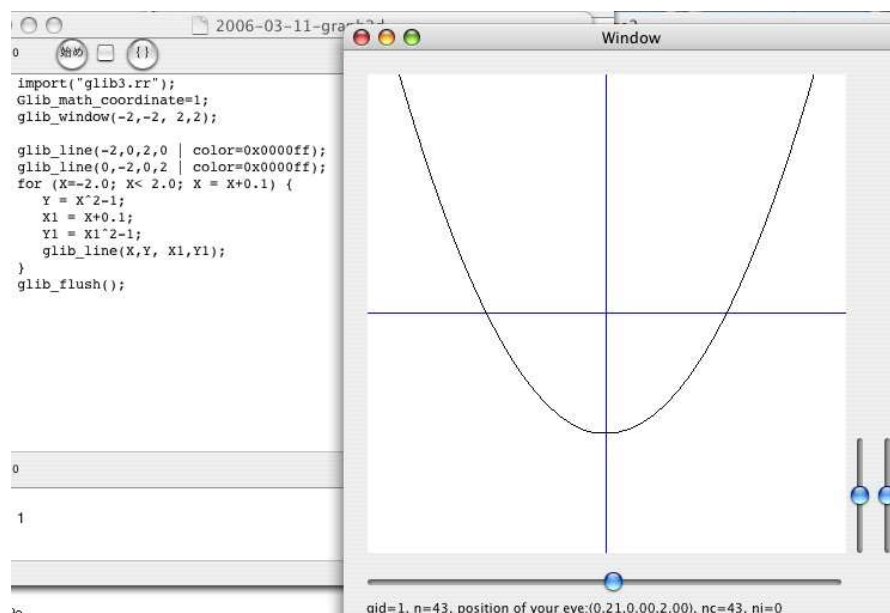


図 3.3: 2 次関数のグラフ

3.3 円を描く関数を作ってみよう

```
import("glib3.rr");
Glib_math_coordinate=1;
glib_window(-1,-1,1,1);
glib_clear();
E = 0.2; X = 0; Y = 0; R = 0.5;
for (T=0; T<=deval(2*@pi); T = T+E) {
  Px = X+deval(R*cos(T));
  Py = Y+deval(R*sin(T));
  Qx = X+deval(R*cos(T+E));
  Qy = Y+deval(R*sin(T+E));
  glib_line(Px,Py,Qx,Qy);
  glib_flush();
}
```

—プログラムの解説はまだ書いてない。

上のプログラムでは \cos , \sin を用いて円を描いている。中心, 半径を変更したり, 色を変更したりしながらたくさんの円を描くには, どのようにすればよいであろうか? “関数” を用いるとそれが容易にできる。

あるひとまとまりのプログラムは関数 (function) としてまとめておくとよい。計算機言語における関数は数学でいう関数と似て非なるものである。関数を手続き (procedure) とか サブルーチン (subroutine) とかよぶ言語もある。関数を用いる最大の利点は, 関数を一旦書いてしまえば, 中身をブラックボックスとして扱えることである。大規模なプログラムを書くときは複雑な処理をいくつかの関数に分割してまず各関数を十分テストし仕上げる。それからそれらの関数を組み合わせていくこ

とにより、複雑な機能を実現する。このようなアプローチをとることにより、“困難が分割”される。

円の例をしばらく離れ、簡単な関数の例をとり関数の書き方を説明しよう。前の節では2の中を表を作成するプログラムを書いた。これを元に次のような関数を作る。

```
def power_table(N) {
  X=2;
  for (I=1; I<=N; I++) {
    print(X^I);
  }
}
power_table(8);
```

関数の定義は次のように def 命令で行なう。

```
def 関数名(引数) {
  関数本体
}
```

上の例では関数名は power_table であり、引数(argument)は N である。関数名は英数字と _ を用いてつける。ただし数字や大文字ではじまる名前をつけることはできない。処理内容を連想させるような名前をつけるのが望ましい。def 命令では関数を定義するだけで実行は行なわない。実行させるには、上の例のように power_table(8); と引数の部分に実際の数字等を入れて呼び出す。

引数は二つ以上あってもよくて、たとえば、

```
def power_table2(X,N) {
  for (I=1; I<=N; I++) {
    print(X^I);
  }
}
power_table2(3,8);
```

なる関数定義と最後の行のその呼び出しは 3^i を $i = 1, \dots, 8$ の範囲で計算して表示する。このような関数を用意しておけば、 $2^i, 3^i, 5^i, 1 \leq i \leq 10$ の表を表示したいとすると、

```
power_table2(2,10);
power_table2(3,10);
power_table2(5,10);
```

と書くだけで良く、プログラムも短くなり、かつ整理されているので、読みやすくなる。

さて、power_table2(2,3); を実行すると

```
2
4
8
0
```

と表示される。最後の 0 は一体何であろうか？ これは実は関数の値である。関数の値のことを戻り値

ともいう。戻値を指定するには、`return` 文を用いる。

```
def twotimes(N) {
    S=2*N;
    return(S);
}
```

関数 `twotimes(N)` は $2N$ の値を計算して戻す。

この定義を書いておいて

```
A=twotimes(10);
B=twotimes(100);
print(A+B);
```

を実行すると、`A` には 20 が代入され、`B` には 200 が代入され、`print` 文により 220 が出力される。そのあと 0 が出力されるがこれは `print` 文の戻り値である。

関数の戻り値 (return value) は `return` 文で指定する。いまの場合変数 `S` の値である。なお、`print` と `return` は違う。`print` は画面に値を印刷するのに対して、`return` は関数の値を戻す働きを持つ。`print` 文では、関数の値を戻すことはできない。

“戻り値” (return value) という言い方は計算機言語特有の言いまわしである。“関数 `twotimes` は引数の 2 倍を計算して結果を戻す” みたいに使う。上の例でいえば `A=twotimes(10)` としたとき、戻り値が関数の値として変数 `A` に代入される。

関数のなかで利用されている変数と引数は、その関数の実行中のみ生成される変数であり、さらにその関数外部の同名の変数の値を変えない。このように一時的に生成される変数を局所変数 (local variable) とよぶ。関数の中で変数の値を変更したら、その関数の外の同じ名前の変数の値もかわってしまうとしたら、処理を分割した利点がすくない。そこででてきた概念がこの“局所変数”の概念である。上のプログラム例では、`N`, `S` が局所変数である。局所変数はその関数のなかだけで有効な変数である。これを、“局所変数のスコープはその関数のなかだけ” という言いかたをする。局所変数の考え方は、計算機言語の歴史では大発明の一つである。

例:

```
S=3;
twotimes(2);
print(S);
```

このプログラムの `S` と関数 `twotimes` のなかの変数 `S` は別物である。したがって、`twotimes` の終了時点で関数 `twotimes` のなかの変数 `S` の値は 6 であるが、`print` 文で `S` の値を表示させてみてもやはり 3 のままである。

さて円を描く例にもどろう。以下のように関数 `circle(X,Y,R,Color)` を定義 (def) する。この関数を `R` や `Color` を変化させながら呼ぶことにより、図 3.4 のような同心円の図を描くことが可能となる。関数についてさらに詳しくは“`asir` ドリル”を参照してほしい。

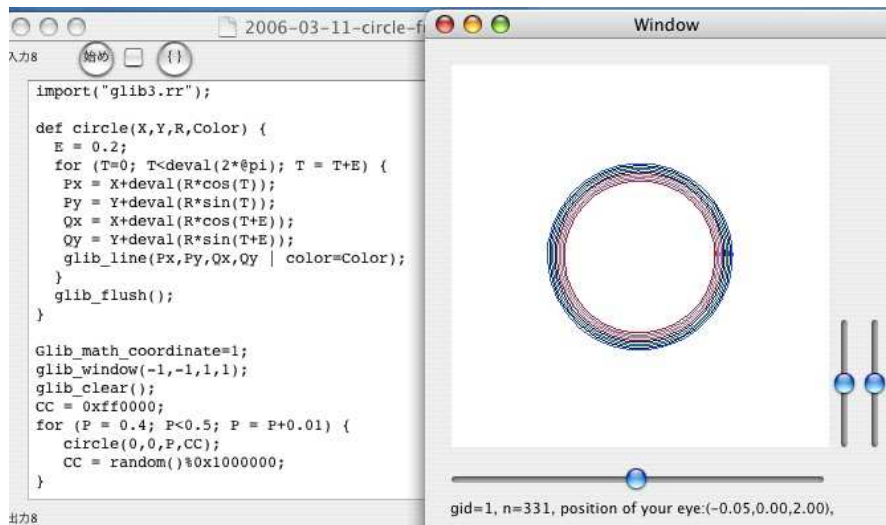


図 3.4: 関数による同心円の描画

```
import("glib3.rr");

def circle(X,Y,R,Color) {
  E = 0.2;
  for (T=0; T<deval(2*@pi); T = T+E) {
    Px = X+deval(R*cos(T));
    Py = Y+deval(R*sin(T));
    Qx = X+deval(R*cos(T+E));
    Qy = Y+deval(R*sin(T+E));
    glib_line(Px,Py,Qx,Qy | color=Color);
  }
  glib_flush();
}

Glib_math_coordinate=1;
glib_window(-1,-1,1,1);
glib_clear();
CC = 0xff0000;
for (P = 0.4; P<0.5; P = P+0.01) {
  circle(0,0,P,CC);
  CC = random()%0x1000000;
}
}
```

(Q_x, Q_y) と (P_x, P_y) は E だけ偏角が異なる。円を多面体で近似して描画している。

—プログラムの詳しい解説まだ。

問題 3.1 1. このプログラムの関数を用いて接する半径の同じ円を二つ描画するプログラムを書きなさい。

2. 円を塗り潰す関数を作れ.
3. 分度器を描くプログラムを作れ.
4. (発展課題) この分度器, 糸, おもり, わりばし, 板, cfep/asir によるプログラム等を用いて, 木やビルの高さを測定する機械とソフトウェアシステムを開発せよ.

問題 3.2 (これは発展課題) cfep には OpenGL インタプリターが組み込んである. OpenGL は 3 次元グラフィックスを用いるソフトウェア作成のために用いられる約 150 種類のコマンドから構成されているパッケージで 3 次元グラフィックスの標準規格のひとつでもある. cfep 1.1 ではその中の 10 弱のコマンドを利用できる.

この OpenGL インタプリターを用い, 多面体 (polygon) を材料にし, cfep 上級編, OpenGL のプログラムを参考に “家” を書いてみよう.

実習の落とし穴

1. unix 版, windows 版では asir ドリルにあるように end\$ をプログラムの最後に書かないと行けないが, cfep/asir ではこの命令は計算エンジンの停止命令となるので書いてはいけない. ただし 行の頭に空白をいれずに書いてある end\$ は自動削除されるので, unix, windows 版共通のプログラムを書くときは空白を入れずに行の頭に書いておくと便利である.

2. 空白や改行は原則自由に挿入していいが, 空白を入れてはいけない表現がある. たとえば

0.1

と書くべきところを

0. 1

と 1 の前に空白を入れるとエラーとなる. 数字には空白を入れてはいけない. 理由は asir ドリルの構文解析の章を読むと理解できると思う.

3. $\sin x$ なる表現は受け付けてくれない. 必ず括弧がいる. つまり $\sin(x)$ と書く.

4. 掛け算の * は省略できない.

発展学習. [1,2,3] のように [] で囲った表現をリスト (list) と呼ぶ. 関数の戻り値を数の組にしたときはリストを値として戻すと良い. print 文で多くの数を一行で表示したいときもリストを使うと便利である. L をリストとするとき L[0] でリストの最初 (0 番目) の元, L[1] でリストの 1 番目の元, ... を表す. 詳しくは asir ドリル参照.

```
L=[3,2,1];
print(L);
print(L[0]+L[1]+L[2]);
```


第4章 For 文による数列の計算

4.1 超入門, 第2の関門: 漸化式でできる数列の計算

例 4.1 a を正の数とすると,

$$\begin{aligned}x_{n+1} &= \frac{x_n + \frac{a}{x_n}}{2}, \\x_0 &= a\end{aligned}$$

できる数列 x_0, x_1, x_2, \dots は \sqrt{a} にどんどん近付くこと (収束すること) が知られている. $a = 2$ の時, $x_1, x_2, \dots, x_4, x_5$ を計算するプログラムを書いてみよう.

```
A = 2.0;
X = A;
for (I=0; I<5; I++) {
  Y = (X+A/X)/2;
  print(Y);
  X = Y;
}
```

このプログラムの実行結果は図 4.1.

超入門での関門は

```
Y = (X+A/X)/2;
X = Y;
```

の意味を完全に理解すること

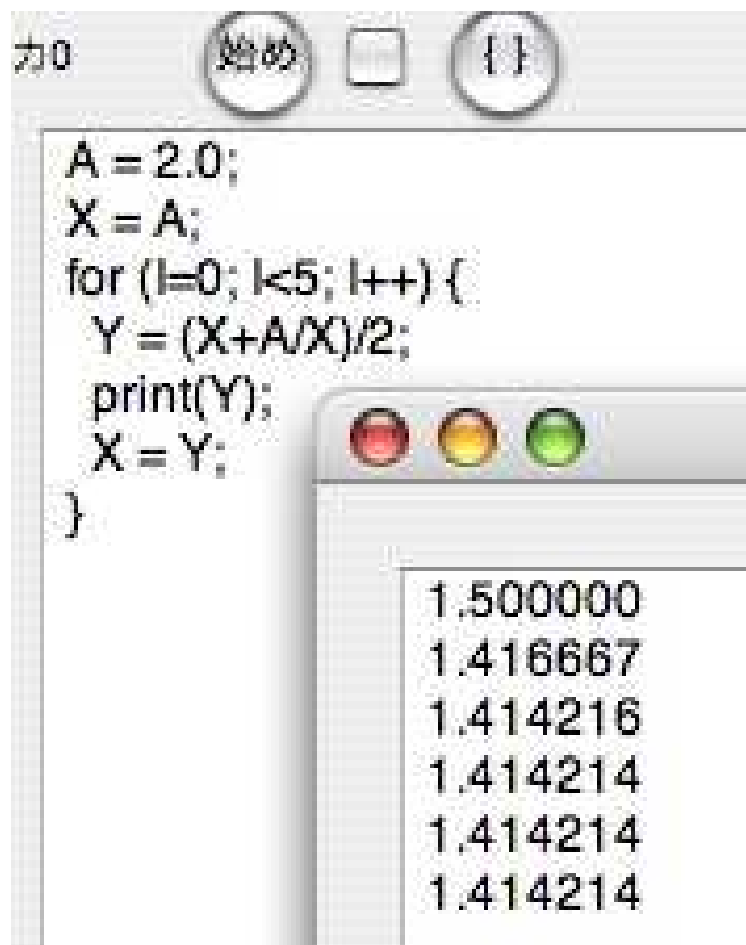
である. 変数の章で説明したように,

変数名=式;

はまず右辺の式を計算しそのあとその計算結果を左辺の変数に代入せよという意味である. したがって, $Y = (X+A/X)/2;$ は現在の X と A に格納された数字をもとに $(X+A/X)/2$ の値を計算し, その結果を変数 Y へ代入せよ, という意味である. また

$X=Y$ は X が Y に等しいという意味ではなく, 変数 Y に格納された数字を 変数 X に代入せよという意味である.

このように考えれば, 上のプログラムが x_1, x_2, x_3, x_4 の値を順番に計算して print している理由が理解できるであろう. 自分が計算機になったつもりで, 変数の中の数値がどのように変化していくのか, 書きながら理解して頂きたい. これがはっきり理解でき, 応用問題が自由に解けるようになった, 超入門卒業である.

図 4.1: $\sqrt{2}$ に収束する数列

問題 4.1 変数 I, X, Y の値は for ループ内でどのように変化するか? $Y = (X+A/X)/2$ の行が実行される前のこれらの変数の値を表にまとめてよ. `print([I,X,Y])` をはさむことによりこの表が正しいことをたしかめよ.

表は次のような形式で書く.

I	0	1	2	3	4
X					
Y					

問題 4.2 プログラムのバグ (bug) とはなにか?

4.2 円を描く数列

前の章で円を描く関数を紹介した. \sin, \cos の計算に時間がかかる計算機では, なるべくこれら三角関数を用いなくて円を描画する必要がある.

一つの方法は $s = \tan \frac{t}{2}$ とおくと $\cos t = \frac{1+s^2}{1-s^2}$, $\sin t = \frac{2s}{1-s^2}$ と書けるという公式を使う方法である. s はタンジェントで定義されていることを忘れてしまえばよい.

もう一つは数列の計算を用いて, \cos や \sin の計算をやらずに円を描く方法である.

```

import("glib3.rr");
Glib_math_coordinate=1;
glib_window(-2,-2, 2,2);
glib_clear();
E = 0.1;
C1 = 1.0; C2=1.0;
S1 = 0.0; S2=E;
for (T=0; T<=deval(2*@pi); T = T+E) {
    C3 = 2*C2-C1-E*E*C2;
    S3 = 2*S2-S1-E*E*S2;
    glib_line(C1,S1, C2,S2);
    C1=C2; S1=S2;
    C2=C3; S2=S3;
    glib_flush();
}

```

このプログラムの実行結果は図 4.2.

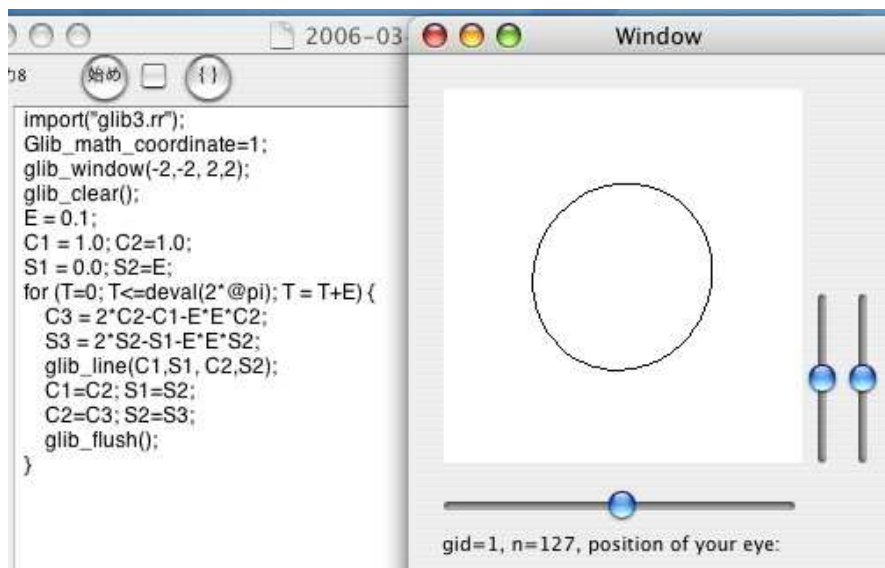


図 4.2: cos, sin を使わずに円を描く

—プログラムの解説まだ書いてない。

ヒント: 微分方程式 $d^2x/dt^2 = -x$, $d^2y/dt^2 = -y$ を t をラジアンとして差分法で近似的に解いている。

この話題は、数列の計算と差分方程式によるシミュレーションに続く。これについてはまた稿をあらためて書いてみたい。

以上で超入門は終了である。続きは“Asir ドリル”を読んでね。特に配列と関数をマスターすると数学プログラムには重宝する。

なお, asir ドリルに紹介してあるプログラムは `end$` または `end;` が最後に書いてある場合が多いが, cfep/asir ではこの `end` を書いてはいけない。 `end` は計算エンジンの停止命令であり, 実行

されると“計算中の表示”がでて無応答となる。(一応、行頭にあるこれらの命令は自動的に削除するようになってはいる。)

問題 4.3 (レポート問題の例)

なにか図を描くプログラムを書きなさい。(定番ドラエモンでもよい)

ノート 著者の場合、週に1コマ講義、演習1コマでは、ここまでで3週である。上のレポート問題が最初のレポート。3週目の演習の時間からとりかかり、5週目の演習の時間に発表。4週目からは asir ドリル。

第5章 cfep 上級編

5.1 T_EX によるタイプセット (実験的)

出力を TeX でタイプセットするには“実行”メニューから“出力を TeX でタイプセット”を選択する。latex, dvipng がインストールされていないと動作しない。これらはたとえば fink から T_EX をインストールしたり, ptex_package_2005v2.1.dmg (最近の状況は ptex package macosx で検索)などで Mac 用の pTeX をインストールしておけばよい。T_EX を用いた仕上り例は図 5.1 を見よ。なお, T_EX でタイプセットする場合ホームの下に OpenXM_tmp なる作業用のフォルダが作成される。タイプセットは実験機能のため, このフォルダの中の作業用ファイルは自動では消去されない。時々手動で作業ファイルを消去されたい。

5.2 選択範囲のみの実行

画面上の“選択範囲のみを実行”をチェックすると, “始め”ボタンをおしたとき, 選択範囲のみが評価される。選択範囲がない場合は caret 位置の行が自動選択されて実行される。[COMMAND]+[Enter] と組み合わせてこの機能を使うと, ターミナルから asir を利用するのにちょっと似てくる。図 5.1 はこのような実行をしている例である。

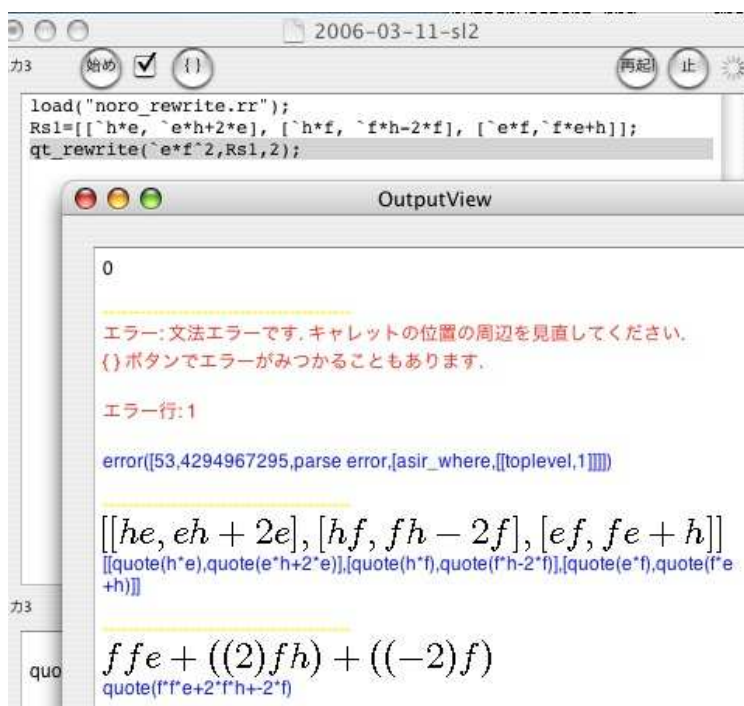
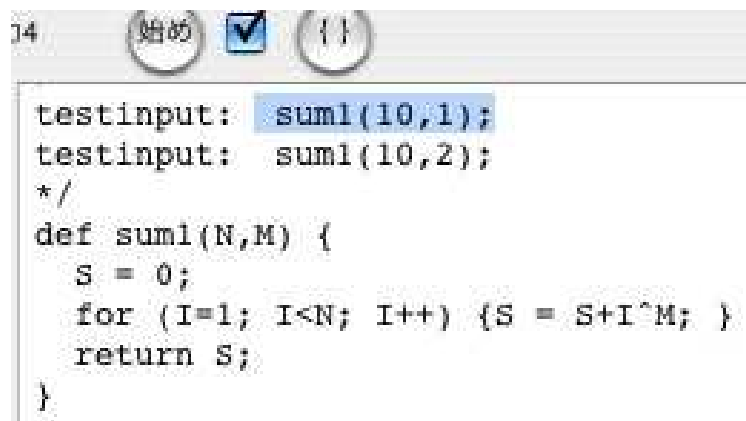


図 5.1: ターミナル風



```

14  (始め) (✓) ({} )
testinput: sum1(10,1);
testinput: sum1(10,2);
*/
def sum1(N,M) {
  S = 0;
  for (I=1; I<N; I++) {S = S+I^M; }
  return S;
}

```

図 5.2: “選択範囲のみを実行” の活用

質問 cfep のインタフェースでデバッグをしながらプログラムを開発するにはどのようにやるとよいか?

答え cfep は初心者向けのインタフェースなので、大規模なプログラム開発を想定していないが、私は次のようにライブラリの開発をしている。

1. 必要な関数を書く。下の例では sum1.
2. 関数をテストする入力をコメントの形でその関数の近くを書いておく。下の例ではコメントにある sum1(10,1); 等.

```

/*
testinput: sum1(10,1);
testinput: sum1(10,2);
*/
def sum1(N,M) {
  S = 0; i=1;
  for (I=1; I<N; I++) {S = S+I^M; }
  return S;
}

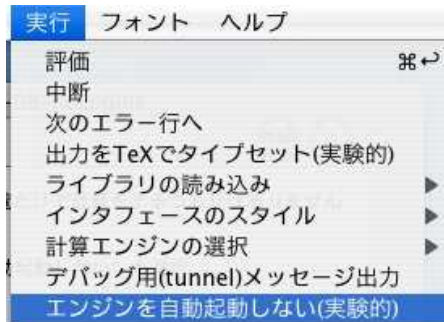
```

1. “始め” ボタンで関数定義をロード。この時点で文法エラーなどがあればメッセージにしたがって修正。
2. そのあと “選択範囲のみを実行” のモードに変更してコメント内の testinput を実行。
3. 実行時のエラーの行番号への移動は “選択範囲のみを実行” のモードを解除してから行う。

5.3 エンジンを起動しない

質問 テキスト編集またはテキストの閲覧だけで計算をするつもりはありませんが。

答え “実行” メニューで “エンジンを自動起動しない” を選択。



あとでエンジンを起動したい場合は“再起”ボタンをおしてエンジンを起動する。



5.4 OpenGL インタプリタ

Cfep には OpenGL インタプリターが組み込んである。OpenGL は 3 次元グラフィックスを用いるソフトウェア作成のために用いられる約 150 種類のコマンドから構成されているパッケージで 3 次元グラフィックスの標準規格のひとつでもある。cfep 1.1 ではその中の 10 弱のコマンドを利用できる。詳しくは cfep.app/OpenXM/lib/asir-contrib/cfep-opengl.rr を参照。

OpenGL ではまず OpenGL グラフィックオブジェクトを配置し、それから視点の位置から見た画像を描画する方法を用いる。したがって、システムは常に OpenGL グラフィックオブジェクトの集合を保持している。glib_remove_last() 命令はその最後の要素を削除する命令である。cfep-opengl.rr ライブラリでは、opengl.metaRemoveLast() 関数で最後の要素を削除できる。

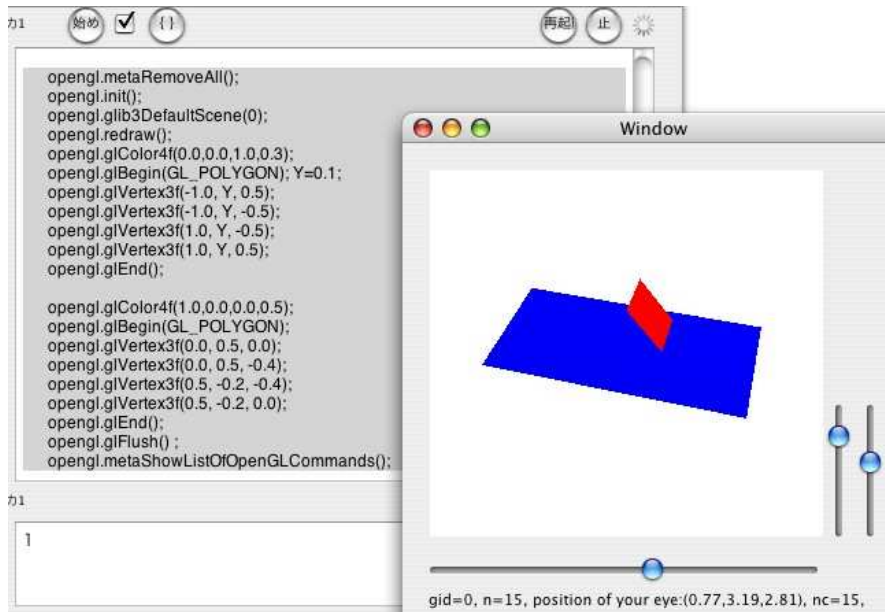


図 5.3:

```

import("cfep-opengl.rr");
opengl.metaRemoveAll();
opengl.init();
opengl.glib3DefaultScene(0);
opengl.redraw();
opengl.glColor4f(0.0,0.0,1.0,0.3);
opengl.glBegin(GL_POLYGON); Y=0.1;
opengl.glVertex3f(-1.0, Y, 0.5);
opengl.glVertex3f(-1.0, Y, -0.5);
opengl.glVertex3f(1.0, Y, -0.5);
opengl.glVertex3f(1.0, Y, 0.5);
opengl.glEnd();

opengl.glColor4f(1.0,0.0,0.0,0.5);
opengl.glBegin(GL_POLYGON);
opengl.glVertex3f(0.0, 0.5, 0.0);
opengl.glVertex3f(0.0, 0.5, -0.4);
opengl.glVertex3f(0.5, -0.2, -0.4);
opengl.glVertex3f(0.5, -0.2, 0.0);
opengl.glEnd();
opengl.glFlush();
opengl.metaShowListOfOpenGLCommands();

```

このプログラムでは 2 枚の長方形を描いている。このプログラムの出力は図 5.3。— 詳しい説明はまだ。

OpenGL の画面には普通の数学のように (x, y) 座標がはいており, 画面から手前側が z 座標が正の方向, 画面の向こう側が z 座標が負の方向である. “目” から原点方向を見た画像が図 5.3 にあるように 3 つのスライダーを用いて目の位置を動かせるので, OpenGL オブジェクトをいろいろな角度からみることが可能である. 下のスライダーが目の x 座標, 右の二つのスライダーがそれぞれ目の y, z 座標である. 目の動きに慣れるには, 次の二つのデモ画面をためすと面白いだろう.

```
import("cfep-opengl.rr");  
opengl.glib3DefaultScene("mesa demo/ray");
```

```
import("cfep-opengl.rr");  
opengl.glib3DefaultScene("cfep demo/icosahedron");
```

5.5 asir 以外の計算エンジンの利用

cfep から asir 以外の OpenXM 準拠の計算エンジンも利用できます. たとえば, 実行, 計算エンジンの選択で, kan/sm1 を利用することも可能です. cfep, ox_texmacs, ox_sm1 が相互に通信しながら計算しています.

なお kan/sm1 の run コマンドは使えません.

[(parse) (ファイル名) pushfile] extension

で代用して下さい.

索引

- ;, 6
- =, 16
- <=, 18
- 2 の累乗, 11, 15, 19
- 3 次元グラフィックス, 32
- asir-contrib, 25
- cfep, 22
- deval, 9
- fctr, 17, 23
- for, 18
- for 文, 18
- glib, 25
- help, 22
- import, 25
- interrupt, 20
- load, 25
- OpenGL, 32, 39
- opengl, 39
- OpenGL グラフィックオブジェクト, 39
- OutputView, 6
- plot, 11
- print, 19, 20
- restart, 21
- rtostr, 19
- spotlight, 23
- T_EX, 37
- X11, 11
- 余り, 20
- 因数分解, 17
- エラー, 11, 38
- エラーメッセージ, 11
- エンジン, 21
- オプション引数, 26
- 括弧, 13
- 関数, 22, 28
- 組み込み関数, 22
- くりかえし, 18
- 繰り返し, 18
- クリック, 6
- グラフ, 27
- 計算エンジン, 21
- 計算サーバ, 21
- 再起動, 21
- 出力結果, 10
- 出力小窓, 6
- 数式処理, 17
- 選択範囲のみの実行, 37
- 多項式, 17
- 多項式の変数名, 15
- 多項式変数, 17
- 代入, 16, 33
- 代入記号=, 16
- ダブルクリック, 6
- 中止, 20
- 次のエラー行へ, 13
- 入力窓, 6
- 評価, 6
- 文法エラー, 12
- プログラム, 10
- ヘルプ, 22
- 変数, 15, 17
- 変数名, 15
- 文字列の結合, 19
- 文字列表現, 19

ライブラリ, 25

ラジアン, 9