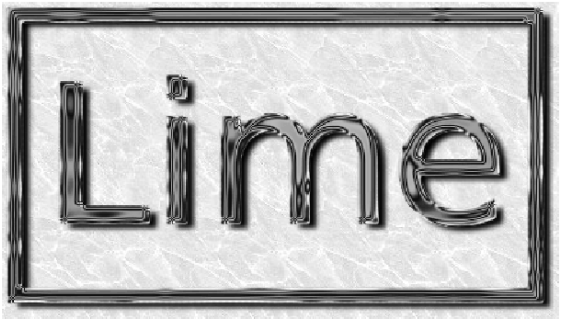
平成 18 年 11 月 24 日
京都工芸繊維大学コンピュータ部



はじめに

こんにちは。今年も Lime を無事発行できました。編集を担当してくれた林君、小長谷君には感謝します。Lime は、部員各々がコンピュータ部における活動の一部を記したものです。興味がございましたら、手にとって読んでいただけると光栄です。

ところで、今年は、本学、コンピュータ部に共に変革の時期となりました。学部、学域が大きく再編され、コンピュータ部においても例年より数倍も多い数の新入生を迎え、また、BOX も改修工事中であります。そういうと未来は明るく見えます。とはいうものの、現コンピュータ部の抱える問題もまた多く、油断のできない状況にあるのも事実です。けれども、コンピュータ部はさまざまな困難を乗り越え、存続してきたものであるし、今後も成長続けてていくことができると思っております。そのためには、惜しみなく努力をして行く所存です。その一つとして、この Lime34 号があると私は思うのです。

平成 18 年 11 月 19 日
コンピュータ部部長 黒田龍二

目 次

1 LEDディスプレイについて

電子システム工学課程　1回生 小長谷 拓、楠 健也

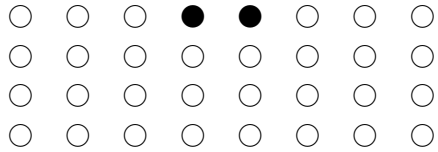
1.1 はじめに

ここでは、先輩が苦労して作ったと思われるLED（発光ダイオード）ディスプレイについて、私たちがわかる範囲で説明？していきます。このLEDディスプレイは512個ものLEDを使っています。LEDはすべてプラスチック製のダンボール上にうまく配置されています。他にもいろいろな工夫が見られる作品です。ところで、512個のLEDをどうやって制御して、文字を表示するのでしょうか。

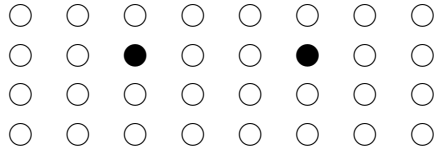
1.2 動作原理

LEDはすべてプロセッサによって制御されています。しかし、LED1個につきプロセッサの出力ポート（出力端子）を1本使ってしまうと、とても非効率的です。つまり、この場合、プロセッサには512本もの出力ポートが必要になってしまいます。このようなことは出来ないので、ちょっと工夫してみましょう。
本来ならLEDは縦16行、横32列で並んでいますが、ここでは簡単にするため、縦4行、横8列にしてみました。たとえば、”A”を表示したいとします。下の図のように、”A”を表示するには、1,2,3,4を高速で繰り返します。LEDが常に”A”の形で光っているのではないのがわかりますか。高速で1,2,3,4を繰り返すことにより、人間の目には”A”が表示されているように見えます。○ーーーー消灯しているLED
●ーーーー点灯しているLED

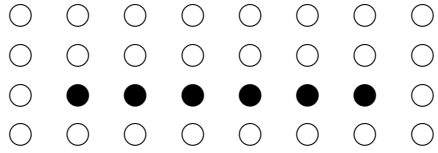
- ”A”の表示（人間にはこのように見えます）
  ○ ○ ○ ● ● ○ ○ ○
  ○ ○ ● ○ ○ ● ○ ○
  ○ ● ● ● ● ● ● ○
  ● ○ ○ ○ ○ ○ ○ ●
- 1　しかし、実際は ····



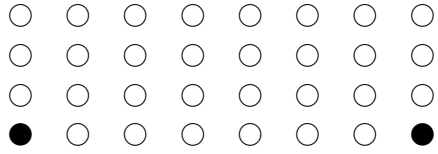
- 2



- 3



- 4



上の図の LED の配置をそのまま回路図としてあらわしてみると …

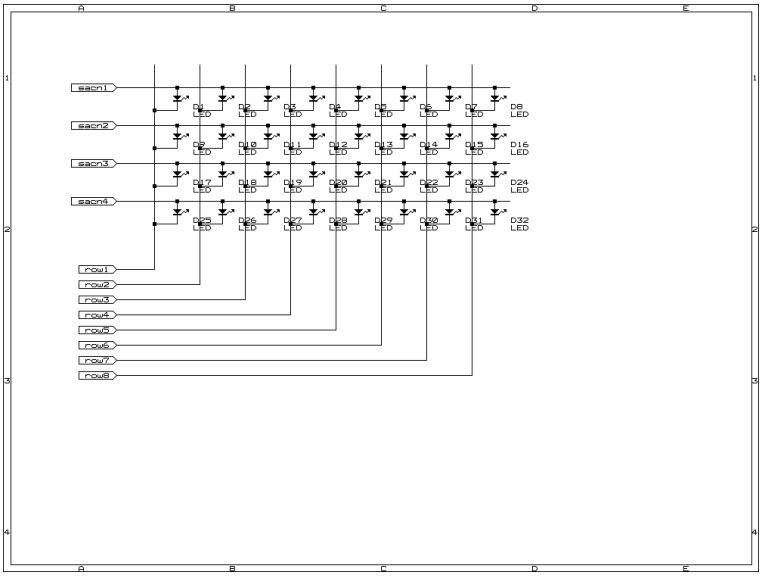


図 1.1: LED 基板の回路図

（上の図はわかりやすくするため、縦 4 行、横 8 列にしています。実際は縦 16 行、横 32 列。つまり、scan １〜scan16、row1〜row32 まであります）
回路図と”A”の表示の図をてらしあわせてみましょう。上の図の 1 のとき、回路図の scan1 と row4,5 を ON にしています。2 のとき、回路図の scan2 と row3,row6 を ON にします。次に 3 のとき、scan3 と row2,row3,row4,row5,
row6,row7 を ON にします。4 のとき、scan4 と row1,row8 を ON にします。この４つの段階を高速で繰り返せば、人間には残像として”A”が見えるわけです。

1.3 回路

次に、ディスプレイの回路について簡単に説明します。このディスプレイはプロセッサで制御されています。プロセッサーの出力ポート（出力端子）に流すことのできる電流の上限は限られているので、LED を直接つなぐわけにはいきません（LED を直接つなぐとプロセッサが故障する可能性があります）。そこで、トランジスタを使います。プロセッサの出力によって、トランジスタでスイッチングを行います。トランジスタを使うと、小さな電流で大きな電流を制御することが出来ます。回路図をみると、2 種類のトランジスタが使われていることがわかります。1 つは、LED 基板の回路図の scan1〜scan16 につながっているトランジスタで、これらのトランジスタは複数個の LED へ電流を流す必要があるので、電流をたくさん流せるものを使用しています。他方は、row1〜row32 につながっているもので、これらはあまり大きな電流を扱わないので、小さなトランジスタアレイ（複数のトランジスタを 1 個のパッケージにまとめたもの）を使っています。
また、こちらは scan 側、row 側のスイッチングをする回路です。scan,row は上の図のとおりです。以下の回路はプロセッサから送られてきた信号を増幅し、LED に流す電流を制御します。ここでは、プロセッサ部分については、難しいので、書きません。
図 3 は row1 から row32 への出力をする回路です。
プロセッサから信号がそれぞれ IN1 から IN32 に送られてきます。IN1 から 8 個 1 組となり、まとまって合計 4 個のトランジスタアレイにつながっています。トランジスタアレイは回路」の章に書いた通りのものです。トランジスタアレイは信号を受け取るとそれぞれ IN1 から IN32 までに対応している row1 から row32 に電気を流します。

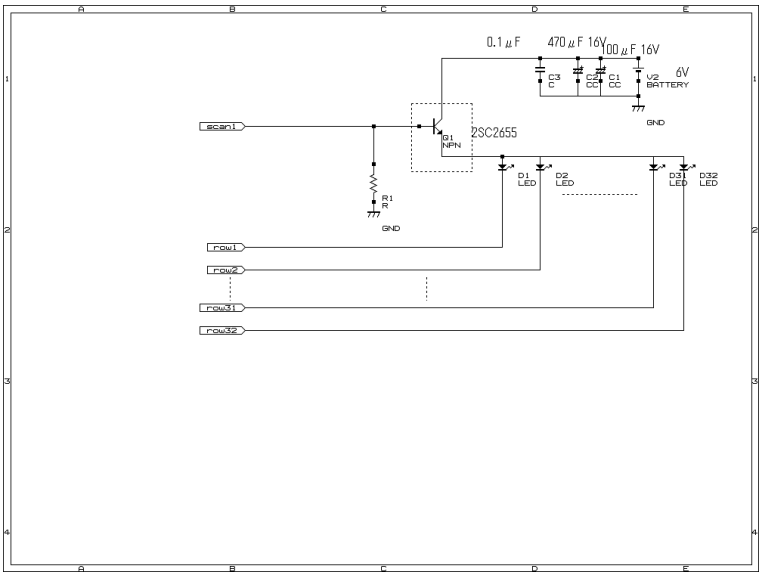


図 1.2: scan 出力用の回路図

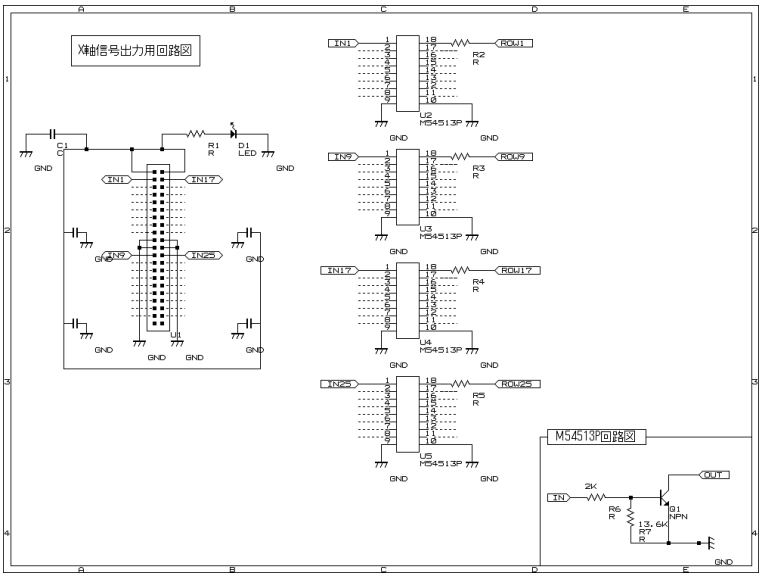


図 1.3: ROW 出力用の回路図

2 Maximaについて

電子システム工学課程 1回生 小宮山 敦史

2.1 Maximaとは

Maxima は MIT の Macsyma system を Common Lisp で実装したもので、非常に本格的な数式処理システムです。Macsyma system は数式処理システムの中では歴史ある汎用数式処理システムの 1 つで、非常に長い期間メンテナンスされてきており高い評価を得ています。Maxima は GNU Public License(GPL) のもとにリリースされており、汎用数式処理システムでは数少ない非商用 (無料)・オープンソースのもので、その完成度は商用のものに劣りません。このソフトウェアは非常に優秀であるにもかかわらず、日本ではあまり知られていません。

2.2 具体的には何が出来る？

Maxima が何をするかというと、

- 四則演算
- 数式の展開及び因数分解
- 微分、積分
- 線型方程式、3 次方程式や非線型方程式
- グラフ表示

などです。普通の計算機（電卓とか）と違って、複雑な計算とかもやってくれます。

2.3 Maximaの入手

Maxima は
http://maxima.sourceforge.net/
辺りから入手できるはずです。詳しいことは、
http://www1.bbiq.jp/kougaku/maxima.html
を参考にしてください。

2.4 Maxima の使い方

2.4.1 基本

足す、引く、掛ける、割るの記号はそのまま、$+, -, *, /$ を使います。

```
^,(,)
```

も使えます。そして入力終了時には ; を用います。

最後に『;』(セミコロン) をつける事に注意してください。Maxima は『;』が入力されるまで処理を行わないので、複数行にわたって記述が可能です。最後に『;』をつけ忘れても次の行でセミコロンを書けば OK です。

具体的には $(4+4)/(4+4)$; と入力すれば、次の行に答えの 1 が表示されます。

こんな感じ

```
(%i1) (4+4)/(4+4);
(%o1)                1
```

複雑な計算—例えば因数分解など—を計算するには関数を用います。

```
factor(因数分解したい式)
```

() の中、関数の引数部分に式を書けば OK です。

こんな感じ

```
(%i2) factor(4*x^2-4*x+1);
                                      2
(%o2)                        (2 x - 1)
```

(%i2) とか (%o2) というのはその行に対する「ラベル」です。Maxima が勝手に生成してくれるもので、このラベルは次回からの計算に使う事ができます。

次はさっきの結果、$(2x-1)^2$ を展開してみます。

```
expand(%o2);
```

とラベルを使ってみました。

```
(%i2) factor(4*x^2-4*x+1);
                                      2
(%o2)                        (2 x - 1)
(%i3) expand(%o2);
                              2
(%o3)                    4 x  - 4 x + 1
```

ようは、一度表示したものは省略して書けるということです。

2.4.2 微分、積分

微分は

`diff(被微分関数, 微分変数);`

と書きます。引数が複数あるときは , で区切ります。

```
(%i4)diff(sin(x),x);
(%o4)                cos(x)
(%i5)diff(sin(x));
(%o5)              cos(x) del(x)
```

第 2 引数を省略すると dx が付きます。当然といえば当然ですが……。
積分は

`integrate(被積分関数, 積分変数, 下限, 上限);`

などと書きます。これは、定積分の場合です。

```
(%i6) integrate(x^3,x,1,a);
                     4
                    a    1
(%o6)               -- - -
                    4    4
```

積分範囲に変数があるときは
Is a positive, negative, or zero? みたいに、正負を聞いてくることがあります。そのときは、正なら pos;、負なら neg;、0 なら zero; と返事してください。
第 3、第 4 の引数を省略すると不定積分になります。

```
(%i7) integrate(x^2,x);
                            3
                           x
(%o7)                      --
                           3
```

2.4.3 方程式

$3x + 4y = 7$ を x について解きます。

```
(%i8) linsolve(3*x+4*y=7,x);
                                4 y - 7
(%o8)                    [x = - -------]
                                  3
```

変数が多くても、その分だけ式があれば解けるはず。

$\begin{cases} 3x & + & 4y & = & 7 \\ 2x & + & ay & = & 13 \end{cases}$ を x、y について解いてみます。

```
(%i9) linsolve( [ 3*x + 4*y = 7, 2*x + a*y = 13], [x,y]);

                          7 a - 52         25
(%o9)                [x = --------, y = -------]
                           3 a - 8       3 a - 8
```

複素数が絡む場合は、

```
allroots(x^2-4*x+13=0);
```

というように、別の書き方をします。

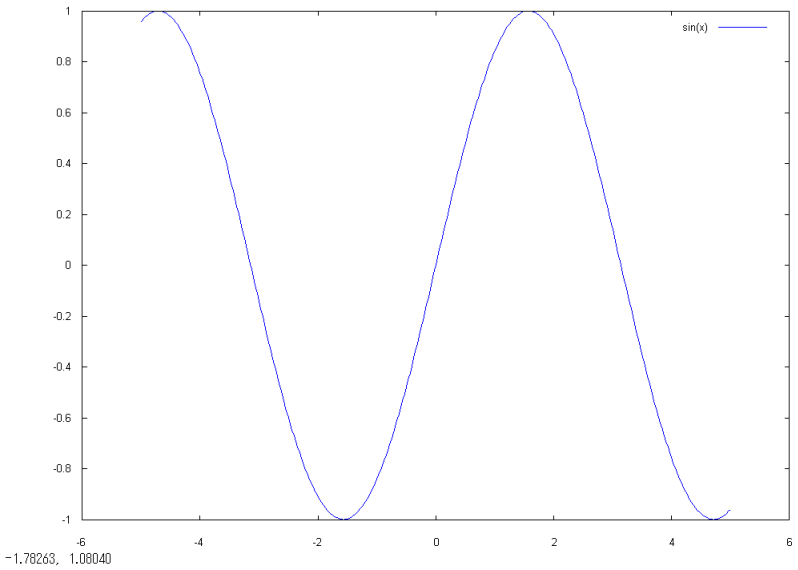
```
(%i10) allroots(x^2-4*x+13=0);
(%o10)             [x = 3.0 %i + 2.0, x = 2.0 - 3.0 %i]
```

%i は、虚数単位 i です。

2.4.4 グラフ

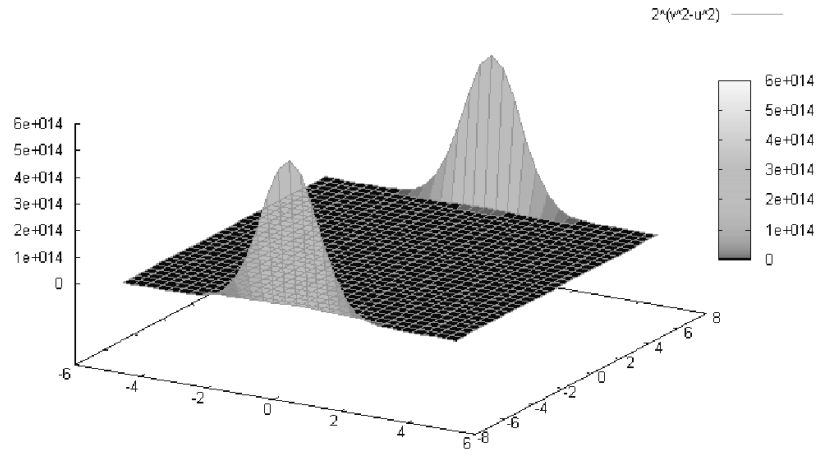
```
plot2d(sin(x),[x,-5,5]);
```

と入力すると、別窓 (gnuplot) が開いて、グラフが表示されます。これは、$y = \sin x$ のグラフを $-5 \leq x \leq 5$ の範囲で表示します。



2 次元だけでなく 3 次元も可能です。

```
plot3d(2^(-u^2+v^2),[u,-5,5],[v,-7,7]);
```



view: 60.0000, 30.0000 scale: 1.00000, 1.00000

大括弧 ([、] の二つ) に注意してください。2 番目以降の引数に [、] を使って、変数とその範囲を指定します。

別窓を閉じるまでは、次の入力は出来ないようです。

2.5 終わりに

ここで紹介できたのは Maxima の持っている機能のほんの一部です。関数定義など便利な機能もありますが、詳しくは参考資料を見てください。また、ここでは解が存在するものだけを扱いました。まあ、存在しなかったら元のままが表示されたりエラーと出るだけだったりするだけですが……。web ばかりですが、参考資料を明記して終わりとさせていただきます。

参考文献

[1] Maxima 入手元
http://maxima.sourceforge.net/

[2] Maxima 紹介
http://www1.bbiq.jp/kougaku/maxima.html

[3] Maxima 簡易マニュアル
http://www.bekkoame.ne.jp/ ponpoko/Math/maxima/ManualBook/ManualBook.html

3 Blu-ray Disc vs HD DVD

電子システム工学課程 1回生 藤井 基史

3.1 はじめに

最近ようやく一般家庭でも浸透し始めた DVD。規格の乱立による互換性の問題でかなり使い勝手の悪い状況になっていたが、-R、-RW、-RAM に対応したドライブが出てきて少し状況は改善。しかし、DVD-RAM に録画された動画は DVD プレイヤーでは再生できないなどめんどくさいことも多々あります。

このような規格の対立は VHS とベータの例などでも見られ、そのたびにユーザを悩ませ続けているのですが、そのようなことが次世代光ディスクにおいても起こっています。

今回は、その対立している Blu-ray Disc（以下 BD）と HD DVD の 2 つについて比べてみたいと思います。

3.2 次世代光ディスクの概要

まずはいきなり 2 つの規格の比較の前に、次世代光ディスクに共通な大きな特徴について書きたいと思います。

3.2.1 開発のきっかけ

もともと次世代光ディスクが開発されるきっかけになったのはテレビのハイビジョン化です。DVD はこれまでの標準画質を 2 時間録画できるように設計されていましたが、ハイビジョン画質を録画するにはあまりにも容量が足りません。そこで、ハイビジョン画質で 2 時間以上録画できるよう開発されたのが BD や HD DVD といった次世代光ディスクなのです。

3.2.2 統合への協議

VHS とベータのときのように 2 つの規格が対立することを嫌い、2005 年の春から夏にかけて東芝とソニー・松下で統合協議がありました。しかし両者の主張に妥協策を見出せず、また製品化へのスケジュールが迫っていたことから規格の統合は叶いませんでした。

3.2.3 大容量なわけ

これらのディスクの大きさはこれまでの CD や DVD と同じ直径 12cm の円盤ですが、記録方法には大きな違いがあります。それは、波長の短い青紫色レーザーを使用していることです。波長の短いレーザーを使うことにより、密度の高い記録を実現し、同じ大きさのディスクに対しても大容量のデータが記録できるわけなのです。

3.2.4 動画圧縮と音声圧縮

これらの動画圧縮方式には今まで DVD で使われていた MPEG2 のほか、新しい圧縮技術である MPEG4AVC や VC-1(Windows Madia 9) が採用されています。これらは、MPEG2 に比べて圧縮率が高く録画時間を稼ぐことができ、また高画質であるといわれています。それらもまた録画できる時間が増えた一因です。

また音声圧縮についても新しい技術が採用されています。まず非可逆圧縮のほうでは DVD で採用されていた Dolby Digital や MPEG に加えて、Dolby Digital Plus や DTS が採用されており、可逆圧縮のほうでは、従来のリニアー PCM のほか MLP や DTS-HD が採用されています。

3.2.5 著作権保護技術

BD と HD DVD の再生専用ディスクには AACS(Advanced Access Content System) という新しいコピープロテクト規格が採用されています。この規格は DVD で採用されている CSS(Content Scramble System) というアクセスコントロール技術に取って代わるもので、128 ビットのキーを持った強固な著作権保護規格です。

3.3 Blu-ray Disc(BD)

3.3.1 仕様

まず BD には今のところ BD-ROM、BD-R、BD-RE の 3 種類あります。BD-ROM は再生専用メディアであり、片面 25GB、片面 2 層で 50GB。BD-R は DVD-R のように 1 度しか記録できないタイプ、BD-RE は繰り返し書き換え可能タイプです。ファイルフォーマットは UDF のため、DVD-RAM のようにリムーバブルメディアとして扱えファイナライズも必要とせず比較的扱いやすくなっています。

3.3.2 特徴

ソニーが主導の規格であることから PLAYSTATION3 への採用が決まっています。また、DVD のときのような記録メディアの乱立の反省から、BD-R や BD-RE の規格がまず最初に作られました。そのため、初期の BD レコーダーでは BD-ROM が再生できないようです。

3.3.3 長所

BD-ROM に関しては 8 層まで実用可能とされており 1 枚で 25GB × 8=200GB を超える可能性を持っています。

また BD には著作権保護技術として AACS 以外にも ROM Mark という技術を採用しています。これは、メディアに収録されるコンテンツに識別子を付加し、それはライセンスを受けたメーカーでなければ扱えない特殊なものです。これによって原盤の複製ができなくなります。

3.3.4 短所とその克服

BD の短所は、ディスクの保護層が 0.1mm しかないため非常にキズに弱いことです。これは高密度な記録をするため、開口数の高いレンズを使用した結果、焦点距離が短くなるので保護層が薄くなっているのです。これを克服するために TDK がディスクの耐久性向上技術「DURABIS（デュラビス）」を開発。これにより、保護層の厚さが 0.6mm の DVD や HD DVD に対抗できるようになったといわれています。実際にどんなものなものかは知りませんが…。

3.3.5 主な協賛企業

シャープ、ソニー、三菱、松下、日立、パイオニア、TDK、Apple、Dell など

3.4 HD DVD

HD DVD とは High-Definition Digital Versatile Disc の略。BD とはっきり区別するためにカラーは赤で統一されています。

3.4.1 仕様

種類は再生専用の HD DVD-ROM、追記型の HD DVD-R、書き換え可能な HD DVD-Rewritable があります。記録容量はそれぞれ、15GB、15GB、20GB と少し違っています。またそれぞれに 2 層タイプや両面タイプも開発されており、実際に書き込める容量はもっと多くなるでしょう。

3.4.2 特徴

もっとも大きな特徴は、DVD の規格を制定した DVD フォーラムに唯一認められた規格で DVD との互換性が高いことです。そのため、現在の DVD メディアの工場を 5 分で HD DVD メディアの工場にすることが可能なようです。そのため、DVD から HD DVD への移行が容易であり、メディアの生産コストも BD に比べてかなり安くなると言われています。

また、HD DVD-Video 規格では 3.2.4 で述べたコーデックのうち、DTS HD 以外のすべてを必

須としており、BD との違いを出しています。

3.4.3 長所

HD DVD では DVD との互換性を生かしてツインフォーマットディスクというのが策定されています。これは片面の 2 層ディスクで、レンズに近い層が赤色レーザー用（DVD)、奥側が青紫色レーザー用（HD DVD）です。これにより、DVD プレイヤーでも、HD DVD プレイヤーでも、それぞれのフォーマットを読み取ることが可能になります。この技術が DVD から HD DVD への橋渡し的な役割をするため BD よりもこの点では有利だといわれています。

また、Microsoft の次期 OS である Windows Vista では、HD DVD が正式にサポートされています。

3.4.4 短所

HD DVD は BD に比べて劣る点がいくつか指摘されています。一つには容量があります。BD-ROM が片面 25GB であるのに対して HD DVD は 15GB です。HD DVD 陣営は当初新しい圧縮技術で補えると主張していましたが、その技術をはじめは採用していなかった BD 側も採用したため、主張が通らなくなりました。また、MPEG2 が DVD 時代からの蓄積があり信頼性がある一方、MPEG4AVC や VC-1 はまだ映画配給会社の信頼を得られておらず、結局高画質を求めるにあたって容量は多いほうがいいわけなのです。

3.4.5 主な協賛企業

東芝、NEC、サンヨー、富士通、キャノン、Intel、Microsoft など

3.5 勝負のゆくえ

さて、この対立どちらが勝つのでしょうか？PS3 にも搭載される BD？それとも、Windows Vista においてサポートされる HD DVD か？現在、BD と HD DVD の両規格に対応したドライブが開発中だとか言われていて、どちらの規格も残るかもしれません。また、ビデオ用では BD、データ用では HD DVD などと住み分けが起こるかもしれません。しかし、この戦いのゆくえを大きく左右するのは映画配給会社だといわれています。現在 HD DVD のみを支持するハリウッド企業はユニヴァーサル・ピクチャーズのみで、ハリウッド映画ソフトの売上シェアの約 8 割が BD を支持しています。HD DVD 陣営の NEC が BD ドライブを搭載したパソコンを発売するなど、BD 優位とも言われています。

3.6 終わりに

まだ、自分で使ってみたことがないので、「もし自分が買うとしたらどちらを買うか？」というのを決めるような気持ちで調べてみました。結局、自分の中の結論としては、「流行ってきたほうでいいや」っていう答えになっていない答えにたどり着いてしまいました。もう少し様子を見たほうがいいようです。

参考文献

[1] HD DVD プロモーショングループ
http://www.hddvdprg.com/jpn/index.html

[2] DVD Forum
http://www.dvdforum.org/forum.shtml

[3] ソニー ブルーレイディスク ポータルサイト
http://www.sony.jp/products/Consumer/BD/

[4] ブルーレイジャパン
http://www.blu-ray.jp/

4 mixiについて

情報工学課程 1回生 東川 知生

4.1 はじめに

mixi はＳＮＳ (ソーシャルネットワークサービス) の一種です。

ＳＮＳとは何か？

例えて言うならば、ネットワーク上の交流パーティーみたいなものです。つまり、色々な人と一つの話題を話し合ったりそのパーティーで古くからの知り合いに出会ったり同じ考え方や同じ趣味の人とも知り合えるのです。

4.2 参加方法

ですが、mixi は会員制のパーティーです。会員以外は利用することが出来ません！

では、どうすれば利用できるのか？

既に会員となっている人に招待されればいいんです。常連さん以外お断りのクラブのシステムに似ています。この方法のほうが、人と人が繋がりやすいからなんでしょうか？

とにかく、会員に招待 (招待メールが送られてくる) されれば次に自分のプロフィールを登録します。その時に、本名とハンドルネームと E-mail アドレスを要求されます。E-mail アドレスは他の人には公開されないので、安心して打ち込めます。ただ、本名は他の人に公開されるので、偽名を使いましょう。そうしないと、セキュリティ上安全とは言えなくなります。若し、実名を書き込んでしまったとしても安心です。あなたの mixi のページのトップにある「プロフィール」のボタンをクリックすれば、「プロフィール変更」というところがあるので、そこをクリックしましょう。そうすれば、本名は変えることが出来ます。

これで、ようやく mixi の一員になれるわけです。

4.3 利用の仕方

では、mixi に入って実際どんな事をするのか？基本的に日記を書きます。その日記をマイミクに登録している人や他の人に公開します。そして、そういう人たちから日記に対してのコメントを貰います。

ここで、上記に出てきた「マイミク」について説明します。

「マイミク」とは「マイミクシィ」の略です。そして、マイミクとは mixi 上でのあなたの友人と思ってください。では、その登録方法を説明します。

まずは、ホーム上部にある「友人を検索」等を使ってあなたの友人を探してください。見つけたなら、次にその人の mixi 上のページに飛んでください。そして、上部に「マイミクに登録」という青字の箇所をクリックします。そうすると、その人にメッセージを送れます。その、メッセージで自分が○○であると表記して友人に自分だと認識させましょう。これで、自分がやる全ての手順は終了です。後は、友人が認めるのを待つだけです。

ちなみに、拒否されたとしても自分は気がつきません。

筆者は友人と合意して実際にやってみましたが何のメッセージも出ませんでした。これは mixi 側の人間関係を壊さないようにという配慮なんでしょうか？

さて、話を戻しますが次に何をするのか？

何か共通の話題で語り合いたいと思うなら、そういう話題だけを話す「コミュニティ」に参加する。そういうのが無ければ自分で作りましょう。言ってみれば、ネット上の掲示板と同じものです。具体的にどうすればいいのか？コミュニティに参加するのはとても簡単です。そのコミュニティのトップページから「このコミュニティに参加」という青字の箇所をクリックすればいいだけです。作るのは、mixi の指示通りにすれば出来ます。

また、自分の好きなものを他の人に紹介したいならば「レビュー」を活用しましょう。

4.4 おわりに

以上で、mixi の基本的な機能等の説明は終わりです。mixi には他にもたくさんの機能があります。これを読んだ機会に、mixi に入ってみるのはどうでしょう？

5 はじめてのSchemeインタプリタ

情報工学課程 1回生 湯浅 信吾

prologue

　終わりまで残り 914 ページ。
そこで立ち尽くす。
「はぁ」
ため息と共に空を仰ぐ。
その先に CommonLisp 第 2 版の終わりがあった。
誰が好んで、こんな分厚い本を書いたのか。
長い文字列が、悪魔のように伸びていた。
「はぁ…」
別のため息。俺のよりかは小さく、短かった。
隣を見てみる。
そこに同じように立ち尽くす女生徒が居た。
「Lisp は、好きですか」
いや、俺に訊いているのではなかった。
「わたしはとってもとっても好きです。
　でも、Common Lisp は…大きすぎるんです。
　言語の仕様とか、インタプリタの仕様とか、ぜんぶ。
　…ぜんぶ、大きすぎるんです。」
たどたどしく、ひとり言を続ける。
「それでも、この言語が好きでいられますか」
「わたしは…」
「作ればいいだろ」
「えっ…？」
少女が驚いて、俺の顔を見る。
「言語の仕様とか、インタプリタの仕様を
　作ればいいだけだろ。あんたにとっての Lisp は
　 Common Lisp だけなのか？　違うだろ」
そう。
何も知らなかった無垢な頃。
誰にでもある。
「ほら、作ろうぜ」

俺たちは作り始める。
　　へぼい、へぼいインタプリタを

5.1 はじめに

5.1.1 おことわり

これは「坂道の下で出会った少年が少女が、逆境を乗り越えてインタプリタを完成させる、発売日がやたらと伸びる物語」ではありません。タイトルに『はじめての』がついているからといって、アレな趣味 の人向けな文章でもありません。

これはろくに勉強せずに Lisp インタプリタを作り始め、気がつけば Scheme インタプリタに変り、自分に使いやすいようにという大義名分の下、自分しか使えないであろうインタプリタを作った記録を元に Scheme インタプリタの作り方を書いたものです。本物の Lisp や Scheme を勉強したい人は内容を信じないように。また、これを読んで Lisp に失望したりしないように。

Lisp を知らない人にはわかりにくく、知ってる人にはまどろっこしいのは仕様です。**最低限 S 式を読めないとまったく分からないと思います。**この文章を鵜呑みにして人前で恥をかいても当方は一切責任を負いません。

5.1.2 ここで作るもの

先に完成品について少し触れておきます。言語仕様は Schemeっぽくしましたが、マクロは使えません。代わりに引数を評価しない関数 (FSUBR と FEXPR) に呼び出し元の環境を送ることによって、代用しています。末尾再帰、GC、コンティニュエーションは実装してます。用意している関数や特殊オペレータは限られていますが、自分で関数を定義することによって大抵のものは作れます。最後に、ここで作成するインタプリタ[1]の名前ですが、”**murasaki**” です。

5.1.3 3 分で分かる (?)Scheme 入門

Scheme の説明が全く無しではあんまりかと思ったのでちょっとだけ説明を書いておきます。Scheme とは Lisp という言語から派生した関数型言語[2]です。Scheme では多くのデータをリストで表します。リストは **(要素 1 要素 2 ... 要素 n)** という風に括弧を使って書きます。Scheme インタプリタは入力された式 (=プログラム) を評価 (=実行) し、その結果を返します。1, 2, 3 といった即値を入力すると、そのままの値が返され、x, y, z といった変数が入力されると、それの中身の値が返されます。
(関数 引数 1 ... 引数 n) と書くと、関数の呼び出しとして扱われます。このように、式も内部的にはリストで表現されます。

5.2 データの設計

5.2.1 データ

murasaki の扱うデータは全て 64bit(またはその倍数) の固定長で、データ自身にデータの種類を判別するためのタグ (5bit) をつけた、いわゆる「オブジェクト・タグ方式」というものを採用しました。[3]また、GC のマーク付けのために 1bit を使用し、以上 6bit は固定なので、残り 58bit をデータの種類に応じて使います。このような、64bit から成るデータを総称し、**オブジェクト**と名づけ、実装においては構造体 LObject というもの[4]に値を突っ込みました。

---

[1] Scheme の仕様を無視しているので、一つの言語ともいえるかもしれません。
[2] 副作用があるから正確には…って話は無しで (笑)
[3] データを指すポインタの方にタグを付ける方式を「ポインタ・タグ方式」というそうです。
[4] 実体は単なる 1byte × 8 要素の配列です。

5.2.2 コンス

リストの元となるデータ型であるコンスは murasaki では、

- GC のための領域 (1bit)
- データ型を現す領域 (5bit)
- CAR 部 (29bit)
- CDR 部 (29bit)

からなる 64bit の構造体で構成されています。データ長は 8bytes 固定なので、オブジェクトのアドレスは必ず 8 の倍数となるため[5]、3bit 右シフトできるので[6]、29bit で 32bit のアドレス空間を表現できます。

数値

数値は必ず独立した一つのオブジェクトであり、コンスの CAR 部や CDR 部に即値として入れられることはありません。理由は**面倒だったから**です (笑)

シンボル

シンボルは、オブジェクトの後ろから 56bit(7bytes) をつかって 7 文字を格納し、7 文字以上の場合は、後ろに『シンボルの”続き”型』のオブジェクト[7]を作って、同様に 7 文字格納し、まだ足りない場合は同様にします。終端にはヌル文字 (0x00) を置いています。例えば、”long-symbol-name”という名前のシンボルを格納する場合下のようになります。

```
      64bit     64bit     64bit
    [long-sy] [mbol-na] [me     ]
```

この 3 つのデータは必ず連続して配置される必要があります。

5.2.3 メモリ管理

murasaki では、オブジェクトを保持するために、連続した領域を起動時にまとめて取得します。あとは、オブジェクトを生成するたびにその領域を端から使っていきます。評価の過程で生じるごみは、後述する GC が回収します。

5.2.4 環境

変数の名前と値を対応付けるものを環境といいます。環境は単純に連想リスト[8]を利用しました。とりあえず、大域環境 (グローバル変数用の環境) として以下のようなリストを設定しておきます。

```
((#t . #t) (#f . #f) (nil . nil))
```

ここまで作れば評価器 (プログラムを実際に走らす部分) が作れます。

---

[5]始点が 8 の倍数とは限らないので、実際には始点のアドレスをオフセットとして保持する必要があります。
[6]8 の倍数なら xxx...xxx000 となるので、右端の 0 の部分を省けるという仕組みです。
[7]シンボル型とまったく同じ構造で、タグだけが違うというものです。
[8](名前 . 値) の対がつながったリストのことです。

5.2.5 評価器

評価器には引数として**式**と**環境**を渡し、以下のように動かします。

- 式が数値ならそれを返す
- 式がシンボルの場合は環境から対応する値を取り出して返す
- 式がコンスの場合は関数呼び出し (後述)

5.2.6 トップレベル

- 式の入力を待つ
- 入力された式と大域環境を評価器に渡す
- 評価器の返した値を表示

これをループさせれば、インタプリタの出来上がり。入力された S 式は、そのままの構造を持つリストに変換し、表示処理ではその逆をやるだけです。ここは**根性**で書くだけだと思うので、特に何も書きません。実際に作ってみようと思う方は頑張ってくださいね。

5.3 関数

5.3.1 SUBR/FSUBR

murasaki で扱う関数は大きく分けて、インタプリタ側に組み込む関数と、Scheme で定義した関数に分かれます。組み込み関数のうち、引数を評価するものを **SUBR**、引数を評価しないものを **FSUBR** と呼びます。SUBR には引数のリストが渡しますが、FSUBR には引数のリストと呼び出し元の環境も送ります。SUBR である car[9], cons、FSUBR である quote[10]は以下のように定義できます。

```
LObject *car(LObject *s)
{
    s = CAR(s); /* 第 1 引数を取得 */
    return CAR(s);
}

LObject *cons(LObject *s)
{
    LObject *ret = CreateCons();
    LObject *p = CAR(s); /* 第 1 引数を取得 */
    SETCAR(ret, p);
    p = CDR(s);
    p = CAR(p); /* 第 2 引数を取得 */
    SETCDR(ret, p);
    return ret;
}

LObject *quote(LObject *s, LObject *env)
```

[9]car、cdr は実際は nil に対して nil を返す必要があります。
[10]quote は eval の内部で処理させる例をよく見かけますが、murasaki では FSUBR として処理させてます。cond も同様です。

```
{
    return CAR(s);
}
```

5.3.2 環境

car や cdr を関数として認識できるように、大域環境をいじる必要があります。まず、SUBR 型/FSUBR 型のオブジェクトというものを作ります。中身は後ろから 32bit に SUBR/FSUBR の関数のアドレスを入れただけのものです。あとは、それのオブジェクトで car や cdr という名前のシンボルを束縛すれば大域環境の出来上がり。

5.3.3 SUBR/FSUBR の呼び出し

関数を呼び出せるようにするために、評価器を **Eval** と **Apply**[11] の二つの関数に分けます。二つの定義は以下の通りです。

Eval(form, env)

- form が数値ならそれを返す
- form がシンボルの場合は env から対応する値を取り出して返す
- form がコンスの場合は Apply(Eval(CAR(form)), CDR(form), env) を呼び出しその値を返す

ただし、CAR(s), CDR(s) はコンスの CAR 部、CDR 部を取り出す関数。

Apply(fn, args, env)

- fn が SUBR の場合は args の値を evlis(args, env) に書き換える
- fn から呼び出す関数のアドレスを取り出す
- SUBR なら args を引数に、FSUBR なら args と env を引数につけ関数を呼ぶ

ただし、evlis はリストを受け取り、各要素を評価したリストを返す関数。

ここまで作ると (car '(a b c)) とか、 (+ 1 2) とかが評価できるようになります。[12] 入れ子構造を持った (car (cdr '((a b c) (d e f)))) とかも評価できます。感動ものですね (笑)

5.3.4 EXPR/FEXPR

さて、これで自由に組み込み関数を作れるようになったので、今度は Scheme 側で関数を作れるようにしましょう。Scheme で書かれ、引数が固定個であり、呼び出し前に引数が評価される関数を **EXPR** といいます。同様に、引数が不定個であり、呼び出し前に引数が評価されない関数を **FEXPR** といいます。今度はこの二つを定義/呼び出しできるようにします。

5.3.5 begin

EXPR を実行するための足がかりとして、begin という FSUBR をつくります。begin は引数として、1) 複数の式、2)begin を呼び出した環境 を受け取ります。そして、式を順番に評価し、最後の式の評価の結果を返します。これ自身は単純な関数ですが、EXPR/FEXPR を動かす上では重要な役割を果たします。

---

[11] これは Scheme 側から呼び出せる組み込み関数ではないので先頭を大文字にして区別しました。組み込み関数としての eval/apply は先頭を小文字で表します。

[12] ただし 'hoge を (quote hoge) に変換するのが案外めんどくさいのですが。

5.3.6 lambda

EXPR を定義するために lambda という FSUBR をつくります。lambda は引数として、定義する関数の式のリスト[13](forms)、仮引数のリスト (args)、lambda を呼び出した環境 (env) を受け取ります。そして、これを以下のようなリストにして返します

```
(forms args env)
```

ただし、普通のリストと区別するために先頭のコンスのタグを別のものに書き換え、それを EXPR オブジェクトと呼ぶことにします。

5.3.7 EXPR の呼び出し

引数のことを無視すれば、EXPR オブジェクトの forms をとりだし、begin の引数として送れば EXPR が呼び出せそうですね。[14] さて、その引数ですが、evlis で実引数を評価したリストを得た後に、EXPR オブジェクトの args を取り出し、両方のリストから要素を一つずつ取り出し、対を作ったものをリストにします。

```
[例 : ((lambda (x y z) (+ x (* y z))) 1 2 3)]
仮引数 (x y z)
実引数 (1 2 3)
生成物 ((x . 1) (y. 2) (z . 3))
```

そして、このリストの後ろに EXPR オブジェクトの env を取り出したものをつなげ、それを env2 とします。最後に begin(forms, env2) を呼び出して、その値を返せば EXPR の呼び出しは終わりです。これらの処理は全て Apply の中に書けばいいでしょう。

環境から値を探すときは先頭から探すので、もし、大域環境などで x, y, z といった仮引数と同じ名前の変数が定義されていても、引数の値の方が先に見つかり、こちらの値が使われます。

これにより、引数を正しく扱うと同時にレキシカルスコープも実現できました。クロージャも作り放題です。一応うまく動くんですが、思いつきでこんな設計にしたので本当にいいのかどうかよくわかりませんが (笑)

5.3.8 FEXPR

FEXPR を作る手順は EXPR を作るのとほぼ同じです。FEXPR を定義するために nlambda という FSUBR をつくります。nlambda も引数として、定義する関数の式のリスト (forms)、仮引数のリスト (args)、lambda を呼び出した環境 (env) を受け取り、それをリストにして返します。先頭のタグは EXPR オブジェクトとは別のものを付け、それを FEXPR オブジェクトと呼ぶことにします。ただし、仮引数の数は 2 個と固定です。[15]

5.3.9 FEXPR の呼び出し

これもまた、EXPR とほぼ同じですが、引数の扱いが若干異なります。まず、実引数は評価せず、そのままのリストの形で第一引数を束縛します。次に、現在の環境 (=呼び出し元の環境) で第二引数を束縛します。

```
[例 : ((nlambda (s a) s) hoge foo)]
仮引数 (s a)
実引数 (hoge foo)
生成物 ((s . (hoge foo)) (a . 呼び出し元の環境))
```

で、これの後ろに FEXPR オブジェクトの env をつなげ、それを env2 と名付け、begin(forms, env2) を呼び出せば終わりです。これも EXPR と同じく Apply の中に書けばいいだけです。

---

[13] リストなのは複数の式を定義できるようにするためです。
[14] まあ、そのように設計したから当たり前なんですが (笑)
[15] 第一引数に実引数のリスト、第二引数に呼び出し元の環境が束縛されます。

「呼び出しもとの環境」とは、FEXPR を呼び出したときの環境であり、nlambda を呼び出した環境とは別ものなので、そこに注意してください。[16]

ここで、Eval や Apply を組み込み関数として Scheme 側から呼び出せるようにすると、[17]マクロのようなものが書けるようになります。もう…マクロなんてなくてもいいよね？

5.3.10 束縛を作る

さて、せっかく関数を作れるようになったので、それを束縛できるようにした方が便利でしょう。という訳で、define と set! を作ります。

define は FSUBR で、引数としてシンボルと任意の値を受け取り、それを対にしたものを現在の環境リストの先頭に付け加えます。第一引数であるシンボルは評価しませんが、第二引数は評価するということをお忘れなく。こうして関数を束縛できて、何がうれしいかというと、これだけで再帰が書けるようになるんです。関数定義のところで面倒なものを書いただけの価値があったということですね (笑)

さて、束縛を変更する set!ですが、これも FSUBR にしてもいいんですが、実は FEXPR でもかけたりします。[18]FEXPR の価値を感じれる瞬間です。だから……マクロは作らないって事でいいですよね (笑)

```
;set!の定義の例（エラー処理なし）
(define set!
  (nlambda (s env)
    (set-cdr! (assoc (car s) env)
              (eval (cadr s) env))))
```

5.3.11 ここまででできること

条件分岐を行う cond も FSUBR として定義すれば比較的簡単に書けます。さらには、cond があれば FEXPR として if を定義することも可能ですし、lambda を使うことによって let を FEXPR として定義することさえも可能です

つまり、ここまで作ればほとんど完成なんです。

……『**継続**』というものさえなければ……

5.4 ごみ集め

5.4.1 Garbage Collection

メモリ管理の方式については最初に少々述べましたが、ここでは実装した GC について、簡単に書きます。

murasaki ではせっかくだから GC と一緒にコンパクションもやろうと思って、Stop and Copy 方式の GC を使うことにしました。[19]

この GC は下準備、マーク付け、コピーの三つから構成されています。マーク付けとコピーはメモリが不足したときのみ行います。

下準備とは

- オブジェクトが使用するメモリ空間を A と B の二つに分け、始め A を使用する

[16]FEXPR を束縛せずにその場で呼び出した場合は二つの環境は同一ですが、FEXPR を作成後束縛し、作成時とは別の環境で FEXPR を呼び出した場合、二つの環境は別物となります。

[17]組み込み関数としての Eval/Apply は小文字で示します。なお、処理内容は Eval/Apply を呼び出すだけです。

[18]ただし、コンスの参照先を書き換える set-car!, set-cdr!を用意しておく必要があります。

[19]けど、マーク付けをするところは完全に Mark and Sweep なんで正確には Stop and Copy ではないです。Mark and Copy とでも言うのでしょうか？

- インタプリタ側の変数[20]がオブジェクトを参照するたびに、その変数のアドレスをスタック[21]S に積む
- オブジェクトを参照した変数の寿命が尽きると、S からそのアドレスを取り出す

マーク付けとは

- S に入っているアドレスの中身が参照しているオブジェクトにマークをつける
- そのオブジェクトが参照している[22]オブジェクトにもマークをつける

コピーとは

- A にあるオブジェクトを全てたどり、マークが付いているもののみ B にコピーする
- コピー元の内容をコピー先のアドレス (”引越し先”) に書き換える
- S に入っているアドレスの中身の参照先を”引越し先”に書き換える
- A と B の役割を交換する

以上です。はっきりいって結構面倒です。素直に Mark and Sweep を使って置けばよかったと後で後悔しました (笑)

5.5 継続

5.5.1 評価器を射ち堕とした日

> 主よ、私は評価器を殺めました。
> 私は、この手で大切な評価器を殺めました。

Scheme には「継続をファーストクラスオブジェクトとして扱える」という大きな特徴があります。これだけでは何のことやらさっぱりわかりませんが、簡単に言うと、関数を越えた大域的な goto を使えるということです。

これは C の setjmp/longjmp と非常に似ています。しかし、longjmp が使えるのはスタックの状態が setjmp を呼び出したときから保たれているとき—即ち、スタックが浅くなる時だけです。それに対し、継続を利用したジャンプはスタックが深くなる時でもできます。

例えば、A, B を関数とすると、
A → B → (setjmp)
という順番に関数を呼んでいくと、関数 B から抜けてからは longjmp は使えません。なぜなら、 A → B という順番に呼び出されたという情報は、どこにも保存されていないため再現できないからです。しかし、Scheme の継続を利用すると失われたスタックを再現し、より汎用的なジャンプが実現できます。

非常に分かりにくい説明でしたが、継続の概念を実装するには、関数呼び出しのスタックを完全に記憶できる必要があります。現在、関数呼び出しは Eval と Apply によって制御され、このままではスタックを触るどころか、スタックの状態を取得することすらできません。つまり、**現在の仕様ではどうやっても実装できない**ということです。

> 「避けられぬ終焉は　せめて作ったこの手で…」

5.5.2 ラベル

さて、Eval/Apply という二つの関数の存在を抹消したら、新たな評価器を作らなければなりません。とりあえず、Evaluator という名前にでもしておきましょう。

そこに、過去の Eval と Apply を**ラベルとして**定義します。お互いの呼び出しは goto で行います。詳しくは下のソースをご覧ください。

---

[20] インタプリタ内部で使われるローカル変数やグローバル変数のことです。
[21] ポップせずに中身が見れるようにする必要があるので、これも正確にはスタックではありません。
[22] 例えばコンスの CAR 部と CDR 部。それに加えて EXPR/FEXPR オブジェクトなども参照を持っているので注意してください。

```
#define EVAL_LABEL   100
#define EVAL_LABEL_1 101
/* 中略 */

LObject *Evaluator(LObject *s, LObject *env) {
  /* 中略 */
eval_label: /* a1=s, a2=env と割り当てられる */
  /* 中略 */
  PushVar(a1); /* a1 の値を保存 */
  a1 = CAR(a1);
  PushLabel(EVAL_LABEL_1); /* goto の後の戻り先を設定 */
  goto eval_label;
eval_label_1:
  a1 = PopVar(); /* 保存した a1 の値を取り出す */
  a3 = a2;
  a2 = CDR(a1);
  a1 = ret_val; /* "上の "goto eval_label"によって得られた値 */
  PushLabel(EVAL_LABEL_2);
  goto apply_label;
eval_label_2:
  goto return_label;
  /* 中略 */
apply_label: /* a1=fn, a2=args, a3=env と割り当てられる */
  /* 中略 */
return_label:
  /* スタックから値を取り出し、対応するラベルに goto で飛ぶ。 */
  /* スタックが空になった場合、関数を抜ける。 */
}
```

長い上に分かりにくいソースで申し訳ございません。重要なのは二つ。一つは、ローカル変数、関数の呼び出しによるスタックの管理を自分でやるということです。もう一つは、上のソースからは分かりませんが、このスタックの管理をリスト (コンスオブジェクト) を用いてやるということです。[23]このリストを返すと、スタックの状態を Scheme 側でただのオブジェクトと同様に取り扱えるようになります。例によってタグを付け替え、それをコンティニュエーションオブジェクトと呼ぶことにします。

5.5.3 call-with-current-continuation

言葉の説明だけでは分かりにくいので、実際に継続を用いた Scheme のプログラムを見てみましょう。

```
(define cont #f) ; 適当な初期値を代入
(+ 1 (call/cc (lambda (k)
                (set! cont k)
                1))) ; 結果=>2
(cont 2)             ; 結果=>3
```

まず、 + が関数と解釈され、引数の評価が始まります。call/cc を評価した時点で現在のスタックの状態 (= コンティニュエーションオブジェクト) が生成され、call/cc は、それを引数として lambda により定義された

[23] リストの先頭に新たなものを積むようにスタックを作るのがコツです。この先頭へのポインタは適宜書き換えますが、スタックを成すコンスの参照先を書き換えてはいけません。保存したスタックが書き換えられてしまうからです。また、ラベルは適当に割り当てた数値として格納しておくといいでしょう。

無名関数を呼び出します。そこで、コンティニュエーションオブジェクトでグローバル変数 cont を束縛し、1 を返します。これで + の引数の評価が終わったので、 + が実行され、結果として 2 が返ります。

問題は次の行です。束縛したコンティニュエーションオブジェクトに引数を付けて呼び出しています。こうすると、現在のスタックの状態が破棄されて、[24]スタックの状態はコンティニュエーションオブジェクトが生成されたときのものになります。(つまり、 + の引数を評価**しようとしていた**状態です。) そして、call/cc の戻り値が引数である 2 であるかのように振る舞い、結果として + の引数の評価後の値が (1 2) となるため、3 が返るというわけです。

これを実装するために評価器に追加するのは、call/cc が評価[25]された時点でコンティニュエーションオブジェクトを作成するという処理、call/cc のために goto した apply_label にて、そのコンティニュエーションオブジェクトを引数として、call/cc の引数である関数を呼び出すという処理。そして、コンティニュエーションオブジェクトに引数を付けて呼び出した場合、スタックの状態を書き換え、引数を返り値にするという処理です。

面倒なようですが、案外簡単にかけます。どちらかというと、今までの Eval/Apply を壊してラベルをたくさん付ける方がよっぽど大変です。

5.5.4 修正は何処までも続いてゆく → 世界の果てを目指して

さてさて、評価器を書き換えたことによる修正はまだ続きます。FSUBR の中には cond、begin のように内部から Eval を呼んでいたものがあります。これらをそのまま使用することはできないので、内部から Eval を呼んでいた FSUBR は**全て** Evaluator の内部のラベルとして書き換える必要があります。物凄く面倒です。異常なまでに面倒で途中で泣きたくなりました。しかし、これさえ作れば一応簡単な Scheme インタプリタとしては一応完成です。

5.5.5 継続まとめ？

「Scheme の継続の実装は多くの人が苦労し、各々の実装をしていった。」と後に聞きましたが、murasaki での継続の実装ははっきり言って**非常に面倒**なものです。先人が私と同じ実装をしたのか、全く違う実装をしたのかは定かではありません。[26]しかし、この方式ではコンティニュエーションオブジェクトを作るのにコストが全くかかりません。また、それを元にスタックを書き換えるのもコストが全くかかりません。ここだけが自慢です。……ここだけが。

5.5.6 末尾再帰

これまで末尾再帰に触れませんでしたが、関数呼び出しのスタックが自由に触れるようになった今、末尾再帰を goto に書き換えるのはそれほど難しくはないでしょう。ご自由にお作りください (笑)

murasaki では begin の最後でごにょごにょやってるだけです。

---

[24]Scheme 側のオブジェクトなので放って置いたら GC が勝手に回収してくれます。

[25]apply_label でやってもいいのですが、コンティニュエーションオブジェクトに引数を付けて呼び出す処理のことを考えると、eval_label でやった方が楽です。

[26]色々調べたところ、いくつか実装方法を見ましたが、私と同じ方法をとっているものは見つかりませんでした。

epilogue

「もし…
　Scheme に人と同じように、意志や心があるとして…
　そして、それを使う人たちインタプリタを作らせようって…
　そんな思いで、いるとしたら…
　murasaki が完成したという奇跡も、そんな Scheme のしわざかもしれないです」
いや…奇跡はこれからたくさん起こるのだろう。
そんな気がしていた。
「でも、それは奇跡じゃないですよね
　Scheme を大好きな人が、インタプリタを作り…
　人を好きな Scheme が、人にインタプリタを作らす…
　そんな、誰にでもある感情から生まれるものです
　murasaki だけじゃないです
　どんな Scheme だって、そうです
　わたしたちは Scheme を愛して、Scheme に育まれてるんです
　そう思います」
「なぁ…
　Scheme は、大きな家族か
「はい。Scheme も人も、みんな家族です」
「そっか…」
「だんご大家族です」
ああ…そうか。
そうだったのか。

ずっとずっと歌っていよう。
今日からの思い出を…
このインタプリタと…
Scheme のために。

　—完—

6 我が家のファイル共有事情

電子情報工学科 2回生 林 奉行

6.1 はじめに

ファイル共有と言ってもいろいろありますが。今回はあるコンピュータにあるファイルをネットワーク上の別のコンピュータが「共有」出来るしくみ、といった意味で用います。それを WAN [1]越しにできたらいいなというのが趣旨です。結局、そのような環境を手に入れることが出来たので、その体験でもお話ししようと思います。

6.2 その方法

方法はいろいろあると思います。例えば、FTP や WebDAV [2]。WWW もある種のファイル共有であると考えられます。

しかし、それらは共有しているファイルに影響を及ぼそうとすると一度ローカルに保存したファイルに編集を施し、再びそのファイルを共有ファイルに上書きするという作業が必要になります。共有しているファイルへの操作が限られているのです。

そこで、共有しているファイルをローカルなファイルと同じように扱いたいという要求が出てきます。

このような要求に応えるのが NFS や CIFS によるファイル共有です。

NFS(Network File System) 多くの UNIX 系 OS で動作する分散ファイルシステム。NFS サーバが動作しているコンピュータのファイルシステムをマウントして使用することが出来る。

CIFS(Common Internet File System) Windows 上でネットワークを通じてファイルやプリンタの共有を実現するプロトコルである SMB を拡張し、Windows 以外の OS やアプリケーションソフトでも利用できるよう仕様を公開したもの。

これらを使用すれば、共有ファイルの一部だけを読み書きしたり、ローカルのストレージにコピーせず実行したりといったことが出来るようになります。このほかにも ASF などもあるらしいのです。(どんな物か全く知りません)。

6.3 VPN 接続する

上に挙げたような便利な環境を WAN 越しに利用したい、などと思うのは当然なことだと思います。しかし、WAN を経由すると言うことは NAT や FireWall を通り越して行かなくてはならない場合があるということです。複数のポートを使用するファイル共有プロトコルの場合、その全てについて設定する必要があって大変ですし、NAT の内側にある 2 つのコンピュータとファイル共有したい場合もあります。

---

[1] Wide Area Network: 広域通信網、屋外の公衆回線を通ってという意味で使っています。

[2] HTTP を拡張し、クライアント (Web ブラウザ) から Web サーバ上のファイルやフォルダを管理できるようにした仕様。

そんな問題を一気に解決する方法があります。VPN 接続を張って相手方の LAN の一員になれば良いのです。NFS も SMB も使い放題です。ついでに、データを暗号化してくれたり、圧縮して効率の良い通信をしてくれたりするらしいです。

今回は一番簡単そうという理由で、PPTP で VPN 接続を張る事にしました。私の家には FreeBSD をインストールしたマシンが常時起動しているので、そのマシンに pptp サーバになってもらいます。

Linux 用に開発された poptop という pptp の実装が FreeBSD に移植されているのでこれを使います。ports の net/poptop にあります。poptop は ppp を呼び出すので tun が使えるようにしておく必要があります。

ports から導入した場合 poptop の設定ファイルは/usr/local/etc/pptpd.conf ですが、FreeBSD の user-ppp と poptop はこの設定に関して協調して動作しないのでここには何も記述する必要はありません。/etc/ppp/ppp.conf の pptp プロファイルに直接記述します。

/etc/ppp/ppp.conf

```
pptp:
 set timeout 0
 set log Phase Chat connect LCP IPCP command
 enable proxy
 accept dns
 set dns 192.168.1.1
 set rdns 192.168.1.1
 enable MSChapV2
 enable mppe
 set mppe * stateless
 set ifaddr 192.168.1.1 192.168.1.110-192.168.1.120 255.255.255
```

場合によっては

```
disable deflate pred1
deny deflate pred1
```

を記述する必要があるかもしれません。

192.168.1.1 は pptp サーバ (FreeBSD マシン) のローカル IP アドレスです。set ifaddr のところで接続してきたクライアントに 192.168.1.110 から 192.168.1.120 までの何れかのアドレスを割り振るよう ppp にお願いしています。この IP アドレスがサーバ側、クライアント側の LAN にある他のマシンとかぶらないように設定します。

また、/etc/ppp/ppp.secret に接続するユーザ名とパスワードを

```
user password
```

のように記述しておきます。

pptpd を起動するには/etc/rc.conf に

```
pptpd_enable="YES"
```

を追加して、/usr/local/etc/rc.c/pptpd.sh start を実行します。

これでサーバ側の設定は終わりです。後はクライアントから接続するだけです。

Windows(Windows XP の場合) からは、ネットワーク接続-¿新しい接続を作成する、で、新しいネットワークウィザードが起動するので、そこから職場のネットワークに接続する-¿仮想プライベートネットワークを選ぶと VPN クライアントの設定ウィザードに入ります。

FreeBSD からは pptpclient を使って接続します。/etc/ppp/ppp.conf に以下のようなプロファイルを追加します。

/etc/ppp/ppp.conf

```
pptpclient:
 set authname USER
 set authkey PASSWORD
 set timeout 0
 set ifaddr 0 0
 add 192.168.1.0/24 HISADDR
 alias enable yes
```

あとは、

`/usr/local/sbin/pptp サーバの IP アドレス (WAN 側) pptpclient`

の用に pptpclient を起動します。

これで VPN 接続が張れているはずです。試しに相手方のネットワークにある pptpd が動いているマシンとは別のマシンに ping を打ってみます。

```
> ping 192.168.1.2 1000 5
PING 192.168.1.2 (192.168.1.2): 1000 data bytes
1008 bytes from 192.168.1.2: icmp_seq=0 ttl=127 time=45 ms
1008 bytes from 192.168.1.2: icmp_seq=1 ttl=127 time=16 ms
1008 bytes from 192.168.1.2: icmp_seq=2 ttl=127 time=25 ms
1008 bytes from 192.168.1.2: icmp_seq=3 ttl=127 time=15 ms
1008 bytes from 192.168.1.2: icmp_seq=4 ttl=127 time=22 ms
```

つながっているみたいです。Windows の共有フォルダも見えています。これで LAN 内と同じようにファイル共有が出来るようになりました。とりあえずはここでおいておきます。

6.4 おわりに

本当は何かを作りましたよという記事を書きたかったのですが、何も出来なかったのでこのような記事を書いてしまいました。この記事を書くために関連するインストール作業や設定をそこそこ時間を使ってやっていて、そこから書いていこうと思ったのですが、Lime24 の松村さんの記事にほぼ全て書かれていたのでがっかりしているところです。Lime のバックナンバーは KITCC のウェブサイトからダウンロードできます。そちらも是非、ご覧下さい。

参考文献

[1] FreeBSD 日本語マニュアル検索
http://www.jp.freebsd.org/man-jp/search.html

[2] FreeBSD ハンドブック日本語版
http://www.freebsd.org/doc/ja_JP.eucJP/books/handbook/index.html

7 いつか、もも缶は眠れぬ夜を抱いて

電子情報工学科 2回生 林 奉行

7.1 Hello, World!

トルルルルルルルル、トルルルルル。そろそろ日付も変わろうかというころ、あたりに電話の呼び出し音が鳴り響く。狭い部屋の中にはその半分もある大きなベッドがひとつ。それ以外にはこれと言って調度品もない。電話も床に置きっぱなしである。そのベッドの上には、がたいのよい男が一人寝苦しそうに布団に頭を突っ込んでいた。しばらくして、観念したのか男が受話器を取る。ガチャリ。

??? ああ、俺だ。そうだ、bokko だ。え？、nish じゃなかったのかって？その名前も時々使うな。なーに偽名は星の数ほどあるさ。しいて言えば、よくこう呼ばれているな。「マスター」ってな。

そう、彼こそが、年齢不詳、性別どう見ても漢、その他一切不詳の宇宙をまたにかけて暴れまわる「マスター」その人であった。

7.2 マスター現る

マスター こんな時間に呼び出しやがって。少しは時差ってものを考えろ。

電話の相手 ＜あ、いやすみません。どうしてもあなたの手を借りたいことがありまして＞

マスター 俺の手を借りたいだって？いったいどうしたってんだ？bugyo。

この電話の相手の名前は bugyo。以前はマスターの金魚の糞として宇宙を渡り歩いたが、腰を痛めたためマスターについていくことを断念して、たいした実力もないのにある大企業に就職した。当然裏はあるが今語るべきことではない。人を頼ることに余念がない男である。

マスター 俺に頼むってことは、それなりのもんを用意してるってこったろーな。

bugyo ＜それはもう。特上の桃を用意しました。真っ白でぷりぷりの白桃を＞

マスター ほう。ぷりぷりのやつか。そりゃーいい。早速だが詳しく教えてくれ。

bugyo ＜それが、勤め先のサーバがクラッシュしまして。早急にデータのサルベージと新システムの構築が必要になったのです。それはもう、今夜中に＞

マスター おまえさん、新入社員だろ。それに管理者はどこへいったんだ。

bugyo ＜それがどうも、自分の才能に限界を感じたらしく、自給自足生活をするんだと言って、つい二日前に田舎に帰ってしまったばかりで、代わりの人を探していたところだったんです。他にわかる人もいなくて、でもとりあえず復旧させないと業務に支障が出るから皆で押し付けあってるうちに＞

マスター おまえさんに白羽の矢がたったってことか。前の管理者を最後の仕事だって呼び出してみたらどうだ。

bugyo ＜それがその人の田舎がどうもＡＱＵＡらしくて。今から追いつこうと思ったら通信でも３週間はかかりますよ。新しく人を雇って来るまで何とか動くようにしないと僕の窓際族としての地位が失われてしまうんです。＞

マスター 他の部署から人を借りてくればいいだろう。

bugyo それが全く取り合ってくれないんですよ。なんか阻害されている感じです。

マスター それは大変だな。それで、どこまで進んでるんだ？

bugyo それがまったく。

マスター …… 。

bugyo …… 。

マスター しかたねぇ。今からそっちに行くからおとなしく待ってろよ。

こうして、マスターは bugyo を助けてやるために愛車、狐妖手号を駆って株式会社 栗実生ルへと急ぐのであった。

7.3 おとなしくその鯖を渡すんだ

ここは株式会社 栗実生ル本社ビル最上階。広い部屋、高い天井。薄闇の部屋の中央で豪華な飾り椅子にいかにもマダムといった洋装で腰掛ける婦人が一人。じゅるり。この人こそ株式会社 栗実生ルの社長兼大株主のマダム○チアーノである。もぐもぐ。

マダム 卯月。このバッテラとても美味しいわ。

卯月 地中海は大分市で水揚げされた関鯖ですもの。それをわざわざバッテラにしていただくなんて、お母様の食へのこだわりがいかほどのものか思い知りましたわ。私たちも地中海まで捕りに行った甲斐があります。

マダム もぐもく。それで、例の計画は進んでいるのかしら。

卯月 ええ、すこぶる順調ですわ。たった今 bugyo が何者かに連絡しているところを確認したと長月から報告があったところです。おそらく相手はマスターでしょう。

マダム ふふ。昔の仲間が窮地に立つときマスターは必ず助けに来る。作業が終わってくたくたになったマスターへ子供たちが強襲をかける。完璧な作戦ね。このバッテラのように惚れ惚れするわ。この会社を買ったのも bugyo を就職させたのも管理者をノイローゼにしたのもサーバをＥＭでクラッシュさせたのもこのため。必ずやマスターを捕らえて１兆宇宙ドルの在り処を聞き出してやるわ。

卯月 お母様も、なかなかのワルですわね。

マダム さあ、そろそろマスターが来るわ。準備はよろしくて？

卯月 まかせてくださいな。お母様。ふふっ、**ぶっ壊して差し上げますわ。**

7.4 困ったときは、ちょいのせ大

マダム○チアーノの罠とも知らず bugyo を助けるために株式会社 栗実生ルにやってきたマスター。bugyo のところに行ってみると、bugyo は精一杯の逃避エネルギーで遊んでいた。

マスター おい。相変わらずだな。せっかく俺が来てやったのに、何を楽しそうに遊んでやがるんだ。

bugyo マスターが遅いからこの陰鬱な気を紛らわせていたんですよ。

マスター まあ、いいが。よし、はじめるぞ。

bugyo えー。もう少しで「これ、食べてからでいいよね」ってシーンなんですよ。ここだけ見ていきましょうよ。

マスター いいから、来い。

bugyo うわーん。ずるずるずる。

マスターに引きずられて、bugyo はサーバの前までやってきた。

マスター クラッシュしたって言うマシンはこれか。

bugyo そうなんですよ。何時の間にか動かなくなっていたんです。

マスター こりゃ明らかに人為的に壊されてるな。まぁいいが。データの移行は終わってるんだったな。

bugyo はい。そのぐらいは何とかできました。

マスター どれ、見せてみな。…… 、これどうやって移行したんだ？

bugyo 普通に cp コマンドですけど。

マスター パーミッションやタイムスタンプを保持したままコピーするには cp コマンドに p オプションをつけるんだ。そして普通は直接 cp するようなコマンドの使い方はしない。…… 。

＊＊＊＊＊＊なんか技能的な話＊＊＊＊＊＊

マスター ふー、何とか終わったな。

bugyo 首の皮一枚でつながりました。ありがとうございます。これ、お約束の桃です。

マスター おいおい、こんなところにつれてきちまったのか。って、なんだこの缶詰。

bugyo いやあ、僕の実家はここからちょっと遠いので缶詰にして送ってくれたんですよ。どうです。このぷいぷりさくさく。絶品ですよ。はい、あーん。

マスター ほんとに桃だったとは … 。ふざけるなこのやろー。

ガラガラッ、、どっかーん !!

睦月 ハァーイ、マスター。

マスター お、お人形ちゃんたちじゃねーか。すると、ここは。

卯月 そうよ、これはお母様の仕掛けた罠。クスクスっ、せっかくこれからお楽しみだったって言うのに、**お気の毒に ••• 。**

神無月、霜月、極月 あー。ももか〜ん。

卯月 やっておしまい。

ダダダダッ、ドカンドカン、バシュバシュッ、ズサズサ、うっ。

マスター おい、bugyo。ここはやべぇ。ずらかるぞ。

bugyo え、僕もなんですか。

マスター あたりまえだ !!

bugyo わーん。せっかく就職したのにー。

この後もマスターは宇宙をまたにかけておお暴れしたが、その背後に bugyo が隠れていたかどうかは定かではない …… 。

THE END.

8 Wiki パーサを作ろう

電子情報工学科 4回生 久保 達彦

8.1 はじめに

去年の夏, O'Reilly Media の創立者である Tim O'Reilly 氏が「What Is Web 2.0」を発表してからネット上の記事のみならず, 新聞でも「Web2.0」というキーワードをよく見かけるようになりましたが, この論文で定義されている「Web2.0」の意味自体が随分と曖昧なせいか, いろんな解釈の仕方がされて言葉が一人歩きしてしまい, なんでもかんでも 2.0 と叫ばれるようになったり, [1] Web2.0 バブルだとか言われるようになったりして, 有名な IT 系ベンチャーが頼んでもいない VC[2]から融資を申し出られたり, Web2.0 はもう終焉に向かっているとかいう記事が出てきたりと昔どこかで見たような光景が広がっている今日この頃ですが, Web2.0 というのはものすごく簡単に言うと, Ajax などの技術の台頭により Web アプリケーションでもデスクトップアプリケーションと遜色のないことができるようになったとか, ブログや SNS や Wiki のようにネット上でより高度な情報発信やコミュニケーションや情報共有ができるようになったとか, つまり昔の Web(Web1.0) に比べて Web がパワーアップしたということです。[3]今回の記事では, そんな Web 上で動く情報共有アプリケーションの 1 つである「Wiki」に注目してみます。

8.1.1 Wiki とは？

Wiki とは Web ブラウザ上から文書の作成, 編集, 管理, 共有ができるシステムのことです。1995 年にワード・カニンガム氏が最初のウィキサイトである『Portland Pattern Repository』を創設して以来, いろいろな Wiki クローン[4]が作られています。

8.1.2 Wiki パーサ

Wiki では文章を HTML でゴリゴリ記述するのではなく, 独自の記法を使って記述することができます。こうすることによって HTML で記述するよりも簡単に文章を記述することができます。この記法を解析して入力したテキストを HTML テキストに変換するのが Wiki パーサの仕事です。今回の記事では, この Wiki パーサの作り方について解説します。本当は Wiki エンジン全体の作り方について解説したかったのですが, 時間がなかったのでパーサの作り方に的を絞って話をさせていただきます。

---

[1] マーケティング 2.0, Security2.0, etc
[2] ベンチャーキャピタル
[3] Web2.0 という言葉に明確な定義はありません。本記事のテーマから外れますので詳しくは述べませんが, 興味のある方は最後に参考になる URL を載せておきますのでそれを見たり, 検索して調べたりしてみてください。
[4] FreeStyleWiki, PukiWiki, Hiki, etc

8.2 使用言語

パーサを実装する言語には PHP を使用しています。本当は Ruby でも良かったんですが, 諸事情により PHP を選択しました。

8.3 動作確認した環境

- Windows XP SP2
- PHP5.0.2
- Apache2.0.52

もしくは,

- Vine Linux3.2
- PHP5.1.4
- Apache2.0.58

PHP や Apache はセキュリティ向上のため, できるだけ最新のものをお使い下さい。(もちろん OS も)

8.4 正規表現

実装の話に入る前に少し正規表現の話をしておきます。文字列の集合を一つの形式で表現する方法のことで, これを使うとテキストの中から指定した文字列やそのパターンを抜き出したり, 置換することが容易になります。パーサの行う処理は結局のところ, 文字列処理ですから正規表現を積極的に使うことで, シンプルで短く見やすいコードを書くことが出来ます。後述する記法の実装には正規表現をたくさん使っています。これを見れば正規表現がいかに強力かわかると思います。

8.5 記法

Wiki では, 文章を HTML タグで直接書くのではなく, 定められた記法で文章を書くことが出来ます。[5]Wiki の記法を使うと直接 HTML で記述するよりも簡潔に HTML を使った文章を書けます。例えば, 私が現在開発している Wiki では,

```
!!!Wikitcc の記法一覧

*見出し記法
*引用記法
*リンク記法
*リスト記法
*テーブル記法
*mailto 記法

詳しくは, mailto:bokko@kitcc.org まで。
または, [こちら|http://www.kitcc.org] まで。
```

[5]HTML タグが全く使えないというわけではありません。

を,

```
<h2 id="headline">Wikitcc の記法一覧</h2>
<ul>
<li>見出し記法<li>引用記法</li>
<li>リンク記法</li>
<li>リスト記法</li>
<li>テーブル記法</li>
<li>mailto 記法</li>
</ul>

詳しくは, <a href="mailto:bokko@kitcc.org">bokko@kitcc.org</a>まで。
または, <a href="http://www.kitcc.org">こちら</a>まで。
```

のように変換します。

8.5.1 テスト用ページの準備

動作に最低限必要なファイルは, test.html(入力), parser.php(記法解析, 出力), Wikitcc.class.php(記法解析) の 3 つです。また, これらのファイルの文字コードは統一してください。

test.html

```
1
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
5
6  <title>Wiki テスト</title>
7  </head>
8  <body>
9
10 <center>
11
12 <h1>Wiki テスト</h1>
13 <form action="parser.php" method="post">
14 <textarea name="text" cols="50" rows="10"></textarea>
15 <br>
16 <input type="submit" name="edit" value="編集" />
17 </form>
18 </center>
19
20 </body>
21 </html>
```

test.html から文章を入力して編集ボタンを押すと, 解析結果が出力されます。

parser.php

```
1
2  <html>
3  <head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="stylesheet" type="text/css" href="wiki.css">
<title>出力</title>
</head>
<body>

<?php
require_once "Wikitcc.class.php";

$wiki_text = stripslashes($_POST['text']);
$wiki = new Wikitcc($wiki_text);
$wiki->parse();                              //入力した文章を解析
print($wiki->text);                           //解析結果を表示
?>

</body>
</html>
```

解析結果を出力するページです。文章を解析する関数 (parse) を呼び出して, その結果を出力しています。

Wikitcc.class.php

```

<?php

class Wikitcc{

    var $text;

    public function Wikitcc($text){
     $this->text = $text;
    }

    public function parse(){
        $this->unifyKaigyo();
        $this->addBrTag();
        $this->rmEmpLine();
        $this->createHeadline();
        $this->createQuotation();
        $this->createLink();
        $this->createList('*');
        $this->createList('+');
        $this->createTable();
        $this->createMailTo();
        $this->createSuperPre();
        $this->createPre();
    }

    //見出し記法
    private function createHeadline(){
         ・
         ・
         ・
    }

    //引用記法
    private function createQuotation(){
```

```
36          ・
37          ・
38          ・
39      }
40
41      //リンク記法
42      private function createLink(){
43          ・
44          ・
45          ・
46      }
47
48      //リスト記法
49      private function createList(){
50          ・
51          ・
52          ・
53      }
54
55      //テーブル記法
56      private function createTable(){
57          ・
58          ・
59          ・
60      }
61
62      //mailto 記法
63      private function createMailto(){
64          ・
65          ・
66          ・
67      }
68
69      //pre 記法
70      private function createPre(){
71          ・
72          ・
73          ・
74      }
75
76      //スーパー pre 記法
77      private function createSuperPre(){
78          ・
79          ・
80          ・
81      }
82
83      //改行コードを統一
84      private function unifyKaigyo(){
85          ・
86          ・
87          ・
88      }
89
90      //改行 (\n) の前に br タグを挿入
91      private function addBrTag(){
92          ・
93          ・
```

```
94          .
95      }
96
97      //改行記法
98      private function rmEmpLine(){
99          .
100         .
101         .
102     }
103 }
104 ?>
```

文章を解析するための関数を集めたファイルです。このファイルは大きいので各関数ごとに後述します。

8.5.2 記法の実装

次は, それぞれの記法を解析するためのコードの解説に入ります。「改行コードの統一」, 「改行の前に br タグを挿入」には入力された文字列をプログラムが処理しやすいようにするためだったり, HTML として表示するための体裁を整えるという目的があります。

改行コードを統一

```
1
2 private function unifyKaigyo(){
3     $this->text = preg_replace("|\r\n?|", "\n", $this->text);
4 }
```

改行コードを統一しています。こうすることによって環境が違っても入力した文章を同じ文字列として扱えるようにしています。

改行の前に br タグを挿入

```
1
2 private function addBrTag(){
3     $this->text = nl2br($this->text);
4 }
```

改行コードの前に
 という文字列を挿入します。入力された文章ではちゃんと改行されていてもHTML で表示する場合には改行されません。HTML で改行するには
 と書く必要があります。

見出し記法

入力[6]

```
!!!大見出し
!!中見出し
!小見出し
```

出力

```
<h2 id="headline">大見出し</h2>
<h3 id="headline">中見出し</h3>
<h4 id="headline">小見出し</h4>
```

ソースコード

```
1
2  private function createHeadline(){
3
4      $this->text = preg_replace("/\n!!!(.*)<br \/>/",
5                                 "\n<h2 id=\"headline\">\\1</h2>", $this->text);
6
7      $this->text = preg_replace("/\n!!(.*)<br \/>/",
8                                 "\n<h3 id=\"headline\">\\1</h3>", $this->text);
9
10     $this->text = preg_replace("/\n!(.*)<br \/>/",
11                                "\n<h4 id=\"headline\">\\1</h4>", $this->text);
12 }
```

!で始まる行は「見出し」と見なします。このコードでは見出しの大きさを!の数で区別しており, 全部で 3 段階の大きさを指定できます。
1 というのは (.*) の部分で!と
 との間にある文字列を指しています。上記の例では指定した部分 (.*) を取り出してタグの中に入るように文字列を加工しています。

引用記法

入力

```
>>
ここは引用です。
<<
```

出力

[6] この記事を参考にして Wiki を動作させようとしている人は文章を入力する際に最後に空行を入力してください。正しく動作しない可能性があります。

```
<blockquote>
ここは引用です。
</blockquote>
```

ソースコード

```
1
2 private function createQuotation(){
3     $this->text = preg_replace("|\n>><br />|", " <blockquote> ", $this->text);
4     $this->text = preg_replace("|\n<<<br />|", " </blockquote> ", $this->text);
5 }
```

>> と << で囲まれた部分を blockquote で囲みます。blockquote タグは引用を明示するためのタグです。コードでは >> と << を <blockquote>, </blockquote> に変換しているだけです。

リンク記法

入力

```
リンク記法その 1 [KITCC|http://www.kitcc.org]
リンク記法その 2 http://www.kitcc.org
リンク記法その 3 link:KITCC;url:http://www.kitcc.org
```

出力

```
リンク記法その 1 <a href="http://www.kitcc.org">KITCC</a>
リンク記法その 2 <a href="http://www.kitcc.org">http://www.kitcc.org</a>
リンク記法その 3 <a href="http://www.kitcc.org">KITCC</a>
```

ソースコード

```
1
2  private function createLink(){
3
4      $reg = "(https?|ftp):\/\/[-_.!~*\'a-zA-Z0-9;\/?:\@&=+\$,%#]+";
5
6      //例:http://www.kitcc.org
7      $this->text = preg_replace("/([^\|:])($reg)/",
8                                 "\\1<a href=\"\\2\">\\2</a>", $this->text);
9
10     //例:link:KITCC;url:http://www.kitcc.org
11     $this->text = preg_replace("/link:([^;]+);url:($reg)/",
12                                "<a href=\"\\2\">\\1</a>", $this->text);
13
14     //例:[KITCC|http://www.kitcc.org]
15     $this->text = preg_replace("/\[([^\[]+?)\|($reg)\]/",
```

```
16                              "<a href=\"\\2\">\\1</a>", $this->text);
17 }
```

リンク先の名前と URL を指定します。正規表現が少しごちゃごちゃしていますが, それぞれの記法を解析する際に衝突しないようにしているためです。

リスト記法

入力

```
*1
**1-1
**1-2
***1-2-1
***1-2-2
**1-3
```

出力

```
<ul>
<li>1<ul>
<li>1-1</li>
<li>1-2<ul>
<li>1-2-1</li>
<li>1-2-2</li></ul>
<li>1-3</li>
</li>
</ul></li>
</ul>
```

ソースコード

```
1
2  private function createList($sl){
3      $text = "";
4      $lines = explode("\n", $this->text);
5      $in_list_flag = false; //リストの中か?
6
7      if(strcmp($sl, '*') == 0){
8          $lt = "<ul>";                                    //リストタグ (始端)
9          $rt = "</ul>";                                   //リストタグ (終端)
10     }
11     else if(strcmp($sl, '+') == 0){
12         $lt = "<ol>";                                    //リストタグ (始端)
13         $rt = "</ol>";                                   //リストタグ (終端)
14     }
15
16     $ii = 0;
17     foreach($lines as $line){
18         if(!$in_list_flag){                              //リストの外
```

```
19              if(preg_match("/^\\$sl/", $line)){          //リストの開始
20                  $in_list_flag = true;
21                  $line = preg_replace("|<br />|", "", $line);
22                  $a_num = strspn($line, $sl);
23                  $line = substr($line, strspn($line, $sl), strlen($line)-1); //*,+の除去
24                  for($i=0;$i<$a_num;$i++){$text .= $lt."\n";}
25                  $text .= "<li>".$line;
26              }
27              else{$text .= $line."\n";}
28          }
29          else{                                               //リストの中
30              $a_num_before = strspn($lines[$ii-1], $sl);     //前の行の+,*の数
31              $a_num_current = strspn($line, $sl);            //現在処理している行の*,+の数
32              $a_num_next = strspn($lines[$ii+1], $sl);       //次の行の+,*の数
33
34              if(!preg_match("/^\\$sl/", $line)){ //リストの終了
35                  $in_list_flag = false;
36                  $text .= "</li>\n";
37                  for($i=0;$i<$a_num_before-1;$i++){
38                      $text .= "$rt</li>\n";
39                  }
40                  $text .= $rt."\n";
41                  $text .= $line."\n";
42              }
43              else{
44                  $line = preg_replace("|<br />|", "", $line);
45                  $line = substr($line, $a_num_current, strlen($line)-$a_num_current);  //*,+の除去
46                  if($a_num_before < $a_num_current){
47                      $n = $a_num_current - $a_num_before;
48                      for($i=0;$i<$n;$i++){
49                          $text .= "$lt\n";
50                      }
51                      if($a_num_current < $a_num_next && $a_num_next != 0){$text .= "<li>".$line;}
52                      else{$text .= "<li>".$line."</li>\n";}
53                  }
54                  else if($a_num_before > $a_num_current){
55                      $n = $a_num_before - $a_num_current;
56                      for($i=0;$i<$n;$i++){
57                          $text .= "</li>$rt";
58                      }
59                      if($a_num_current < $a_num_next && $a_num_next != 0){$text .= "<li>".$line."\n";}
60                      else{$text .= "<li>".$line."</li>\n";}
61                  }
62                  else{
63                      if($a_num_current == $a_num_next && $a_num_next != 0){$text .= "<li>".$line."</li>\n";}
64                      else{$text .= "<li>".$line;}
65                  }
66              }
67          }
68          $ii++;
69      }
70      $this->text = $text;
71  }
```

*, +で始まる行はリストと見なします。このコードでは関数の引数が*の場合はただのリスト, +の場合は番号付きのリストになるようにしています。また, *,+の数が増える度にリストの階層が深くなります。

テーブル記法

入力

```
,1-1,1-2,1-3
,2-1,2-2,2-3
,3-1,3-2,3-3
```

出力

```
<table border="1">
<tr><th>1-1</th><th>1-2</th><th>1-3</th></tr>
<tr><td>2-1</td><td>2-2</td><td>2-3</td></tr>
<tr><td>3-1</td><td>3-2</td><td>3-3</td></tr>
</table>
```

ソースコード

```
1
2  private function createTable(){
3
4      $in_table_flag = false;
5      $is_header = true;
6      $text = "";
7      $lines = explode("\n", $this->text);
8
9      foreach($lines as $line){
10         if(preg_match("/^,/", $line)){
11             $in_table_flag = true;
12             if($is_header){
13                 $text .= "<table border=\"1\">\n<tr>";
14                 $tag = "th";
15             }
16             else{
17                 $tag = "td";
18             }
19             $is_header = false;
20             $elements = explode(",", $line);
21             for($i=1;$i<count($elements);$i++){
22                 $text .= "<$tag>".$elements[$i]."</$tag>";
23             }
24             $text .= "</tr>\n";
25         }
26         else{
27             if($in_table_flag){
28                 $text .= "</table>";
29             }
30             $in_table_flag = false;
31             $is_header     = true;
32             $text .= $line."\n";
33         }
34     }
35     $this->text = $text;
```

```
}
```

, で始まる行の集まりはテーブルと見なします。このコードではテーブルの最初の 1 行を項目名と見なしてほかの行のように td タグではなく th タグを使っています。

mailto 記法

入力

```
mailto:bokko@kitcc.org
```

出力

```
<a href="mailto:bokko@kitcc.org">bokko@kitcc.org</a>
```

ソースコード

```
1
2 private function createMailTo(){
3     $reg = "/mailto:([a-zA-Z0-9_\.\-]+?@[A-Za-z0-9_\.\-]+)/";
4     $this->text = preg_replace($reg, "<a href=\"mailto:\\1\">\\1</a>", $this->text);
5 }
```

mailto:メールアドレスと記述すると, 指定したメールアドレスをリンク先として指定することができます。これも正規表現で指定した部分を取り出してちょっと加工してやるだけでできます。

pre 記法

入力

```
>|
ここはテキスト整形ブロックです。
スペースや改行がそのまま適用されます。
|<
```

出力

```
<pre>
ここはテキスト整形ブロックです。
スペースや改行がそのまま適用されます。
</pre>
```

ソースコード

```
private function createPre(){
    $text = "";
    $lines = explode("\n", $this->text);
    $in = false;

    foreach($lines as $line){
        if($in){
            $text .= preg_replace("/<br \/>/", "", $line)."\n";
        }
        else{
            $text .= $line."\n";
        }
        if(preg_match("/^>\|<br \/>/", $line)){
            $in = true;
            $text = preg_replace("/>\|<br \/>/", "<pre>", $text);
        }
        elseif(preg_match("/^\|</", $line)){
            $in = false;
            $text = preg_replace("/\|</", "</pre>", $text);
        }
    }
    $this->text = $text;
}
```

>|と|<で囲まれた箇所はテキストが整形される pre タグブロックになります。このブロックの中では br タグをはさんだりしなくても改行するだけで, ブラウザで表示するときには改行してくれます。また, スペースもそのまま適用されます。

スーパー pre 記法

入力

```
>||
int main(void){
    printf(" & < > ' ");
}
||<
```

出力

```
<pre>
int main(void){
    printf(&amp;quot; &amp;amp; &amp;lt; &amp;gt; &amp;#039; &amp;quot;);
}
</pre>
```

ソースコード

```
private function createSuperPre(){
```

```
3     $text = "";
4     $lines = explode("\n", $this->text);
5     $in = false;
6
7     foreach($lines as $line){
8         if($in){
9             $line = preg_replace("/<br \/>/", "", $line)."\n";
10            if(!preg_match("/^\|\|</", $line)){
11                $text .= htmlspecialchars($line, ENT_QUOTES);
12            }
13            else{
14                $text .= $line;
15            }
16        }
17        else{
18            $text .= $line."\n";
19        }
20        if(preg_match("/^>\|\|/", $line)){
21            $in = true;
22            $text = preg_replace("/>\|\|<br \/>/", "<pre>", $text);
23        }
24        elseif(preg_match("/^\|\|</", $line)){
25            $in = false;
26            $text = preg_replace("/\|\|</", "</pre>", $text);
27        }
28    }
29    $this->text = $text;
30 }
```

> || と || < で囲まれた箇所は pre 記法と同じく pre タグで囲まれます。この記法はそれに加えて, HTML タグを無効にしたり, HTML エンティティに変換が必要な文字を HTML エンティティに変換します。ソースコードを貼り付けたりするのに便利です。

改行記法

入力

```
*1


*2
```

出力

```
<ul>
<li>1</li>
</ul>
<br />
<ul>
<li>2</li>
</ul>
```

ソースコード

```
private function rmEmpLine(){
    $this->text = preg_replace("/<br \/>\n<br \/>\n/s", "<br />\n", $this->text);
}
```

空行を 2 回続けると br タグを 1 つはさみます。これは余計な br タグが挿入されるのを防ぐのと, 入力時の文章を見やすくするために使います。

8.6 CSS

さて, これで入力した文章を HTML テキストに変換することができますが, これだけでは見栄えが悪いです。引用やテーブルの背景は真っ白だし, 見出しは真っ白な上に 3 段階のレベルに分けられてるのにもかかわらず, 違うのは大きさだけです。せっかくだから出力ページをもう少し彩り豊かなものにしてみましょう。以下のファイルを見てください。

wiki.css

```
/* 見出し */
h2#headline {
background-color: rgb(208, 228, 252);
}

h3#headline {
border-left-width : 10px;
border-left-style : solid;
border-left-color : rgb(208, 228, 252);
border-top-width  : 5px;
border-top-style  : solid;
border-top-color  : rgb(208, 228, 252);
}

h4#headline {
border-left-width : 10px;
border-left-style : solid;
border-left-color : rgb(208, 228, 252);
padding-left : 2pt;
margin-bottom : 5pt;
}

/* 引用ブロック */
blockquote {
background-color: rgb(208, 228, 252);
}

/* テーブルヘッダ */
th {
background-color: rgb(208, 228, 252);
}
```

それぞれのタグや id で指定した部分に対してのスタイルを記述しています。見出し記法で id="headline"という記述があったのを思い出してください。h2#headline というのは $< h2id = "headline" > \sim < /h2 >$ のことを指しています。background-color: rgb(208, 228, 252); というのは背景の色を指定した RGB のものに変更します。つまり, h2 タグの中で id が headline のものに対して背景の色をここで指定した RGB のものに変更するということです。このファイルは前に紹介した 3 つのファイルと同じディレクトリに置いてください。

8.7 最後に

Wiki というシステムはパーサの更新履歴の表示や文書の作成, 表示, 編集等々, ほかにもいろいろな機能から構成されていますが, それほど難しいものでは無いと思います。この記事を読んでおもしろいと思った人は自分だけの Wiki を作ってみてはいかがでしょうか？

参考文献

[1] What Is Web 2.0
http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html
↑の日本語訳その 1 http://japan.cnet.com/column/web20/story/0,2000055933,20090039,00.htm
↑の日本語訳その 2 http://japan.cnet.com/column/web20/story/0,2000055933,20090424,00.htm

[2] Wikipedia - Wiki
http://ja.wikipedia.org/wiki/Wiki

[3] はてなダイアリーのヘルプ - はてな記法一覧

http://hatenadiary.g.hatena.ne.jp/keyword/\%e3\%81\%af\%e3\%81\%a6\%e3\%81\%aa\%e8\%a8\%98\%e6\%b3\%95\%e4\%b8\%80\%e8\%a6\%a7

[4] FreeStyleWiki

http://ja.wikipedia.org/wiki/\%E6\%AD\%A3\%E8\%A6\%8F\%E8\%A1\%A8\%E7\%8F\%BE

[5] 正規表現とは - はてなダイアリー

http://d.hatena.ne.jp/keyword/\%C0\%B5\%B5\%AC\%C9\%BD\%B8\%BD

編集後記

Lime32 号をお届け致しました。最後まで目を通して頂き、ありがとうございます。本年度より Lime の編集を担当させていただいております、林と申します。

計画性を持って、と 2 年前の Lime の編集後記にて当時編集担当の若松さんは書いておられるのですが、今回は私事にかまけてそれを大幅に無視することになってしまいました。また、手伝ってくれた小長谷君には自分の頼りなさ故に大いに迷惑を掛けました。

このような状況であったわけでございますが、何とか発行にこぎ着けることが出来ました。記事の内容についてですが、HD-DVD や Blu-ray、Web2.0 といった新しい話題から、自作 Scheme インタプリタのような濃いものまで多岐にわたっており、楽しんで頂ける内容となったのではないかと思っているところであります。

来年もこの仕事を任せてもらえるかは、まだ結果の出ていない事でありますが、どうであれ内容、体裁共に充実した物になるよう努力したいと思いますので、来年の Lime にも、どうぞご期待下さいますよう。

平成 18 年 11 月 19 日 編集担当 林 奉行

Lime Vol.34

平成 18 年 11 月 24 日 発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部
http://www.kitcc.org/