Code Golf

浜地慎一郎

要求仕様を満たす最短のコードを考え、競う遊びを Code Golf と呼ぶ。その Code Golf 全般について、基本的なルール、ゴルフを行う文化、ゴルフに特有のイディオムや際立ったアルゴリズムなどについて、概略的な説明を行う。

Code Golf

Shinichiro Hamaji

1. Code Golf とは

ゴルフとは可能な限り短いストローク数 (打数) でカップにボールを入れることを競うスポーツだが、それになぞらえて、可能な限り少ないストローク数 (打鍵数、バイト数) で要求仕様を満たすプログラムを作成する遊びを Code Golf と呼ぶ。ここでは Code Golf に関する様々な知見をまとめてみたい。

まずはゴルフはどのように行われるかを具体的な例で示す。要求仕様を満たすプログラム、と書いたが、曖昧な要求 (例えばオセロを作れ) であれば、具体的な仕様を単純に考えた者が有利になってしまう (例えばオセロに AI をつければつけない者より不利である) ため、競争の題材は簡単な物が選ばれることが多く、たいていは標準入力やコマンドラインから入力を受け取り、標準出力に定められた結果が出ているかを確認する、というようなものである。

具体的な例として、 “delete blank line” 問題[⋆1] を見てみよう。この問題は標準入力から文章を受け取って、空行だけの行を削除するというものである。1 行が 98 文字以下であると仮定して C で書くと、

```
#include <stdio.h>
int main() {
    char line[99];
    while (gets(line)) {
        if (line[0]) puts(line);
    }
}
```

⋆1 http://golf.shinh.org/p.rb?delete+blank+lines

などとなるであろう。これを私が適当に縮めてみたところ、

```
main(i){for(;gets(&i);)i&63&&puts();}
```

というコードになった。このコードのポイントを箇条書きにしてみると以下のようなものがある。

- `#include` が無いが、問題ない。 C では宣言してない関数は `int f()` と解釈される。
- `int` 型の宣言が省略されている。
- `main` の引数が一つしか無い。
- `gets` に渡しているのが文字列ではなく、 `int*` である。スタック領域を `gets` の結果置き場に利用している。
- `i&63` は少しややこしい。まずリトルエンディアンで実行されることを仮定すると、 `i&255` で先頭の 1 文字が読めるはずである。ここでは、入力文字列が ASCII であり、行頭に `@`(64) が来ないと仮定することによって `63` としている。
- `puts` の引数を渡していない。こうすることによって x86 の環境ではスタックに残っている引数が使われ、つまり `gets` の引数がそのまま残っているので問題ない。

このように、プログラムを書く時の常識的な作法 (型をちゃんと書くなど)、可搬性 (エンディアンなど)、仕様に対する完全性 (行頭 `@` への対応)、などを完全に無視して、とにかく問題で与えられた入力に対して可能な限り短いコードを書くことだけを競うことが、ゴルフの流儀になっている。

ただし、 C 言語で Perl を呼び出したり、保存しておいたファイルを使用するなど、明らかにパズルとしての面白みを損なう行為に関しては、ルールで一定の

制限がされていることが多い。

2. 歴史やコミュニティ

コードを短くする遊びをゴルフと呼ぶようになったのは恐らく Perl コミュニティが発祥であり、ニュースグループで行っていたころから現在の Web ベース [*1] でのコンペティションまでの歴史を編纂した Perlgolf History Book [*2] は 520 ページという大作になっている。その Perl Golf を Perl 以外の、Ruby、PHP、Python でも遊べるようにしたのが Code Golf [*3] というサイトであり、これに多数の日本人が参加したことから、日本でもゴルフという単語が多少メジャーになっていった。私の運営するサーバ[*4] では対応言語を大幅に増やし、問題もユーザが投稿できるようになっている。

もちろん、このように競技・競争としてコードを短くすることが流行する前から、短いコードを探索することは、Quine などと同様、プログラムを用いたハッカーの遊びとして存在していたと考えられる。例えば、スティーブン・レビーの「ハッカーズ」[*5] には、MIT の初代ハッカー達がコード短縮に血道を上げていたことが記述されていた。もちろん、当時はコードの長さはダイレクトに実行速度に効いていたであろうから、コード短縮は現代のゴルフのように意味のない遊びではないが、「ハッカーズ」に出てきたコード短縮のお題は「ローマ数字をアラビア数字に変換する」などという、実用性から程遠いものであり、やはり「コードを短縮する」という行為自体に彼等は魅きつけられていたのではないかと考える。その他の例としては、伝統ある IOCCC [*6]でも、「短いコードで驚くほど複雑な処理ができている」ことから入賞した作品がいくつか[*7] [*8] ある。

また、最近日本で「Short Coding」[*9] という本が出版された。これは、 ACM/ICPC の練習用に北京大学が公開しているサーバで勝手に C でのショートコーディングを競って楽しんでいるコミュニティの中でも熱心な人が書きあげた本である。この本では、 C におけるショートコーディングについて、C 言語をうまく用いた小手先のテクニックから、コンパイラのクセや機械語についての記述、高速なアルゴリズム、短く記述できるアルゴリズムまで、網羅的に解説されている。

このように様々な人を魅きつけるゴルフであるが、その魅力はどのようなところにあるのだろうか。その魅力はやはり、文章などで述べるより、コードで示した方が伝わりやすいであろう。次章以降で、具体的な例をいくつか示す。

3. 暗号的記述

ゴルフ的に書かれたコードの特徴として、絶対に外せない、まず目につくものとしてあるのは、やはりその暗号的な記述であろう。例えばゴルフでは、 C を始め多くの言語で、`if-else` の構造を使わない。ほとんどの場合三項演算子などの方が短いからである。そして、その三項演算子の使いこなしにも様々なノウハウがある。例えば `printf("%d",a?b:(c++,d))` という C の式は、括弧を取って `printf("%d",a?b:c++,d)` とすると、`d` が第三引数になってしまい、意味が変わってしまう。しかしこのケースでも括弧を取る方法はあり、`printf("%d",!a?c++,d:b)` とすれば良い。

このようなテクニックが組み合わさり、ゴルフで書かれたコードは非日常性の強いコードとなる。各言語でのそういった細かいテクニックを見ていこう。

Perl, Ruby はそういった細かいテクニックの多い言語である。Perl で文字列を一文字ずつ見ていくコードを書く場合、Perl ゴルファーは通常の方法、つまり、変数 `$i` を `0` から `strlen` まで回し、`substr` で切り出す、という方法を一考だにしない。`s/./use $& here/eg` というイディオムをまず考えるであろう。`s/A/B/` は sed などにもある文字列置換の関数であり、`g` オプションをつけることによって複数回マッチさせることができ、`e` オプションをつけることによって置換する文字列を `B` の部分を Perl の式として実行した結果にすることができる。その際、 `B` の部分では `$&` をマッチした文字列として使える。もちろん置換の性質も利用できれば嬉しいが、単にループとしてでも `s///` を使うわけである。

Ruby でも Perl を参考に作られた部分も多いため、正規表現が強力であることをはじめ、似た形で使える機能も多い。Ruby で `a` という変数に標準入力を各行で分割してリストとして代入したい場合、通常は `a=STDIN.map` などとすると思われる。これは Perl の (ゴルフ以外でも使う) イディオム、`@a=<>` より大幅に長い。しかし Ruby は Perl 同様、よく使う変数は組

⋆1 http://perlgolf.sourceforge.net/
⋆2 http://terje2.frox25.no-ip.org/ golf-info/Book.html
⋆3 http://codegolf.com/
⋆4 http://golf.shinh.org/
⋆5 ISBN-10: 487593100X
⋆6 http://www.ioccc.org/
⋆7 http://www.ioccc.org/1995/makarios.c
⋆8 http://www.ioccc.org/2000/natori.c
⋆9 ISBN-10: 4839925232

込み変数として 2 文字の変数であらわせるようになっており、STDIN は $< に書きかえることができる。さらに Ruby の引数を展開する記法により、a=[*$<] と書きかえられる。ここで、 * は自動的に .to_a を行うため、IO オブジェクトを配列に変化させてくれている。この記法を用いると、標準入力を標準出力にそのまま出力するプログラムは puts *$< などと書け、これは Perl の最短コードよりも短い。

次に Python を見てみよう。Python は誰が書いても綺麗なコードになることを意識しているようだが、ゴルフされたコードは読みやすいとは言いがたい。以下のコードは標準入力から読み取った各行を 2 行ずつ出力する Python プログラムである。

```
l=''
while 1:l=raw_input(l+l)+'\n'
```

決して読みやすいコードではないが、この例はまだ単純なコードで、肝は raw_input が引数を出力してから入力を 1 行受ける、ということにある。初回は出力したくないため、l='' として空文字列を入れているのである。ちなみに終了しないように見えるが、入力が無くなった時点で例外終了する。

ここまでは短くコードを書けることが一つのウリとなっているスクリプト言語を見てきた。コンパイル言語も見てみよう。

Java には一つ、際立ったテクニックがある。Java で Hello world を短く書け、と言われた場合、

```
class C{
  public static void main(String a[]){
    System.out.print("Hello, world!");
  }
}
```

というようなコードを短くすることはできないように思われる[⋆1]。しかし、クラスの static イニシャライザを用いれば、

```
class C{
  static{
    System.out.print("Hello, world!");
  }
}
```

というように書ける。もちろん main が無いため例外が飛ぶが、既に目的の出力は得られているため問題無い[⋆2]。

OCaml でのゴルフは、普段のプログラムではバグから救ってくれる型推論をいかに騙すか、というゲームになる。例えば、

```
if a=1 then print_endline"one"
```

というようなコードは、ゴルフのイディオム通り if は使わず&& (OCaml では & が同義なのでその方が 1Byte 短い) を使いたいのであるが、

```
a=1&print_endline"one"
```

とすると型エラーとなる。& の右辺値が bool でなくてはならないが、ここでは unit であるためである。そこで、以下のように右辺値を無理矢理 bool にしてやれば良い。

```
a=1&()=print_endline"one"
```

Haskell は、シンタクスシュガーが多く、気の効いた標準関数がそろっているため、関数型言語としてはゴルフに強い。どの言語でも丸括弧は処理をしないのに 2Byte を必要とするため、目の敵にして除去にいそしむものであるが、Haskell は関数適用の $ や関数合成の . などで括弧を減らしやすい。一つ、ゴルフ以外に用途が無いと思われる、不思議な機能を紹介しよう。Haskell は関数の引数をパターンマッチさせた場合に@ で別名を定義する文法があるが、これは何故か関数名に対しても使えて、これを使うと例えば無限 Hello, world! を出力するプログラムは

```
m@main=putStr"Hello, world!\n">>m
```

などと書ける。

このように、7 つの言語に関してゴルフに使えるテクニックの例を見てきたが、それぞれの言語にゴルフ特有のテクニックが多くあることがわかっていただけたのでないかと思う。こういったゴルフイディオムを多用して作成されたコードは異様であり、その異様な見てくれのコードがゴルフの魅力の一つになっていることは間違いない。だが、こういった小細工の積み重ねだけがゴルフの技術であるかというと、実はそうではなく、簡潔に記述できるアルゴリズムを考える部分こそが問題の本質となることがほとんどである。次章では、ゴルフ的なアルゴリズムについて、いくつかの例を紹介したい。

4. アルゴリズム

4.1 delete blank line

最初に紹介した “delete blank line” 問題を Ruby で解くことを考える。この問題は普通に考えるとやはり、一行ずつ読んでいって、「空行でなければ出力する」、あるいは「空行であれば出力しない」、というようなロジックになる。つまり以下のようなコードで

⋆1 もちろん改行やインデントは除去可能である。以降でも可読性のためにインデントと改行は残してある。

⋆2 このテクニックは私のサーバでは Java が GCJ である関係で使えない

ある。

```
while l=gets
  puts l if=~/./
end
```

これを Ruby のゴルフイディオムを用いて素直に短縮すると、

```
print if/./ while gets
```

となり、 22Byte のコードである。Perl 由来の機能である、gets の結果が自動的に $_ に入ること、/./ などの正規表現が条件式の中にあらわれると$_=~/./ が実行されること、print の引数を省略すると $_ が出力されること、を用いている。while gets はよくある定型句なので、同様のことを実現するコマンドラインスイッチが Ruby や Perl には用意されている。それを用いると

```
#!ruby -n
print if/./
```

となり、 21Byte まで短縮できる。逆に、「空行なら出力しない」方向では、

```
#!ruby -p
sub /^
/,""
```

というようなコードや

```
#!ruby -p
next if/^
/
```

というようなコードに辿りつく。/^\n/ は行頭が空行であるかどうかを確認するための正規表現で、\n を使うより本物の改行の方が短いため少し気持ち悪いコードになっている。-p というコマンドラインスイッチは-n に加えて、最後に自動で print してくれるものである。

これらのコードは全て 21Byte で、色々と小細工を加えてみてもこの種の方針ではこのあたりが限界のようである。だが、この問題ではもっと大幅な短縮が可能である。Ruby には配列と配列を引き算すると集合としての差を求めることができる。例えば [1,2,1]-[1] は [2] になる。この機能を用いるとこの問題の回答は以下のように大幅に短くなる。

```
$><<[*$<]-[$/]
```

全て記号であるという暗号的な記述であるが、14Byte と非常に短くなった。$><<$ は print のようなもので、$/ は改行が初期値に入っている特殊変数である。一旦全ての入力を読んでから処理をする方が入力を順次処理するよりも短いことはゴルフでも多くなく、少し気付きにくい盲点であった。

このように、基本的な方針が同様のまま前章で述べたような小細工を繰り返して 3,4Byte を稼ぐより、根本的な方針を見直した方が大幅に縮むことが多い。その際に、高速に動くアルゴリズムを考えるよりもゴルフ特有の制約条件が関わってきやすいことや、そもそもコードを短く一般的な指針というものが存在していないため、パズルとしての奥の深さを増やしている。感触としては高速なアルゴリズムはゴルフとしても短くなりがちであると感じているが、必ずしもそのルールが成り立つわけでは決してない。

4.2 infix to postfix

ゴルフ特有のアルゴリズムとして、非常に鮮かなものを紹介する。“infix to postfix” 問題[⋆1]は、中置記法で書かれた式 (例えば a*b*(c+d)+e) を標準入力から受け取り、後置記法の式 (今回の例では ab*cd+*e+) にして出力する問題である。この問題は普通に考えて解くと簡単な再帰下降パーサを書いて解くことになるだろう。特にこの問題に対しては、再帰下降の再帰を展開したような形である Shunting yard algorithm [⋆2] というのが知られているらしく、それを用いて Ruby で書かれたコードが以下のものである。

```
#!ruby -p
$\="(
"
z=/([^(]*)/
gsub(/\W/){|c|
  $\["+-"[c]?z:"("[c]?//:c>")"?/([*\/]*)/:
    /#{c="";z}\(/]=c;$1
}
```

このコードは一見して何をやっているかよくわからないが、実のところ私も一度解読したものの忘れてしまったため、これを書いている時点で何をやっているのかよくわかっていない。ただ、 Ruby のゴルフのエッセンスがふんだんに用いられていることは間違いなく見て取れるし、この 104Byte のコードは明らかに短い。

しかしこのコードはもっとシンプルなアルゴリズムを用いることで大幅に縮めることができるのである。そのアルゴリズムとは、アルゴリズムと言うべきかは疑問ではあるが、中置記法のパースを Ruby に任せてしまうというものである。Ruby は元々他の多くの言語と同様、このような数式を中置記法で書くことができる。つまり入力を eval してしまえば望む順番で

⋆1 http://golf.shinh.org/p.rb?infix+to+postfix
⋆2 http://en.wikipedia.org/wiki/Shunting_yard_algorithm

評価が行われる。しかし、何の工夫もなく a*b+c などを評価しても a や b が未定義の変数であるというエラーが発生するだけである。ここでもう一工夫があり、未定義のメッセージが飛んだ場合のハンドラである method_missing を定義するのである。メソッドの中では単にシンボルの名前を出力するだけで良い。そうやってできたコードが以下の 61Byte のコードである。

```
undef:p
def method_missing s,*
  $><<s
end
loop{eval(gets)<<$/}
```

このコードもまだ Ruby ゴルフのイディオムがいくつか使われているが、先程の 104Byte よりははるかに読みやすい。undef:p は Ruby が p というメソッドを最初から持っているので未定義に戻しているだけであることがわかれば、処理はだいたい読めるのではないかと思う。

5. 何でもアリ

前章では、綺麗なアルゴリズムがゴルフにもしばしば有効である、というようなことを主張したのだが、そうは言ってもアルゴリズムの美しさで競っているのではなく、バイト数というある種いびつな基準で競っているのであるから、呆れるようなコードがしばしば投稿される。

例えば “Hello, world!” を出力するだけという単純な問題に、“Hello, world!.rb” というようなファイル名で、$0 (ファイル名が入っている特殊変数) を用いて出力させようと試みた人がいた。非常に面白いハックではあるのだが、残念ながら私のサーバでは実行前にファイル名を変更しているため、そのコードは通ることはなかった。

5.1 入力依存と乱数

ゴルフではしばしば、出題者の意図通り解かない方が短くなるような問題が出てくる。例えば「1000 以下の素数の数を求めろ」という問題が出たら、少し出題者に申し訳ないな、とは思うが、

```
main(){puts("168");}
```

といった回答を提出するだろう。複数テストケースがある問題で入力を真面目にパースするよりも乱数などを用いて何度も投稿し、たまたま全てのテストケースでうまくいくことを目指すというような手段に出ることもある。

そのような邪悪なコードの最たるものとして作られた問題が、“123” [⋆1] という問題である。この問題には入力は全く無いが、プログラムは 1 つ目のテストケースに対して 1 を、2 つ目に 2 を、3 つ目に 3 を出力しなければならない。この問題が作成された次の日は私のゴルフのサイトのアクセス数が 20000 程度であったが、そのうち 12000 がこの問題への POST であった。

この問題は都合良い数値の出てきうる毎度変わる数値を探すことになる。そのようなものとしては、

- 乱数
- 時間
- プロセス ID
- スタックやヒープの変数のアドレス

などがあるようであった。

5.2 Minesweeper

“Minesweeper” [⋆2] 問題はその回答が邪悪なコードになっている問題として特に印象深いものである。問題の要求は、爆弾の位置を与えられてそれ以外のセルに回りの爆弾の数を入れて出力せよ、というものである。具体的には、

```
........
........
....***.
...**.*.
..**....
...*****
...*.*.*
...*.***
```

のようなものが標準入力から与えられ、それに対して

```
00000000
00012321
0013***2
013**5*2
01**6543
014*****
003*7*8*
002*4***
```

を出力すれば良い (つまり*が爆弾である)。確かにそこまで難しい問題ではないが、この問題に対して Perl ゴルファー 4 名から出された回答は、以下の 36Byte である。

```
-p0 s!\.!map/"/g,$‘.666e6x3&$^.$_!eg
```

ただし、 Perlgolf の流儀に従って表記していて、最初の空白までの -p0 はコマンドラインスイッチである。

⋆1 http://golf.shinh.org/p.rb?123

⋆2 http://terje2.frox25.no-ip.org/ pgas/score.pl?func=rules&hole=30&season=0

他の多くのゴルフコードと同様、一部ずつ理解していかないとこのコードは解釈できない。実際、この回答を出した tybalt89 氏自身が IRC で「このコードは何をしてるんだ?」と聞かれて彼が書いたコードであるにも関わらず、「すぐにはわからん」と答えているところを見たことがある。

まずコマンドラインスイッチの`-p0` は標準入力を全て Perl で多用される変数である`$_` に代入するものである。次に `s!\.!XXX!eg` は 3 章でも軽く述べた、置換の正規表現の構文であり、`.` にマッチしたら `XXX` を評価してその結果に起き変える、という意味である。

`XXX` のコードの中身に適切に括弧を入れてみると、`map(/"/g,($‘.(666e6x3))&($^.$_))` というようになる。順に見ていこう。`$‘` は正規表現のマッチした部分の左側が格納されている特殊変数であり、`666e6` は浮動少数で `666000000` 、`x3` は左辺値文字列を 3 回繰り返す。まとめると、この文字列は

```
........
........
....***.
...**.*.
..**.666
00000666
00000666
00000
```

というようなものである (実際には改行が `0` に置き換わっている部分が 3 箇所あるが)。

次に `$^.$_` であるが、 `.` は文字列の接続演算子であり、`$^` にデフォルト値として入っている “`STDOUT_TOP`” という文字列と標準入力から読んだ文字列を接続している。つまり以下のような文字列になっている。

```
STDOUT_T
P.......
 .......
 ....***
 ...**.*
 ..**...
 ...****
 ...*.*.
 ...*.**
```

入力が横幅 8Byte であり、改行を入れると 9Byte であるため、10Byte の “`STDOUT_TOP`” という文字列で 1 行 1 列だけずれたのがポイントで、このおかげで先程用意した `6` の部分がちょうどチェックしたい部分になっている。

次に `&` であるが、これはマスクを行うために使われていて、文字列に対して使用されると恐ろしいことに、個々の文字ごとに論理積を取る、という動作になる。この演算の後の文字列は

```
????????
?.......
 ...***.
 ..*****
 .***"&"
     &&&
     """
```

というようなものになる。つまり今見ている場所の回りの `*` を `"` に変化させることに成功したわけである。

後は `"` の数を数えれば良いが、それが `map` の部分である。`map` は第二引数のリストを一つずつ `$_` に代入して、そのたびに第一引数を実行して個々の結果のリストを返す関数である。ここでは第二引数はリストではなく一つの文字列であるが、Perl なので問題無い。`map` の第一引数となっている、`/"/g` は、暗黙の変数 `$_` とマッチを行い、マッチしたもののリストをかえす。つまり `"` の数だけ `"` の入ったリストになるわけである。`map` は演算の結果をリストに入れていくので、リストのリストになるのではないか、と思われるが、 Perl ではリストのリストは自動的にリストに平坦化されるので `map` の部分は全体としてリストの値である。最後に、置換を実際に行う時、このリストはスカラコンテキストで評価されるため、リストはスカラコンテキストで評価するとそのサイズとなる、というルールに従い、めでたく問題の仕様通り、数値が入ることになる。

5.3 qsort

もう一つゴルフのなんでもアリさをよくあらわす例を紹介する。C 言語の `qsort` をどれだけ短く使えるか、という話である。まず普通に `int` をソートする例を考えると、

```
int cmp(void *a0, void *b0) {
  int *a = a0
  int *b = b0;
  return *a<*b ? -1 : *a>*b;
}
int main() {
  // ...
  qsort(array, size, 4, &cmp);
}
```

というようなコードになるのではないかと思う。この `cmp` という関数を短く書くことを考える。

まず、配列に大きい int が入っていなくてオーバーフローしないと仮定して、型宣言を省略してやると、

```
cmp(int*a,int*b){
  return*a-*b;
}
```

というように書くことができる。

次に二回書かれていて冗長な型宣言 int* を削除することを考える。これは普通に考えると消すことができないが、アーキテクチャに依存して考えると、変数 a と変数 b はスタック上の隣のアドレスにあるだろうことを利用して、

```
cmp(int*a){
  return*a-**(&a+1);
}
```

と書くことができる。ここで引数を後ろから順にスタックに積んでいき、スタックが上に伸びるアーキテクチャであることを仮定している。さらにこの **(&a+1) は滅多に使わない表現であるが C では array[index] と index[array] が等価であることを利用して、

```
cmp(int*a){
  return*a-*1[&a];
}
```

と、ここまで短くなる。

しかし、アーキテクチャ依存を始めてしまった以上、まだまだ縮めることができてしまう。x86 では返り値は EAX を通じてかえされるが、つまり結果がたまたま EAX に入ってくれれば return などという長い予約語を使う必要は無いのである。これはアーキテクチャだけでなくコンパイラにも強く依存するが、グローバル変数やローカル変数に代入しておけば EAX に入ってくれることが多い。つまり、

```
cmp(int*a){
  a=*a-*1[&a];
}
```

などとしてもうまくいったりするわけである。

このあたりで限界か…と思われるのだが、そもそもコールバック関数を作らないという方法がある。他の言語でいうところの無名関数を使えば良い。C 言語には明らかに無名関数は無いように思えるが、要は機械語を埋め込んでしまえば実現できる。つまり、

```
int main() {
  // ...
  qsort(array,size,4,
        "YXZQQQ\x8b\0+\x02\xc3");
}
```

などとすれば、OS がデータ領域の実行を許してくれれば、x86 依存でうまく実行できてしまう。ASCII 以外を受けつけてくれるサーバであれば、\xc3 などはバイナリ文字列にしてやれば良い。この命令列は、

```
0:   59              pop    %ecx
1:   58              pop    %eax
2:   5a              pop    %edx
3:   51              push   %ecx
4:   51              push   %ecx
5:   51              push   %ecx
6:   8b 00           mov    (%eax),%eax
8:   2b 02           sub    (%edx),%eax
a:   c3              ret
```

という命令を実行している。32bit マシンで 11Byte で 9 命令を実行できているのは CISC ならではという感がある。この命令列はコンパイラと libc の実装次第でもっと短くできる可能性があり、例えば EDX に必ず第一引数が入るようになっている、などの実装依存があったりするかもしれないので、デバッガなどを用いて確認していくことになる。しかし、ここではこのあたりで終わりにしておくことにする。

6. 特殊な言語でのゴルフ

ここまで見てきたのは比較的普通の言語でのゴルフであったが、やや特殊な言語でのゴルフも盛んに行われている。

6.1 PostScript

印刷機にデータを送るのに使用されている言語であるが、立派にチューリング完全なスタック言語であるどころか私のゴルフ場では非常に好成績を収めている (現在 Perl,Ruby に次いで全言語中 3 位)。私自身は PostScript には詳しくないのであるが、PostScript は大抵のことを 2Byte でエンコードできるバイナリ表現があって、それが強いそうである。

6.2 sed

置換の正規表現しか無いような言語であるが、これも一応ループが書けるため、どんなプログラムでも理論上書くことができる。実際、 dc(1) の sed 版なども書かれており[⋆1] その意外な力を知ることができる。とはいえ、複雑な問題は「すごく頑張れば解ける」程度であり、本当に力を発揮するのはやはり単純な問題である。例えば、何度か出てきた “delete blank line” 問題を sed で解くと以下の 5Byte になる。

```
/^$/d
```

⋆1 http://sed.sourceforge.net/grabbag/scripts/dc.sed

6.3 Brainfuck

Brainfuck はミニマル言語としてはとても有名な言語で、基本的にはチューリングマシンそのものといった言語である。この言語の仕様を見ると簡単な最適化コンパイラが記述できるように思えるのだが、メモリを次々と使い捨てにしていく (つまりループを回るごとに変数のアドレスが変わっていく) ようなコードが最短になる場合も多く、なかなか難しいのではないかと思う。以下に “delete blank line” 問題の Brainfuck での回答例を示す。

```
,+[-[-<+<+>>]
<----------[<.>>>>]
<<[>.>>]
>>,+]
```

ここでは、ループに入る時に真だった場合と偽だった場合でループ終了時に差しているアドレスが違うということが普通に行われており、1Byte 読むごとに前回のメモリーセルは捨てて右の番地に進んでいる。

6.4 Befunge

2D 空間にプログラムを記述し、プログラミングカウンタも 2D 空間を縦横無尽に走りまわる言語である。回答を 2D 空間に配置しているにも関わらず Byte 数で得点を計算しているため、「改行コードの 1Byte がもったいないのでなるべく横長に」「間を埋める空白がもったいないのでなるべく左揃えに」という一風変わった制約がかかる。“delete blank line” は以下のようになる。

```
#v_>~:52*-:#v_$\
0< ^,  _@#:\<
```

シンプルな言語としては命令の種類が多く、ゴルフでもそれなりに強い。例えば cal(1) を Befunge でそれなりに縮めつつ書いてみたところ、以下のように 253Byte になった。

```
v144725736146
0&#&%&%&&%&%&
>25*"aS rF hT eW uT oM uS">:#,_&v
v+g0p00:&--+/4:/**455:/***2558::<
>\:::554**%!\4%+\8552***%*!\v
v:%7-p12+g12*!-2g00:*!‘2g00$<
>v  >1-"   ",,,
v>:#^_
>1+:: 9‘!#v_   v
 *52*+,\:v>" ",>.\1+:7%!!994++
|--7g1g00<
>52*,@
```

6.5 機 械 語

私のサーバでは、 ELF バイナリをそのまま投稿することもできるようになっている。ELF に関わらずネイティブな OS のオブジェクトコードはあまり強いチェックが入っていないことが多く、結果として、ELF ヘッダとプログラムヘッダと実行コードが重なって入り乱れることになる[⋆1] 。

6.6 そ の 他

コードを短かくする、ということはパズルとしてはわかりやすいため、特にサーバで自動回答チェックして記録を残す…というようなことをしなくても頻繁に行われる。以下にいくつか私が知っている例を示しておく。

ICFP の 2006 年度のコンテスト[⋆2] では、UMIX という仮想的な OS の中で、ユーモアに満ちたプログラム言語がいくつか実装されており、そのうち三つの言語でコードを短縮することによって得点が入るシステムになっていた。

数学セミナー 2007 年 9 月号の「エレガントな回答を求む」というコーナーでは引き算の無いことが特徴であろうミニマルな言語仕様を与えられ、その言語を用いて二変数の最小値を求めるコードの最短化を競う問題が出題されていた。

SPOJ というサイトにはソースコードがクリスマスツリーの形をしている Quine を最小化する、という問題が出題されていた[⋆3] 。

7. 質　　疑

Q. 「パー」はあるのか

A. 知る限り無く、面白そうだが設定が難しそう。みなで競っているうちに当初の予想に反して驚く程縮んでしまうケースが多いため。

Q. アルゴリズム的に綺麗な例は？

A. ピクロスを解く問題[⋆4] などが綺麗で、そもそも高速なものを書かないと通らない。(会場では高速なものが短いケースが多い、と主張しましたが、考えなおしてみると、4.1 章の最後に書いた通り、やはりそうでもないケースも多いかな、と思います)

⋆1 http://shinh.skr.jp/obf/bingolf.html
⋆2 http://en.wikipedia.org/wiki/ICFP_Programming_Contest
⋆3 http://www.spoj.pl/problems/CTQUINE/
⋆4 http://codegolf.com/paint-by-numbers