WORD

2010.09 From College of Information Science

15

非実在WORD編集部号

★☆★☆★☆★☆★☆★☆★☆★☆★☆★☆★

祝・巻頭カラー！

なんて
もってのほか！

おかげさまで戸籍上最高齢の学類誌!

よくもこんな詐欺タイトルを！一般的な事はともかく、専門的な記事に
定評のあるWORDが今号もやりたい放題！読者の皆さん、お許し下さい！

★☆★☆★☆★☆★☆★☆★☆★☆★☆★☆★

非実在WORD編集部号
目次

※この目次には非実在記事が混ざっています

GRな日々。 IV

文 編集部 葡萄酒

「絞り」の極意（？）

前号の発行が終わり、この記事を書くために今までの原稿を読み直していたのだが ――――― 私はとある事に気付いてしまったのだ。

「いつの間にか、このシリーズは写真記事ではなく旅行記事になっていないか？」

旅行記事を否定する気は毛頭ないが、私が書きたかったのは写真の記事であった筈なのだ。もし今号も旅行記事だと期待されていた読者の方がおられたのならば申し訳ないが、初心に立ち返り写真に関するコラムを書く事にする。

一口に写真にまつわるコラムと言っても様々なトピックがあるが、今回のテーマは「絞り」である。ちなみにこの記事は、夏休みに新しくカメラを買ったり、バイトで貯めたお金でカメラを買おうと考えている読者(つまり専門用語で言うところの「初心者」)向けとなっている。そのため既にバシバシ写真を撮られている方には物足りない内容であるかもしれないが、しばらくお付き合い頂こう。

さて、本題に入ろう。写真を撮る際にカメラの設定は欠かせないが、何と言っても重要なのは「シャッター速度」と「絞り」である。他にも ISO 感度の設定やホワイトバランスの設定などがあるとは言え、これはデジタルカメラ特有の仕組みであり、写真の本質ではない。あくまで「ファインダー越しに世界の一瞬を切り抜く」ことが「写真を撮る」ことの本質であり、それは即ち「光を捉える」ということである。ならば「写真記事」と言い張るこのコラムで語られるべきは、撮影素子云々よりも世界と直結するレンズとシャッターであろう。そのなかでも初心者が感覚を掴み辛い「絞り」に焦点を合わせようと思う。動作が目に見えるシャッターと違い、動く実体のない「絞り」の写真に対する影響は直感的にイメージし辛いのではないだろうか？

そもそも「絞り」とは何だろうか。一眼レフのレンズを見れば一目で分かるのだが、コンパクトデジカメやオート絞りのレンズでは分かり辛いかもしれない。簡単に説明すると、「絞り」とはレンズ内部にある「絞り羽根」と呼ばれる複数枚のパーツから成る環状の部品、もしくはそれによって形成される環の大きさの度合いを指す。絞りのダイヤルに連動してこの「絞り羽根」が動き、レンズに入る光量を調節する ――――― 人間の目に例えるならば瞳に相当する部分である。

通常、絞り羽根の枚数は六枚から九枚ほどである事が多いが、ビデオカメラなど特殊な用途のものだと菱形二枚羽根絞りなど変則的なものも見受けられる。この枚数はレンズの特徴を見る際に非常に重要な要素であり、例えばピントが外れている部分のボケ味に強く影響する。このボケ味は絞りが円形に近いほど綺麗に描写されるため、一般的には枚数の多い羽根が好まれる場合が多い。

ここまで「絞り」の構造を見て来たのだが、ここで根本的な疑問が生じる。「何故絞る必要があるのか？」という事である。単純に明るさを調節する機構として考えれば、絞りなど無い明る

いレンズの方が望ましい筈だ。なぜならシャッター速度は早ければ早いほどブレが発生しにくく、そのためにはより多くの光を取り入れられる方が有利だからである。長くシャッターを開けて動きを表現したいのであれば、強引ではあるがレンズに ND(減光)フィルターを装着すれば良い。にもかかわらず、絞りがここまで重要視される理由とは何なのだろうか？

以下の写真をご覧頂こう。限界まで絞った状態と開放した状態でシャッター速度を調整し、同じ明るさに近づけて比較したものである。

RICOH GR Digital II / F9.0 で撮影時

RICOH GR Digital II / F2.4 で撮影

ボケ具合が大きく変わった事がお分かりいただけるだろうか？

このピントが合っている距離の範囲を「被写界深度」と呼ぶ[*1]。絞りが重宝される理由のひとつが、この被写界深度を調節する役割なのである。

物理的な説明は割愛するが、絞りは開放すれば被写界深度が浅くなり、逆に絞るほど被写界深度は深くなる。被写界深度は絞りだけで決まる訳ではない(レンズの焦点距離も関係する)が、同じ大きさ・構成のレンズであれば、より絞りを開けられる=F 値[*2] が小さく明るいレンズのほうが、より強いボケの効果を得る事ができると言えるだろう。逆に、極端に被写界深度が浅くなってしまうマクロレンズを使用する時は、強めに絞る事で被写体全体にピントを合わせる事ができる。

一般的な一眼レフのレンズはコンパクトカメラと比べて焦点距離が長く、大きなボケが得られるものが多い。そのため初めて一眼レフを持ったときに「嗚呼、素晴らしい写真が撮れたなあ」と感動する儀式が執り行われる場合が多いが、実はこの儀式がネックとなる。単純な話、一眼レフを持てば、どんな素人でも「何となく良い雰囲気の写真」は撮れてしまうのだ。しかし、そこから一歩進んだ「自分の写真」を撮るのは非常に難しい。下手に知識を身につけて高い機材を使い始めた途端、感性の赴くままに適当なカメラで適当に撮影していた頃の写真を越えられなくなったというのはよく聞く話である。その壁を越えるために、撮影技術的な側面から「ボケ」という技法を見て行こう。表現としての「ボケ」が持つ意義を理解して効果的に狙う事ができれば、

*1 被写界深度が狭いことを一般的に「被写界深度が浅い」と表現する。広いことも同様に「深い」と呼ぶ。

*2 レンズの焦点距離を有効口径で割った値のこと。

「何となく良い雰囲気の写真」から「良い写真」に進む手助けになるだろう。

まず、ボケが持つ強みについて考えてみよう。写真は視覚芸術の一つである以上、表現者から鑑賞者へのアプローチは一方通行である。即ち作品を解釈するのは鑑賞者の側であり、表現者の意図を強制するのは難しい。その点では、作品の中心に据える被写体を際立たせる事で鑑賞者の意識を意図的に向けるという特性は大きな強みとなる。さらに、絵画との違いとして「写したくないものでも、そこにあれば写る」という縛りを緩和する事ができる。花を接写する時など良いアングルから撮ろうとした時に限って、例えば綺麗に咲いていない花が背後に写り込んだりしてしまうものだが、これをボケによってある程度見えなくすることができるようになる。

さらに副次的な特性として、特殊な「空気」の質が写真に出るようになるというのが挙げられる。強いボケを効かせた写真は、どことなく柔らかな雰囲気になるのが良い例だろう。これを目的としたボケの手法は、主に女性を被写体とする人物写真で有効に用いられることが多い。また、流し撮りと同様に被写体と周囲の対比が、ある種の「非現実感」を醸し出す事もある。こちらは花などの小さいものを撮影する場合に特に有効である。

しかし、何でもボケた写真を撮れば良いという考えは間違っている。そもそも、技術としてのボケはそんなに簡単なものではないのだ。人物写真程度の距離があるならいざ知らず、マクロレンズで接写する際など被写界深度は数ミリ程度になることもある。シャッターを切る時に、手ブレ補正の効かない前後のブレを数ミリ以内に抑えられるだろうか？

また、前述の特性が逆効果になることがある。特に風景写真や建物の写真などは、被写体の雰囲気を生かすためにパンフォーカス[*3]で撮るべき場合は多いだろう。

これらの特徴から、やはり「ボケ」を状況に応じて駆使するためには「何を表現したいのか」というヴィジョンを明確に持つ必要があるのだと考える。私の感覚では、「場所」を撮るのか、それとも「モノ」を撮るのかということを意識するようにすると、構図や表現のイメージを固めやすい気がする。表現のセンスを鍛えるためには、写真雑誌のコンテストの寸評を読んだりして良い写真の良い所を参考するのが良いだろう。勿論一枚一枚に真摯に向き合いつつも、量を撮る事も大切である。

ここまで「絞り」から「ボケ」までの流れで話を進めてきた訳だが、お楽しみ頂けただろうか？まだまだ私も人に自慢できるほどに上手な訳ではないが、これから写真を始めようという方への助力となる事ができたのならば幸いである。

*3 全体にピントがあった状態を指す。

パソコンからケータイサイトを見る

文 編集部 zer0day

はじめに

どうも、zer0day です。あのパソヲタです。

今日、スマートフォンが勢力をつけてきていますが、依然として携帯電話向けのサイトは、いわゆるガラケー専用となっており、スマートフォンや PC からの閲覧が拒否される状態です。

私はこれの意味が分かりません。

スマートフォンユーザーとしてはかなり嫌な風潮です。例えば mixi に登録したくても、例のガラケー登録のせいではじかれるんです。

一応は PC からの荒らし対策や、コンテンツの盗用防止、mixi とかの携帯認証は携帯電話の固有 ID を使って 1 人 1 アカウントのルールを維持する[*1] という建前があるみたいですが、スマートフォンや PC からのアクセスを拒否することで絶対的な客数が減り、市場がやせるような気がします。

「このサイトは携帯電話からのみ閲覧できます。」

腹たつわー……。

ということで、PC からガラケー用サイトに侵入してみることにしました。

私の環境は次のとおりです。

パソコン：MacBook Pro（Snow Leopard）
ブラウザ：Firefox 3.6.8 - アドオン：User Agent Switcher
携帯：SoftBank X05HT（Windows Mobile 6.5 改造 ROM / Android 2.1 Dual-boot）

なお SoftBank のスマートフォンを使用する際は、銀 SIM を搭載した機種（つまり iPhone 以外）である必要があります。

やりかた for Windows Mobile 6 on SoftBank X05HT

なんとかして携帯と PC をネットワーク的に接続したいのですが、キャリア会社の ROM のままではインターネット共有ができないんですねー。自由度低すぎ。でもできるんです。

というわけで、ROM 改造無しの方法を紹介します。しばらく純正 ROM は触ってないので間違ったこと言いそうだけど……。

1.ガラケー回線用の接続情報を携帯に追加します。ググれば出るんですよこれが。検索キーワードは「X05HT wasabi」です。docomo, au についてはガラケー回線用の公開情報が著しく少ないので今回は SoftBank のみの説明になります。[*2]

APN: mailwebservice.softbank.ne.jp
User: softbank
Password: **********
Proxy: sbwapproxy.softbank.ne.jp:8080 HTTP/HTTPS

私はここで SoftBank の機密かもしれなかった情報を流すのはアレだと思うので各自「X05HT wasabi」でググってみてください。q で始まる文字列が PW です。

2.手動で携帯側の APN へ接続テストします。/* 純正 ROM でできたっけ・・・？汗*/

3.DeleGate（http://wince.delegate.org）をダウンロードし、手動で携帯側の APN に接続した後、DeleGate を実行します。これでお使いの Windows Mobile を HTTP/HTTPS プロキシ化します。

---

1 ユーザーによるトラブルの際に ID で個人を特定するためともいわれます。

2 docomo はデータ通信用 FOMA でもいけるという噂。研究中です。

4.Bluetooth を経由して、PC と携帯を相互に認識させます。
5.携帯電話に任意の IP アドレスを割り振ります。「設定＞接続＞ WiFi ＞ネットワークアダプタ＞ Bluetooth PAN Adapter」から。
6.PC 側から Bluetooth PAN を確立させます。
7.ブラウザを開き、HTTP, HTTPS プロキシの欄に、5 で設定した IP アドレスを入力し、ポートを 8080 にします。
8.ブラウザのユーザーエージェントをいじります。
例：SoftBank 708SC
SoftBank/1.0/708SC/SCJ001 Browser/NetFront/3.3
9.これでおｋｗｗｗｗｗｗｗ

やりかた for Android 2.1 on SoftBank X06HT

接続情報は一緒です。APN設定から先ほどの接続情報をぽちぽち投入します。
こっちはテザリングがデフォでできるのでDeleGateやBluetoothが要りません。
USBなりWiFiなりでPCにつなげて[*3]、PC側のブラウザのユーザーエージェントをいじればOK。

やりかた for iPhone

残念ですがiPhoneはSIMカードが違うため、ガラケー回線とは隔離されています。つまり無理です。
ちぇーっ。

トラブルシューティング

Q.　つながらない
A.　黒SIMではないですか？
PCとの接続は確立されていますか？
認証情報は正確ですか？
プロキシ設定は間違っていないですか？

Q.　携帯会社から文句きた
A.　当記事における記載事項を各自で実践する場合は全て自己責任でお願いします。
↑重要

実験結果

というわけで、今回の餌食となるのは某みくちー。新規登録[*4]時の携帯認証を、スマートフォンとPCを使って突破してみました。

---

1 Androidマーケットにあ るEasyTetherがオススメ。
2 実は私つい先日まで退会している状態だったので、出戻りも兼ねて登録してみました。

パソコンからケータイサイトを見る

うひょひょひょwww

w w w w w w w

登録できたw w w w w w w

みくちメンバーが増えるよやったねたえちゃん！！！！！

考察

携帯専用と思われていたサイトに、実はPCを通じて行けるということは、

- ユーザーエージェントに含まれる携帯の固有IDは偽装可能であり、これによる認証は突破、偽装されうる。対策は各キャリア会社のゲートウェイで付加されるHTTPヘッダを固有IDとして取り扱うこと。
- 携帯からは打てない文字や任意のHTTPヘッダを送信することができる、HTMLソースコードを表示させることができるなど、諸々のセキュリティ問題が表面化しうる。携帯サイトだからといって甘い造りは厳禁。

ー完ー

参考：「HTC TouchPRO SoftBank X05HT Part4」
http://anchorage.2ch.net/test/read.cgi/keitai/1229784609/

電子の歌姫はアイドルの夢を見るか

文 編集部 Genyakun

はじめに

去る 8 月 31 日にとある歌手が 3 歳になりました。その歌手の歌声は多くの人を魅了し、わずか 3 年の間に幾多もの曲が作られ、人々に「歌姫（アイドル）」と呼ばれるまでになりました。それこそが、ヤマハが開発した音声合成技術「VOCALOID2[*1]」を使用したソフトウエア、「初音ミク[*2]」です。

初音ミクはソフトウエアにもかかわらずキャラクター性を持たせたバーチャルアイドルとしてブレイクし、その歌声とキャラクターは音楽は元よりイラストや動画、ゲームなど多くのメディアに展開しています。

この記事では、その中でも SEGA が開発したゲームである「初音ミク -Project DIVA- 2nd」の攻略記事となっております。

初音ミク -Project DIVA- 2nd とは？

初音ミク -Project DIVA- 2nd（以下 DIVA 2nd）とは、（あらゆる意味で）有名なセガが~~何を間違えたか~~作った初音ミクをメイン歌手としたいわゆる音ゲーの（PSP では）2 つめのタイトルです[*3]。
DIVA 2nd で特徴的なのが「リアルタイムレンダリング 3D の PV」です。似たようなゲームでは過去に NBGI[*4] のアイドルマスターがリアルタイムレンダリング 3D をゲーム内の PV に採用していましたが、タイミング判定のシビアな音ゲーとしては珍しい方法を採用しています。

肝心のゲームの中身ですが、本作は音ゲーの割には画面に出てくる形と同じボタンを押すだけと操作が簡単な上に、高度なテクニックを要求されるような譜面はあまり無いため音ゲーが苦手な方でもプレイすることが可能です[*5]。また、曲数も 46 曲[*6]、難易度は EASY, NORMAL, HARD, EXTREME の 4 段階があるためすぐクリアする、ということはまず無い[*7] 上に、PV 鑑賞や自作の PV や譜面を作ることが出来る機能が存在するため、費用対時間の面でも大変お得です[*8]。

---

*1 実は VOCALOID という単語はヤマハ株式会社の登録商標だったりする。
*2 エンジンはヤマハから提供されているが、音声 DB と発売元はクリプトン社となっている。
*3 2nd と言うからには 1st（無印）も存在しますが、これは後で説明します
*4 バンダイナムコゲームス
*5 リズムに合わせてというより歌詞に合わせてボタン押す感じ
*6 チュートリアルで使用される楽曲を除く
*7 楽曲ごとに HARD を出すには NORMAL をクリア、EXTREME を出すのには HARD をクリアすることが必要
*8 筆者はすでに 80 時間ほどやりこんで、毎日のようにプレイしているが決して中毒ではない。

画面を見てみる

ということでまずは画面を見てみましょう。おそらく皆さん聞いたことがあるであろう「メルト」を難易度 EASY でプレイしたときのサビ部分はこのようになっています。

EASY で使用するのは〇キーだけで、この画面にはありませんが全譜面共通である長押し[*9]と同時押し[*10]の練習がメインです。また、大抵の曲の攻略にはあまり関係がありませんが、一部の高難易度曲では左下のソングエナジーゲージ(以下エナジーゲージ)が割と大切になってきます。このゲージはコンボをつなげているといつの間にかに増えて、ミスをすると減っていきます。無くなるとその時点で曲が中断してしまいます。さらに、画像では見えにくいですがこのゲージの外側にあるゲージは現時点での成績が暫定的に表示されています。大体は終盤に突然モリモリ増えるので参考程度に使いましょう。

基本的な攻略方法

まず基本的な攻略方法についてですが、次のようなことがおおまかに挙げられます。

・原曲を聴く

歌詞がそのまま譜面になっているので原曲を一度聴いておくと楽でしょう。

・マーカーをちゃんと確認する

個人的な感覚ですがタイミング判定が早押しについては**特に厳しい**[*11]ので注意しましょう。逆に遅く押す分には判定が緩いのでマーカとターゲットマーカが重なったのを確認したくらいで押してもコンボがつながります。

・同時押しの連打は片方のキーを押しっぱなしで対処可能

同じキーの同時押しのマーカーが連続している場合には、片方は入力しっぱなしでも問題なく判定されます。たとえば、「→→→」のように→型の同時押しのマーカが連続して来た場合

*9 普通のマーカが長く伸びてるやつ。初めのマーカで押し初め、終わりのマーカでボタンを離す
*10「←」のような形をしたマーカで、基本的に同じ方向のボタンを押す。例えば〇なら右矢印ボタンが対応している
*11 焦れば焦るほどコンボが切れるので連打も気持ち遅めにした方が良かったりする

は矢印キーを**押しっぱなし**で○キーだけをマーカ数分押せば[*12] 処理が可能です。これの応用で同時押しに挟まってる別キーのマーカに対しても同じような事[*13] をすることが可能です。

・同じキーの連打は左右に負荷分散が可能

たとえば「××××」というマーカがあったとしたら、「×↓×↓」と交互に入力することで片方の指への負荷を下げることが出来ます。しかし、交互に入力しているつもりがタイミングがずれてコンボが切れる事があったり、そもそも大体の曲は親指で対処可能な範囲内の譜面なので最初のうちから習得する必要はありません。

・その他

タイミング判定で SAFE 以下[*14] を取るとコンボが途切れてしまうので注意しましょう。また、個人差はありますがボタン音を切ってリズムを聴き取りやすくするのも一つの手です。

成績について

肝心の成績については総ノート数とタイミング判定で COOL か FINE だった数の割合(以下 C/F 率)で決定し、悪い順に MISS × TAKE(プレイ中に終了)→ CHEAP(C/F 率 84%以下)→ STANDARD(C/F 率 85%)→ GREAT(C/F 率 95%)→ EXCELLENT(C/F 率 97%)→ PERFECT(C/F 率 100%)というランク分けになっています。

また、スコアは成績評価には関係ないのですが、ハイスコアを目指すためにはコンボ数や、CHANCE　TIME(マーカが虹色の部分)でのミスを限りなく無くす（言うならばその間だけでもコンボがつながるようにする）事が重要です。

これらのことを覚えておけば多くの人は EASY を一週する頃には全曲がプレイ出来るようになっているはず[*15] なので、その後は次の難易度を一週ずつしていけばすべての難易度が解放されると思います[*16]。ちなみに、EXTREME 譜面の入門には難易度★ 6 の Packaged や★ 7 の Yellow、ロミオとシンデレラあたりが良いでしょう。

---

*12 この場合三回

*13 →,□,→のノートがあったら右矢印キーを押しっぱなしで、○, □, ○と入力すれば処理が可能。これをさらに応用すると EASY 譜面でイントロから右矢印押しっぱなしで○キーだけ操作できるため、PERFECT トライアル時のミスを減らすことが出来る。

*14 ボタンを押したときのタイミング評価は悪い順に WORST → SAD → SAFE → FINE → COOL となっている。

*15 最初は 4 曲しか見えないが、出てきている曲をクリアしていけばどんどん解放されていく。

*16EASY、NORMAL、HARD を各一周すればアイテムや衣装はほとんど解放されます。が、後ほど説明しますが解放されただけでは使えません。

楽曲ごとの攻略ポイント

紙面の都合上、この記事では EASY や NORMAL、HARD の攻略については割愛し、難易度が高い EXTREME 譜面から筆者が独断と偏見で選んだ難易度が高いと思われる楽曲の攻略ポイント[*17]を紹介したいと思います。

magnet

イントロ直後からリズムの掴みにくい×キー連打があり、一定速度で連打しないと途中からずれて大量の SAFE とか SAD が発生します。その後の譜面も押すキーがコロコロ変わり、次のマーカの予測が難しいところが多いので慎重に進めましょう。

愛言葉

イントロ後の「聴いてくれたあなた方に　感謝、感謝。」の後にある「このご恩を一生で忘れない～」の部分で○→×→□→△の繰り返しが続きます。押すタイミングはベースやドラムの音がもっとも参考になると思われます。その後は普通に進めれば攻略できます。

Just Be Friends

随所において、キーをぐるっと回ったりするような譜面や二連打が頻出しますが、前者はマーカをしっかり確認し、後者は歌詞の通りに連打すれば問題なく攻略できます。ただ、上の画像のようにチャンスタイム直前の「あの日の君に逢いに行くよ」の部分の○と□の連打部分の最後だけが違った連打方法になりますが、ここは○と←キーを交互にタイミング良く押せば処理できます。

ぽっぴっぽー

初見は難易度が高いですが、実は Just Be Friends と同じで歌詞とほとんど同じタイミングで連打をすればクリア可能です。ただ、三連打が多発するので苦手な人(筆者とか)は注意が必要です。

---

*17 クリアランクで EXCELLENT を取るヒントがメインですが、一部楽曲は筆者も攻略できていないのでポイントだけを説明することもあります

クローバ・クラブ、VOiCE

間奏に「わけのわからないもの[*18]」がびっしりとあります。そのため、最初から出来るだけエナジーゲージを貯めて、該当箇所は譜面を見て何とかした方が切り抜けられる確率は高いです。

恋色病棟

基本的に連打譜面なので、ズレないように注意すれば問題はありません。ただ、イントロの「そんな微熱で～」の部分の△連打や、「乱れる呼吸に　加速する心拍数が」の部分にある○長押しとその後の○連打のタイミング、上の画像の部分にあたる「私はただの薬箱じゃないわ」の○連打の後の「あんなに看病してあげたじゃない」の始まりにある△の三連打は注意が必要です。

こっち向いてBaby

イントロから派手な譜面ですがだいたいこのテンポが終わりまで続きます。イントロ部分には

*18 マーカ配置の認識が著しく困難な譜面のこと

微妙に曲のリズムとはずれたような「△↑」のマーカがあるのでタイミングに注意しましょう。そこを抜けるとすぐに連続した「○と→」の混合マーカがありますが、先ほど説明した連打方法を使えば切り抜けられます。その後も一部「↓　↓×」のようなマーカが出現しますが[*19]、同じように処理が出来ます。その後に「デレデレしないでよ」のあとの間奏でイントロと同じ連続した「←と□」の混合マーカは、最後のマーカだけが「○」なので勢い余って連打しすぎないように注意しましょう。

なお、画像のようなかなり短い長押しのマーカが複数回出現しますが、これは説明が難しいので SAFE や SAD、WORST が出たら次回出現時に気持ち長めに押すなどの試行錯誤をしてタイミングをつかんでください。個人的にはこの曲の中では先ほどの画像にある短い長押しのノートが一番難易度は高いと思われます。

サイハテ

初見殺しな上に譜面読解力が試される譜面です。上の画像はイントロ入って最初の部分ですが、見ての通り出だしから理解しにくいマーカが連続で来るので、イントロが始まったらすぐに上矢印を押しっぱなしにして[*20] 焦らず処理しましょう。序盤を抜けたら出来る限りコンボをつないでエナジーゲージをためて第二波に備えましょう。これがクリア出来る頃にはたぶん大抵の EX 譜面はクリア済みになっているはずです。

*19 一つ目の「↓」と二つ目の「↓」には若干の間があります。

*20 先の攻略方法の応用例

初音ミクの激唱

後述する前作では初音ミクの消失がボス曲扱いでしたが、今作で一番難易度が高いのはこの曲です。EXTREME の譜面をだいたいクリアしているとこの曲の NORMAL/HARD は普通に攻略できると思われますが、EXTREME は DIVA　2nd の長押しや同時押しが圧倒的に**少ない**譜面で、誰が作ったのか問い詰めたいほど連打ゲーになっています[*21]。ちなみに、序盤の連打は 12 分ですが、その後 16 分へ変化するという二段構えで我々の右親指を殺しにかかっているので、先ほど説明した方法で指への負荷を分散させる必要があります。

ちなみに後ほど詳しく説明しますが、この楽曲はクリアしなくても後述する衣装やアイテムのコレクションには全く影響がありませんので、安心してプレイしてください[*22]。

キャラクターや衣装、アイテムについて

ちなみに、PV に出演するキャラクターを変更したりキャラクターの衣装を変更することも可能（モジュールコンバートと呼ばれている）ですが、DLC[*23] 以外のすべての衣装を入手するには基本的に[*24]EXTREME 以外の楽曲を全難易度クリアする必要があります。この時注意するべきなのはゲームクリアの時点では「衣装・アイテムを買う権利をやる」という状態なのでゲーム内のストアで衣装・アイテムを購入する必要が生じます。しかし、ゲーム内のストアで衣装やアイテムを購入するには楽曲クリア時の成績に応じて与えられたポイント[*25] を使って購入する必要があります。たとえば、衣装は大体 1 万～ 5 万ポイント[*26] くらいの設定になっていて、全アイテム・全衣装を購入すると約 180 万ポイントほど必要なので大変なやりこみが必要となります。

---

*21 前述の「わけがわからないもの」とは違って認識できるがどうしようもない

*22 やり込み要素である功績とか楽曲クリア率には影響がありますが。というか、筆者も越えられない壁がそこには存在するかのように未だに EXTREME だけは攻略することが出来ません。

*23 ダウンロードコンテンツのこと。後述する Playstation　Store でゲーム発売後に追加された曲や衣装を購入することが出来る（原稿執筆時点では衣装一着が 300 円）

*24 後述する無印からのデータ引き継ぎを行うと EASY は回らなくても良くなることがあります。

*25EXTREME 譜面を EXCELLENT 評価でクリアすると 1.2 万ポイントくらい

*26 水着のモジュールは 3 ～ 5 万ポイントくらいなので普通は 2 万以下で買えます

前作からの引き継ぎ

先ほど偉く大量のポイントが必要と書きましたが、実は前作である「初音ミク -Project DIVA-」をプレイしていた方ならそのデータを**一度だけ**引き継ぐ事が出来る上に、引き継ぎでしかもらえないアイテム等がついてくる大変お得なシステムが実装されています。もし前作をプレイしていて、全モジュールを解放していた場合には実に 59 万ポイントが節約できる[*27] のでおすすめです。

もしプレイしていない方でも、2nd と同じ曲が入っていたり、逆に 2nd には入っていない良い曲が前作には結構収録されています。さらに、同時も長押しも存在しない譜面がほとんどなので余裕でクリアすることが出来るでしょう[*28]。ちなみに、前作は 6 月にお買い得版が発売されて安く入手できるのでプレイしてみることをおすすめします。

さらに、前作では PS3 向けの DLC として PS3 上で動く「初音ミク -Project DIVA- ドリーミーシアター」(以下 DIVA DT)が提供されていて、2nd から前作に UMD を入れ替えるのが面倒な時[*29] や PSP のポリゴンに飽き飽きしてきたとき[*30][*31] に大変威力を発揮します。

おわりに

もしかしたらこの記事だけを見ると何のこっちゃわからない方もいらっしゃると思いますが、興味があれば Playstation Store で体験版が公開されているのでプレイしてみることをおすすめいたします[*32]。また、「PSP 本体や PS3 本体を買うほどでも…」というような方は収録曲やシステムは若干異なりますがアーケード版の「初音ミク -Project DIVA- Arcade[*33]」をプレイしてみてください[*34]。アーケード版では PS3 と同じく HD 画質でプレイすることが可能です。

また、この曲がいまいちよくわからない等の感想は WORD 編集部のサイト[*35] からお願いします。次回の記事ではアーケード版について書く予定なので、その参考にさせていただきます。

なお、この記事を書くに当たって、衣装に必要なポイント等が掲載されている「初音ミク -Project DIVA- wiki[*36]」を参考にさせていただきました。記事に載ってないような細かい情報も掲載されているので、困ったときに参考にすると良いと思われます。また、本記事に掲載したすべての画像の著作権は SEGA 様と Crypton Future Media, Inc.様に帰属します。

---

*27 大体難易度 NORMAL で GREAT、難易度 HARD で GREAT を全曲で取っていれば EASY はプレイしなくても(一部を除いた)衣装が解放されています。

*28 特に前作のボス曲だった消失は今作の激唱 HARD をクリア出来る人ならそれほど苦労せずクリア可能です。

*29DIVA 2nd は Playstation Store でダウンロード版が購入できます。

*30 前作で水着衣装を解放しておけば DIVA DT と 2nd にも引き継げるので一粒で三度おいしい

*31 DIVA DT は PSP では処理速度の都合上出来なかった物理演算ができるので何かが揺れて見えます。詳しくは DIVA DT を買って水着のモジュールでリザルト画面を見てみよう！

*32 何らかの細工を PSP に施している方は体験版をプレイするにはファーム更新が必要になるので注意が必要です。もっとも、そのような人は自力で何とか出来ると思いますが。

*33 公式サイトは http://miku.sega.jp/arcade/ です。

*34 原稿執筆の時点で大学付近でおいてある店はつくばセンター近くのデイズタウン地下にある「アミューズメントジャムジャムつくば店」か LALA ガーデンつくば付近にある「ピンクパンサー つくば店」です。そのほかは公式サイトからどうぞ。

*35 http://www.word-ac.net/

*36 http://www19.atwiki.jp/mikudiva/

面倒な例外処理は他人(スレッド)に押しつけよう *0x*

文 編集部 Flast

9 月なんて来なかった。9 月なんて来なかった……

おはようございます。こんにちは。こんばんは。コンパイル時ソートとか誰得大好きな Flast です。もう早いことで 2010 も半ばです。0xA[*1] ですね。唐突ですが今年から来年にかけてのトレンドと言えばやはり C++0x でしょう。

すでに FCD[*2] は承認され、来年標準化が完了すると言われています。また、いくつかの主要なコンパイラは一部の機能を実装している動きもあります。

ということで今回はこの C++0x の一部をご紹介しましょう。また、今回紹介する内容は GCC4.4 以降および VisualStudio2010 で対応していることを確認しています。それ以外では動かない可能性が非常に高いので予めご了承下さい。まず無いでしょうが、今後変更される可能性もあるのでそこら辺も考慮して読んで下さい。

C++0xでの主要な機能

今回詳しく解説はしませんが、各所で取り上げられている主要な機能をいくつか挙げます。気になる人は Google 先生に聞けば大概載っています。また、情報は古いですが Wikipedia のページ[*3] も参考になるでしょう。

R-value reference(右辺値参照)
Variadic templates(可変長テンプレート引数)
Lambda expression(ラムダ式)
Type inference(型推論)
Multi threading(スレッドのサポート)
Initializer lists(初期化リスト)

R-value reference はこれまでの std::auto_ptr の問題を解決するのに役立つし、Variadic templates はより汎用性の高い template が書けるようになります。Lambda expression に至っては std::for_each 等のアルゴリズムとの組み合わせが最強だと思います。地味に便利なのが Type inference でしょう。無駄に長い宣言文を書く必要がなくなるし、typo を防ぐことにもなります。

また、STL も非常に強化されています。特に Boost[*4] から取り込まれた部分も多く、既存の Boost で書かれたコードも少ないコストで書き換えることができます。STL では Boost に加えて上記の新機能にも対応しているのでより強力です。

---

*1 もちろん 0x が 16 進であることは信者のみなさんならお解りだろう。

*2Final Committee Draft の略。ISO の標準化作業のマイルストーンの一つ。これ以後仕様が大きく変更されることはまずないでしょう。

*3http://ja.wikipedia.org/wiki/C++0x

*4STL になりきれなかったライブラリ達。J リーグで言うと J1 が STL で J2 が Boost ……？

ところで、今回紹介する機能は例外に関する機能ですが、みなさんは例外を使っているでしょうか。どうも C++をあまり理解できていないチームでのコーディング規約には、例外を使用してはならないという規約も存在してしまうぐらい好まれていない様です。そんな例外ですが、とりあえず利点や使い方を次節に挙げたいと思います。

C++の例外

例外を使うと煩雑なエラーコードや大量の if 文やできれば使いたくない goto 文などを排除した上でエラーを上位関数へ通知することができます。また、スタックフレームに存在するスコープの外れたオブジェクトは正しくデストラクタが呼ばれます。また、戻り値をエラーコードにしたくない場合でもわざわざ errno のような変数を用意したりする必要がないのも、例外を使う利点です。

例外の単純な使い方は次のようになります。この場合、文字列へのポインタを投げています。

```
try
{
    throw "throw exception";
    std::cout << "unreachable" << std::endl;
}
catch ( const char *str )
{
    std::cout << str << std::endl;
}
```

まぁ使い方は分かっているものとしてこれからは進めます。

ところで受け取った例外を一時的に保存したり、別のスレッドへ送り付ける方法はどうすればいいのでしょうか。イベントドリブン型のプログラムの場合、イベントを処理するスレッドを用意していることが多いでしょう。そのイベントの一つとして例外を処理したい場合があります。そのような場合どうすればいいのでしょうか。

多分単純に思いついてスマートな方法は以下のようなシステムを自前で作ることでしょう。

```
#include <memory>

typedef struct _exception_base
{
    friend void _rethrow_exception( _exception_base * );

    virtual ~_exception_base( void ) {}

protected:
```

```
    virtual void
    rethrow( void ) const = 0;
} *_exception_ptr;

template < typename _Exception >
struct _exception_impl
  : public _exception_base
{
    template < typename _Ty >
    friend _exception_ptr _copy_exception( const _Ty & );

private:
    _Exception _expr;

    _exception_impl( const _Exception &_x )
      : _expr( _x ) {}

    virtual void rethrow( void ) const
    { throw this->_expr; }
};

template < typename _Exception >
_exception_ptr _copy_exception( const _Exception &_x )
{ return new _exception_impl< _Exception >( _x ); }

void _rethrow_exception( _exception_ptr _x )
{
    std::auto_ptr< _exception_base > ptr( _x );
    _x->rethrow();
}
```

_exception_ptr が一時的に例外を格納するためのポインタ型です。これは以下のコードのように使用します。

```
#include <iostream>

int main( void )
{
    _exception_ptr expr = NULL;
    try
    { throw "hoge"; }
    catch ( const char *ptr )
    { expr = _copy_exception( ptr ); }
```

```
    try
    { _rethrow_exception( expr ); }
    catch ( const char *ptr )
    { std::cout << ptr << std::endl; }
}
```

1 回目の try-catch で投げられた例外は一旦 expr に格納され、2 回目の try-catch で再度投げられます。この様に例外を別の try-catch で再度投げるだけでなく、別のスレッドへ渡すことも可能です。

しかし、この実装では問題がいくつか出てきます。まず、例外が rethrow されなかった場合 delete されない為、メモリリークを起こすことになります。次に exception-declaration[*1] であらゆる例外を受け取るようにした場合、現在の例外オブジェクトを取得する方法が無い為、このシステムを適用することができません。

C++0xでの解決方法

例外は exception-declaration で受け取る例外の型を指定することでその型の例外ハンドラが作られ、例外が投げられたときにはマッチする型のハンドラが呼ばれることになっています。これはどのような型を投げたとしても当てはまることで、std::exception を継承している必要もなく、もちろん int 型やポインタですら投げることが可能です。

しかし exception-declaration であらゆる例外を受け取るようにした場合、現行 C++ではプログラマ側は投げられた型を知ることができません。しかしコンパイラ側は動的にどの型のハンドラを起動するか決定するために、バイナリ上に型を保存するシステムを作らないといけないので、当然投げられた例外の型を知ることができるはずです。

なのでこれを利用できるようになればいいわけです。そして C++0x ではそのための関数を用意しています。先程のコードを C++0x で書き直すと以下のようになります。

```
#include <iostream>
#include <exception>

int main( void )
{
    std::exception_ptr expr = NULL;
    try
    { throw "hoge"; }
    catch ( ... )
    { expr = std::current_exception(); }
```

*1catch（ /*ここのこと*/ ）

```
    try
    { std::rethrow_exception( expr ); }
    catch ( const char *ptr )
    { std::cout << ptr << std::endl; }
}
```

exception ヘッダが必要になるのでこれをインクルードします。そうすると std namespace の exception_ptr 型と current_exception()関数、rethrow_exception()関数、上記のコードでは使用しませんでしたが copy_exception()関数が使えるようになります。

スレッドとの併用

もちろんスレッドと一緒に使用すれば例外処理を別のスレッドへ丸投げすることができます。

このコードは GCC4.6 以降が必要ですが、C++0x の機能を大いに活用すると以下の様なコードになります。

```
#include <iostream>
#include <exception>
#include <thread>
#include <mutex>
#include <condition_variable>
using namespace std;

auto main( void ) -> int
{
    mutex                _sync_mutex;
    condition_variable _sync_cond;
    exception_ptr       expr = nullptr;

    thread th( [&]
    {
        unique_lock< mutex > lk( _sync_mutex );
        _sync_cond.wait( lk, [&expr] { return expr != nullptr; } );

        try
        { rethrow_exception( expr ); }
        catch ( const exception &e )
        { cout << e.what() << endl; }
        expr = nullptr;
    } );

    try
    { throw exception(); }
    catch ( ... )
```

```
    {
        unique_lock< mutex > lk( _sync_mutex );
        expr = current_exception();
        _sync_cond.notify_one();
    }

    th.join();
    return 0;
}
```

th オブジェクトが例外を処理するスレッドになります。このスレッドでは expr に何か例外が格納されるまで wait し続けます。通常 condition variable を使った wait では while ループが使われたり[*1] しますが、std::condition_variable::wait()のオーバーロードされた関数は単体でこれを実現します。

まとめ

マルチコア化が進むことでマルチスレッドプログラミングが一般的になりつつあります。ですが、C++やその祖先の C ではマルチスレッドで使われることを前提で考えていなかったため、言語仕様のレベルではうまくいかない部分も多くありました。しかし今回の C++0x ではマルチスレッドプログラミングを前提とした改良がなされているので是非使ってみてもらいたいと思います。

今回は C++0x の話ということでしたが、メジャーな部分は適当に検索すれば出てくるので、あえてマイナーな例外処理を扱いました。他にもマイナーながら地味に使えるものもあるので是非探してみてください。

---

*1POSIX thread だとこんな感じ

```
pthread_mutex_lock( &mutex );
while( /* 条件 */ )
{ pthread_cond_wait( &cond, &mutex ); }
pthread_mutex_unlock( &mutex );
```

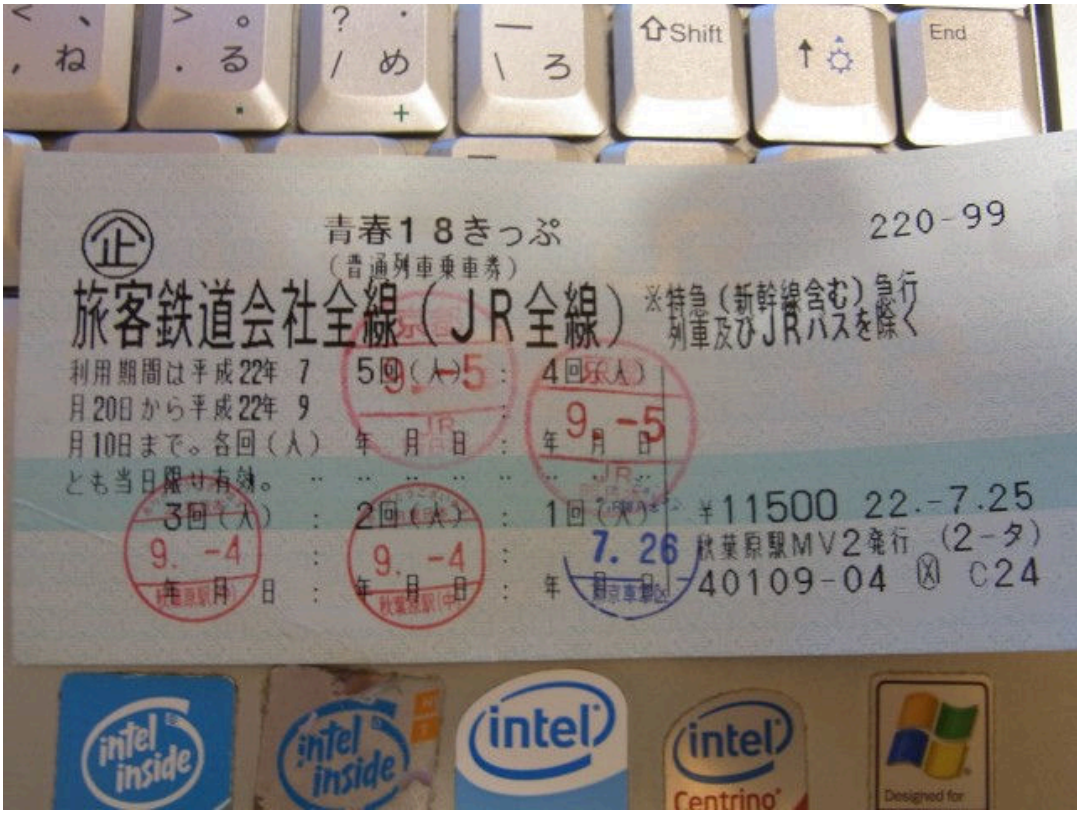
青春18きっぷで青春しよう！

文 編集部 ふぁい

■あいさつ

おはこんばんちは[*1]。皆さん青春してますか？ してる人もしてない人も、この記事を読んで一緒に~~修行~~青春しましょう。この記事では、タイトルの通り青春 18 きっぷの布教をします。

■青春 18 きっぷって何？



青春 18 きっぷの写真

青春 18 きっぷ(以下、18 きっぷと記載)とは、おおまかに言えば **JR の普通電車が 1 日乗り放題になるお得なきっぷ**です。ここで言う「普通電車」とは、「乗車券の他に急行券や特急券を買わなくてもいい電車」の事です。つまり快速とか新快速とか特別快速とか、そういうのも 18 きっぷで乗れます。また、リゾート用の電車などで「快速だけど全席指定席だから指定席料金は取るよ」という電車が存在しますが、これらはあくまでも「快速電車」なので、指定席券だけ買えば 18 きっぷで乗車できます。例外として、特急電車だけ走っていて普通電車が 1 日に 1 本も無いという理由で、18 きっぷで特急に乗れる区間があります[*2]。

使用回数は 1 枚の 18 きっぷで 5 回分で、1 回あたり 1 人 1 日乗り放題になります。1 人で 5 日

*1 おはよう＋こんばんは＋こんにちは

*2 津軽海峡線の蟹田～木古内と、石勝線の新夕張～新得

使ったり 5 人で 1 日つかったり、5 回分の使い道は自由です。複数人で使う場合は、きっぷが 1 枚しかないので常に一緒に行動をする必要があります。

前ページの写真を例に説明しましょう。1 回目が 7 月 26 日に無人駅から乗って車掌にスタンプを押してもらったもの、2,3 回目は 9 月 4 日に秋葉原駅で押してもらったもの、4,5 回目が 9 月 5 日に京都駅で押してもらったものです。つまり、7 月 26 日に一人で使って、9 月 4 日～ 5 日に 2 人で使った事になります。

発売期間は毎年春・夏・冬の 3 シーズンです。それぞれ発売日と利用可能期間が決まっているので、インターネットなどで調べてください。

気になる値段は 1 枚 11500 円。つまり 1 回あたり **2300 円**で 1 日乗り放題になるので、長い距離を移動すれば移動するほどお得になります。

利用の際は自動改札ではなく、駅員さん(無人駅では車掌や運転士など)に見せましょう。日付と乗車駅がかかれたスタンプを押して貰えばその日 1 日乗り放題です。

以上でおおまかな説明は終わりです。少し説明が長くなってしまったので、以下に例をいくつか示します。

○:中央線の青梅特快

快速だろうが青梅特快だろうが、乗車券だけで乗れるので 18 きっぷで乗れます。

×:つくばエクスプレス

つくばエクスプレスは JR ではないので 18 きっぷでは乗れません。

△:北陸線・ほくほく線の越後湯沢～美佐島

越後湯沢から途中の六日町までは JR ですが、そこから美佐島までは JR ではなく北越急行の路線なので、この区間分の乗車料金だけ取られます。

×:新幹線

新幹線は新幹線特急券が必要になります。つまり特急扱いなので 18 きっぷでは乗れません。通常どおり、新幹線特急券と乗車券を買う必要があります。

○:快速「ムーンライトながら」

ムーンライトながらは全席指定の夜行快速電車です。指定席券を買えば快速電車なので 18 きっぷで乗れます。但し、日をまたいで運行するので 18 きっぷを 2 回分つかう事になります[*3]。

○:特急白鳥を蟹田から木古内まで

青函トンネルを通る電車はこの特急電車しか設定されていません。そのため、本当に特急を使わざるを得ない蟹田～木古内は特例として 18 きっぷのみで乗車できます。

×:特急白鳥を蟹田から函館まで

木古内～函館に関しては普通電車が設定されているので 18 きっぷだけでは利用できません。特例として 18 きっぷだけで乗れるのは、あくまで 1 つ上の例にあげた区間だけなので、その区間を逸脱すると乗車した全区間(この場合は蟹田～函館)の乗車券と特急券を買う必要があります。

*3 但し、大垣行きのムーンライトながらは小田原駅で日付をまたぐので、小田原までの運賃が 2300 円以下のときは 18 切符を 2 日分つかうよりも小田原までの切符を買った方が安上がりになります。

×:博多南線

290 円で新幹線に乗れるという神様のような路線です。法的には「在来線で特急しか走ってない路線」という扱いですが、JR 西日本が~~ガめつい~~ 18 きっぷだけでの利用を認めていないので 18 きっぷでは乗れません。というか 300 円くらい払おうぜ。

×:特急オホーツクで上川から遠軽まで

この区間を移動できる普通電車は **1 日に 1 本**しかなく、特急電車以外での移動は困難を極めますが、とにかく 1 日に 1 本でも普通電車がある以上は 18 きっぷで特急に乗ることは出来ません。

×:JR 貨物のコンテナ

そもそも人を運ぶ目的では無い電車には乗れません。

■ 18 きっぷ活用例

18 きっぷは JR 貨物以外の JR の全路線で使えるので、JR さえ走っていればどこにでも行けます。ここでは 18 きっぷを~~使い倒しに~~駆使してつくばから大阪まで行くルートを例に挙げてみます。筆者のおすすめルートでもあるので、関西方面に行く方で時間に余裕があれば是非試してみてください。

◆つくば　6:12 発
↓　　つくばエクスプレス　区間快速 秋葉原行き 1150 円
秋葉原　7:05 着

つくば脱出には、つくばエクスプレスと高速バスの他に土浦まで出て常磐線というルートがありますが、土浦は遠いし高速バスはあまり時間が正確では無いのでつくばエクスプレスを使うのが無難だと思います。

◆秋葉原
↓　　山手線(外回り)か京浜東北線(南行)　18 きっぷ使用
東京

この区間はすぐに電車が来るので時間は特に書きません。先に到着した方に乗りましょう。

◆東京　　7:24 発
↓　　東海道線　普通 伊東行き　18 きっぷ使用
熱海　　9:20 着

この電車は、**「普通電車だけど車輌は特急車輌」**という大変お得な電車です。車輌は特急車輌でも普通電車には変わりはないので 18 きっぷで利用できます。全車クロスシート・リクライニング装備など、他の電車よりもサービスの良い電車なので熱海まで快適に移動できます。つくばを朝早くに出れば十分余裕で乗れる時間なので、この電車の利用をオススメします。余談ながら筆者はこの電車に乗ろうと大学会館 5:17 発の高速バスに乗り込みましたが、渋滞＋ **18 きっぷを買おうとしたら券売機が壊れて 2 万円が出てこなくなった**というコンボを食らって間に合いませんでした。18 きっぷは事前に購入しましょう＆高速バスは渋滞に巻き込まれるの

で TX を使いましょう。

◆熱海　　9:37 発
　↓　　東海道線　普通 島田行き　18 きっぷ使用
　興津　　10:34 着

　ここから JR 東海のテリトリーに侵入します。両数が減るので、並ばないと座席の確保ができません。また、全車ロングシートになります。北側の席(進行方向右側)に座ると日差しが暑くない&たまに海が見えて景色が綺麗なので、北側の席がおすすめです。

◆興津　　10:42 発
　↓　　東海道線　普通 浜松行き　18 きっぷ使用
　浜松　　12:14 着

　先ほどの島田行きの電車で終点の島田まで行っても、結局この興津始発の電車に乗るハメになります。座席が空いている興津駅で乗換えてしまうのがベストです。
　熱海～浜松はほとんどの車輌がロングシートのみの車輌です。頑張って耐えましょう。

◆浜松　　12:29 発
　↓　　東海道線　普通 豊橋行き　18 きっぷ使用
　豊橋　　12:55 着
　浜松到着はちょうどお昼ごろですが、お昼ご飯なんて食べなくても**何の問題も無い**ので、食べないものとしてハナシをすすめていきます。この区間は人が多いのに何故か両数の少ない電車が走っている事が多々あります。座席の確保が難しいかもしれませんが、30 分程度なので立っていても問題は無いでしょう。

◆豊橋　　13:03 発
　↓　　東海道線　快速 大垣行き　18 きっぷ使用
　大垣　　14:31 着

　豊橋～岐阜は JR 東海と名古屋鉄道が客の奪い合いをする**バトル区間**なので、JR 東海が本気を出す区間です。
　豊橋までと比べて、両数が長くなり、座席もロングシートから転換クロスシートにレベルアップして、さらに快速・新快速・特別快速といった速達列車が多数走っています。大府～金山に、東海道線の横にもう 2 本くらい線路がひけそうな空き地が延々とありますが、これは国鉄時代に「貨物専用線を作ろうとして整備して線路もひいたけどやっぱやめた。線路はとっぱらった。」という歴史をもつ壮大な~~税金の無駄遣い~~夢の跡です。延々と続く空き地をみながらかつての野望に思いをはせてみてはいかがでしょうか。ちなみに筆者は思いをはせたことはありません。

◆大垣　14:40 発
　↓　　東海道線　普通 米原行き　18 きっぷ使用
　米原　15:17 着

電車の両数がぐっと減ります。大垣駅での座席争奪戦はかなり熾烈なものになります。古い車輌(117 系電車)だと吊革が無かったり手すりが少なかったりするので、立って移動するには辛い区間となります。

◆米原　15:19 発
↓　　東海道線　新快速 播州赤穂行き　18 きっぷ使用
大阪　16:43 着

米原から西は JR 西日本のテリトリーです。阪急電車･阪神電車･京阪電車･山陽電車などのライバルが多く出現する区間なので、JR 西日本が本気を出す区間です。新快速はひたすら速いので快適に移動できます。
京都には 16:12、神戸には 17:10、姫路には 17:47 に到着します。＼はやい！／

というわけで、無事に大阪に到着しました。気になる運賃は TX の 1150 円と 18 切符の 2300 円で**合計 3450 円也**。＼やすい！／
TX で回数券をつかったり、高速バスをつかったりすればさらに安くなります。
この例を見て分かるように、普通電車しか乗れないので非常に時間がかかります。もはや修行の域です。こういう経験が出来るのは時間のある学生の特権だと思います。ぜひ皆さんも 18 きっぷで鉄道旅行してみて下さいね。

■便利なサイト
18 きっぷを使って移動するときに便利な乗換え検索サイトの紹介です。

乗換え案内｜ジョルダン
http://v203.jorudan.co.jp/norikae/cgi/nori.cgi?pg=1

この乗換え検索サイトは、他の乗換え検索サイトと違って青春 18 きっぷだけで移動できる経路を導き出してくれる「青春 18 きっぷ検索」という機能があります。
普通電車を探し出すために時刻表とにらめっこするよりも、はるかにラクなので利用してみてください。

■ Q&A
Q,なんで熱海～豊橋は各駅停車しか無いの？所要時間 3 時間とか辛いよ＞＜
A,JR 東海「新幹線をご利用下さい。」
Q,熱海～豊橋とか大垣～米原とか、何でこんなに両数少ないの？座れないしもっと増やせよ。
A,JR 東海「新幹線をご利用下さい。」

C++テンプレートの計算力はチューリング完全程度の能力

文 編集部 ranha

表題にある通り、今回は C++のテンプレートの破壊力がチューインガムが耳元で破裂した時の音程度である事を調べてみます。

使うモノは 3 つ

・ラブライブの CD

・Brainf*ck がチューリング完全であるという事

・Variadic Template が使える C++のコンパイラ

手順は次の通り

1:ラブライブの CD を聞く

2:Brainf*ck がチューリング完全であるという事を納得する

3:Brainf*ck の処理系を C++テンプレートで書く

1.1 ラブライブの CD を聞く

2000 円を準備して CD を買いましょう!! それと、CD 情報のページ[*1] にアクセスするです。

2.Brainf*ck がチューリング完全であるという事を納得する

Brainf*ck という言語については、皆さん名前を聞いた事があると思います。そして Brainf*ck で遊んだ事がある方も多いと思います。遊んだ事が無い方は今すぐインタプリタを書いてみてください。すぐ書けます。

Brainf*ck で使用可能な命令は 8 つ。下に、それぞれ C で書いた場合と並べて書きます。

'>' = ptr++; ポインタをインクリメントする。

'<' = ptr--; ポインタをデクリメントする。

'+' = (*ptr)++; ポインタが指す値をインクリメントする。

'-' = (*ptr)--; ポインタが指す値をデクリメントする。

'.' = putchar(*ptr); ポインタが指す値を出力する。

',' = *ptr = getchar(); 1 バイト読み込んで代入する。

'[' = while(*ptr){ ポインタが指す値が 0 なら対応する]の直後にジャンプする。

']' = } ポインタが指す値が 0 でないなら、対応する[にジャンプする。

[[]] はさしずめ、while(*ptr){while(*ptr){}}という事に成ります。意味は在りませんが。

Brainf*ck はこの 8 つの命令しか備えていないにも関わらず、チューリング完全であるという事は良く知られています。[*2]

厳密には Brainf*ck で使える配列のサイズが無限でなければならないのですが、それはともかくとして。"チューリング完全である"というのはなんというか、非常にそれらしい感じではあります。

さて! Brainf*ck がチューリング完全である事を証明するには、チューリングマシンから Brainf*ck のプログラムに変換する術が見つかれば良い、という事に成ります。逆に全ての Brainf*ck

*1http://gs.dengeki.com/lovelive/news/?p=108

*2http://www.muppetlabs.com/~breadbox/bf/

のプログラムをチューリングマシンに変換する術を見付けたとしても、それはチューリングマシンが Brainf*ck 完全であると言うに過ぎないことに注意してください。

チューリングマシンの説明等はしませんが、次のようにして Brainf*ck のプログラムへ変換する事が出来ます。

1:チューリングマシンの状態とシンボルに番号付けをする(特に終了状態を x とおく)。
2:現在の状態とヘッドの位置を保存するメモリ位置を用意する。
3:メモリをテープであるように見立てる。
4:現在の状態が x であれば終了する。
5:ヘッドの位置にある値(つまりシンボル)を適切な位置にストアする。
6:現在の状態とヘッドの指すシンボルを読み取れたので遷移関数を順番に適用していく。
7:3 の過程で次の状態、次のヘッドの位置、書き込むべきシンボルを得るのでこれをヘッドの位置に書き込む。
8:1 に戻る。

とすれば良いです。これはチューリングマシンの挙動そのものに過ぎないので別段難しくはありませんが、Brainf*ck の命令レベルで逐一書いていては大変です。ですから、もう少し高級な命令を用意してその命令からプリミティブな命令への変換を行った方が良いでしょう。

このサイト[*3] が参考になると思います。兎に角 Brainf*ck はチューリング完全であるということを納得してください!!!!!! はい納得出来た!! 今納得した!!

1.2 アニメーションショートフィルムを観る

恐らく CD と DVD の 2 枚組に成っていると思います。そのうち、DVD の方は 1 曲目の"僕らの LIVE 君との LIFE"のアニメーションショートフィルムです。アニメーションショートフィルムって言い方はちょっとって感じですよ……。アニメって何を言ってるんですか!?普通に本人出演の PV なんじゃないんですか!?

こんなマトモそうな学校なのに生徒数が足りなくて大変や……というのは、やっぱり設定がちょっとなぁとか思いつつも他の生徒が全く出ていなくて仕方ないという気持ちに成ります。各人がアップになるシーンでは、その人の声が一際良く聞こえるように編集してくれても良かったんじゃないかと思います。中の人が現時点で公開されていない分、中の人嗅ぎ分け力の余り高く無い私はそう思います。切に!! いや……これはお預けなんや……私は豚になりたい。

ダンスは活力があって、可愛いし可愛いしカワイイ。歌詞カード、ジャケットの絵、PV それぞれで印象が違うのも嬉しいです。というかですね! これ歌詞カードに載っている自己紹介文を見てから PV を観ると、見た目から受ける印象とのギャップではわわ〜って成ります。愚民になるしかない……。

3.Brainf*ck の処理系を C++テンプレートで書く

本記事の目標は、C++テンプレートがチューリング完全である事を言うこと、です。幾つか方法はあるのですが、今回は **C++のテンプレートで Brainf*ck の処理系を書く**事によってそれを証明したいと思います。

*3http://www.iwriteiam.nl/Ha_bf_Turing.html

3.1 そもそも C++のテンプレートって何なんですか?

ご安心を!! C++のテンプレートは何も難しい話ではありません。C++のテンプレートはコンパイル時間を増やして、その間にラブライブの CD を聞く為の技術です!!

取りあえず例を出してみたいと思います。以下のコードは、**コンパイル時**にフィボナッチ数を計算する C++のテンプレートを使ったコードに成っています。**コンパイル時**に計算出来るという事は、そもそも実行しなくても良いという事です。やった!!

```
template<int i>
struct fibonacci{
//fibonacci<1>と ficonacci<2>以外の時はこの実装が選ばれる
  static const int value = fibonacci<i-1>::value + fibonacci<i-2>::value;
};

template<>
struct fibonacci<1>{//fibonacci<1>に特化(specialize)された実装
  static const int value = 1;
};

template<>
struct fibonacci<2>{//fibonacci<2>に特化(specialize)された実装
  static const int value = 1;
};

static_assert(fibonacci<42>::value == 267914296,"＼(^o^)／");
```

ちょっと睨めっこしてみてください。何が起こるか、なんとなくで良いので分かったでしょうか? 駄目押しにもう 1 例です。

```
struct _0{};//_0 は 数 0 を表す。

//s は succ の頭文字で、succ<x>は 1+x を表す。
template<typename T>struct s{};

//_1 は 数 1 を_2 は 数 2 を_3 は 数 3 を それぞれ表します。
typedef s<_0> _1;
typedef s<_1> _2;
typedef s<_2> _3;

/*足し算の定義
0 + b = b
(1 + a) + b = 1 + (a + b)
*/
```

```
template<typename a,typename b>struct add{};

//add を、第一引数が"_0"の時と"そうでない"時とで、場合分けをして実装します。
template<typename b>
struct add<_0,b>{//0 + b =
  //b
  typedef b value;
};

template<typename a,typename b>
struct add<s<a>,b>{//(1 + a) + b =
  //1 + (a + b)
  typedef s<typename add<a,b>::value> value;
};

/*掛け算の定義
0 * b = 0
(1+a) * b = b + (a * b)
*/
template<typename a,typename b>struct mul{};

template<typename b>
struct mul<_0,b>{//0 * b =
  //0
  typedef _0 value;
};

template<typename a,typename b>
struct mul<s<a>,b>{//(1+a) * b =
  //b + (a * b)
  typedef typename add<b,typename mul<a,b>::value>::value value;
};

template<typename T>
struct is_five{ static const int value = 0; };
template<>
struct is_five<s<s<s<s<s<_0> > > > > >{
  static const int value = 1;
};
template<typename T>
struct is_six{ static const int value = 0; };
template<>
struct is_six<s<s<s<s<s<s<_0> > > > > > >{
```

```
  static const int value = 1;
};
static_assert(is_five<add<_3,_2>::value>::value,"3 + 2 != 5 ...!?");
static_assert(is_six<mul<_3,_2>::value>::value,"3 * 2 != 6 ...!?");
```

なんとなく雰囲気で読み取ってもらえると思います。別に丁寧に解説する気はありませんし丁寧に解説する時間がないんですよ!! こんな事を一々説明している暇があったら、絶賛放置中のコープスパーティーブラッドカバーリピーティッドフィアーをやります。というよりも、詳しく解説しているサイトやもっと面白い例を扱っているサイトもあると思いますから、"C++ テンプレート メタプログラミング"とか、"Boost ぺろぺろ"で、まずはググれ?

なお、本記事では誌上での読み易さを優先する為に、例えば

```
typedef typename add<b,typename mul<a,b>::value>::value value;
```

を

```
value = add<b,mul<a,b>::value>::value
```

などと、良い感じに短く見える風に書く事にします。お許しください。

それから今回は、C++0x の記事が他にも掲載されるという事ですので、**Variadic Template** を無駄に使っちゃうというスペシャル仕様に成っています。これは C++0x から入る機能で、適当に訳すと可変長テンプレートという事に成るかもしれません。つまり、任意の数のテンプレート引数を取れるようなテンプレートという事です。意味不明ですね。そう C++は意味が不明なのです。

```
template<int... is>struct hoge{};
hoge<0> a;
hoge<0,1> b;
hoge<0,1,2> c;
hoge<72,72,72,72,72,72,72,72,72> _72;
```

Variadic Template を 1 つ書くだけで幾らでもテンプレート引数を渡せるので、**長さが決まってない何か**を表現するのに、オメガ便利な予感がします……。いや予感じゃなくて使います。

3.2 Brainf*ck を世紀末実装する

任意の数のテンプレート引数を取れるというのはなんだか凄そうです。コレで何か表現出来ないかな～そうだな～。そうだ!! プログラムとメモリを表現すれば良いじゃない!! 決定です。まず次のように書きましょう。

```
template<char... ops>struct program{};//プログラム
template<int... i>struct memory{};//メモリ
```

Brainf*ck の命令 1 つは 1 文字で表現出来るので、プログラムも同様に表現出来そうです。program は文字(char)から構成されていて、memory は数値(int)から構成される。それだけですね。

次に絶対に使うであろう便利な関数を 2 つ、実装しておきましょう。1 つはメモリの指定した位置から値を読み取る関数 get_value。1 つはメモリの指定した位置に、値をストアする関数 set_value です。(でも実際には関数じゃなくて struct なんですけれども)

```
//get_value<i,memory<...>>で、memory<...>の i 番目(0-origin)の要素を取得する
//配列のアクセス風に書くと、memory<...>[i] な感じ
template<int idx,typename Memory>struct get_value;

template<int v,int... is>
struct get_value<0,memory<v,is...> >{
  static const int value = v;
};

template<int idx,int v,int... is>
struct get_value<idx,memory<v,is...> >{
  static const int value = get_value<idx-1,memory<is...>>::value;
};
```

とても簡単ですね。次のように使える筈です！
get_value<2,memory<80,83,72,91>>::value → get_value<1,memory<83,72,91>>::value →
get_value<0,memory<72,91>>::value → 72

```
//set_value<i,v,memory<...>>で、
//memory<...>の i 番目(0-origin)に値 v を代入する
//配列のアクセス風に書くと、memory<...>[i]=v な感じ
template<int idx,int v,typename memory>struct set_value;

template<int v,int v_,int... is>
struct set_value<0,v,memory<v_,is...> >{
  value = memory<v,is...>;
};

template<int idx,int v,int v_,int... is>
struct set_value<idx,v,memory<v_,is...> >{
  value = cons<v_,set_value<idx-1,v,memory<is...>>::value>::value;
};
```

謎の cons というものが現れていますが、コレはとっても簡単で、cons<72,memory<80,83,91>>
::value → memory<72,80,83,91> というように値をパラメタ列の先頭にくっ付ける働きをします。
set_value の振舞は、例えば
set_value<1,83,memory<80,86,91>>::value → cons<80, set_value<0,83,memory<86,91>>::value >::value →
cons<80, memory<83,91> >::value → memory<80,83,91>
となります。memory<80,86,91>の添字 1 の"86"を"83"にする事が出来ました。
次に中心になる関数(のように見えてやっぱり struct です)を実装します。

```
template<typename Rest,typename Eated,
```

```
        int idx,typename Memory,typename Output> struct exec;
```

とする事にしましょう。Rest には program<...>の形でプログラムが入っています。例えば、program<'[','[',...>と成っているならば、今実行するべき命令が'['であるという事に成ります。

次に Eated は、これまでに実行した命令が入っています。例えば program<'>','+','<','-'>というプログラムが在ったとして、順に実行されて'<'に来たとしましょう。exec を使って表現すると、

exec<program<'<','-'>,program<'+','>'>,idx,Memory>

となります。処理し終わった命令を Eated に左から入れて行くので、program<> → program< '>' > → program< '+','>' > というように入って行くのです。この入れ方をすると、']'に対応する'['を探す時に、Eated を左から右に見て行くだけで済むので実装が少し、楽に成ります。

残りのパラメタ idx は、現在のポインタの位置。Memory はそのまま、メモリを表します。では exec の実装をします。

```
//プログラムが終了する時には、出力と最終的なメモリを返す。
template<int idx,typename Memory,typename Output,char... Eated>
struct exec<program<>,program<Eated...>,idx,Memory,Output>{
  value = Pair<Output,Memory>;
};

//処理すべき命令が'>'だった時
template<int idx,typename Memory,typename Output,char... Rest,char... Eated>
struct exec<program<'>',Rest...>,program<Eated...>,idx,Memory,Output>{
 value = exec<program<Rest...>,program<'>',Eated...>,
                idx+1,Memory,Output>::value;
};

//処理すべき命令が'<'だった時
template<int idx,typename Memory,typename Output,char... Rest,char... Eated>
struct exec<program<'<',Rest...>,program<Eated...>,idx,Memory,Output>{
  value = exec<program<Rest...>,program<'<',Eated...>,
                 idx-1,Memory,Output>::value;
};

//処理すべき命令が'+'だった時
template<int idx,typename Memory,typename Output,char... Rest,char... Eated>
struct exec<program<'+',Rest...>,program<Eated...>,idx,Memory,Output>{
  NewMemory = set_value<idx,get_value<idx,Memory>::value+1,Memory>::value;
  value = exec<program<Rest...>,program<'+',Eated...>,
                 idx,NewMemory,Output>::value;
};
//処理すべき命令が'-'だった時
template<int idx,typename Memory,typename Output,char... Rest,char... Eated>
struct exec<program<'-',Rest...>,program<Eated...>,idx,Memory,Output>{
```

```
  NewMemory = set_value<idx,get_value<idx,Memory>::value-1,Memory>::value;
  value = exec<program<Rest...>,program<'-',Eated...>,
                idx,NewMemory,Output>::value;
};

//処理すべき命令が'.'だった時
template<int idx,typename Memory,char... Rest,char... str,char... Eated>
struct exec<program<'.',Rest...>,program<Eated...>,idx,Memory,
            output<str...> >{
  new_output = snoc<get_value<idx,Memory>::value,output<str...>>::value;
  value = exec<program<Rest...>,program<'.',Eated...>,
                idx,Memory,new_output>::value;
};
```

Brainf*ck の命令'.'が少し面倒で、C++ではコンパイル時に普通出力させる事は出来ないので、この命令を実装しようと思った場合は出力を適当な場所に溜め込んでいく必要があります。今回は

```
template<char... ops>struct output{};//出力
```

というものを使いました。

あとは少しややこしそうな'['と']'を実装すれば終わりです。どのように実装すれば良いでしょうか?

例えば、[[[ ]]]という入れ子に成った物を考えた場合、ボールドの角括弧が対応づいたものと成ります。'['に対応する']'を見付けたい場合は、

n = 0 とする

まず次の(つまり右の)文字を読む。そして、

読み込んだ文字が'['であれば n = n+1 として 2 へ。

読み込んだ文字が']'で、かつ n = 0 であれば、この']'が対応する']'である。

読み込んだ文字が']'で、かつ n ≠ 0 であれば、n = n-1 として 2 へ。

読み込んだ文字が'['でも']'でもなければ、2 へ。

とすれば良さそうです。']'に対応する'['を見付けたい場合は、同じようにして、ただし左方向へ文字を見て行けば良い事に成ります。

同じような実装になりますので、'['に対する実装だけ貼ります。

```
//search_forward は、現在注目している '['に対応する ']'を見付ける為の関数
//(に見える struct)
template<typename Rest,typename Eated,int c>
struct search_forward;

template<char... Rest,char... Eated>
struct search_forward<program<']',Rest...>,program<Eated...>,0>{
  value = Pair<program<Rest...>,program<']',Eated...> >;
};
```

```
template<char... Rest,char... Eated,int c>
struct search_forward<program<']',Rest...>,program<Eated...>,c>{
  value = search_forward<program<Rest...>,program<']',Eated...>,c-1>::value;
};

template<char... Rest,char... Eated,int c>
struct search_forward<program<'[',Rest...>,program<Eated...>,c>{
  value = search_forward<program<Rest...>,program<'[',Eated...>,c+1>::value;
};

template<char... Rest,char... Eated,char op,int c>
struct search_forward<program<op,Rest...>,program<Eated...>,c>{
  value = search_forward<program<Rest...>,program<op,Eated...>,c>::value;
};

//処理すべき命令が'['だった時
template<int idx,typename Memory,typename Output,char... Rest,char... Eated>
struct exec<program<'[',Rest...>,program<Eated...>,idx,Memory,Output>{
  static const int now = get_value<idx,Memory>::value;
  result = search_forward<program<Rest...>,program<'[',Eated...>,0>::value;
  new_rest = result::fst;
  new_eated = result::snd;
  Then = exec<new_rest,new_eated,idx,Memory,Output>;
  Else = exec<program<Rest...>,program<'[',Eated...>,idx,Memory,Output>;
  value = if_0<now,Then,Else>::value;
};
```

全体のコードは、ココ[*4]に上げておきます。取りあえず Brainf*ck 界では有名な"Hello World!"を出力する次の Brainf*ck プログラム

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

を実行しましょう。

注意すべきは、どのようにプログラムを読み込めば良いのか？という事です。ファイルを読み込む関数は実行時に実行されるのであって、コンパイル時には実行されません。そこで、皆さんおなじみの include を使いましょう！

```
typedef program<
#include "helloworld.hpp"
> whole_program;
```

*4http://github.com/ranha/bf_in_cpp.t

```
typedef exec<whole_program,program<>,0,
                make_memory<10>::value,output<> >::value ans;

int main()
{
  cout << demangle(typeid(ans).name()) << endl;
  return 0;
}
```

実行結果(つまり上のプログラムの ans)は「型」なので、それを C++で出力出来るようにしなければなりません。demangle や typeid は、その為のお呪いです。気にしないでください。

気になる実行結果は

```
 Pair<output<(char)72, (char)101, (char)108, (char)108, (char)111,
(char)32, (char)87, (char)111, (char)114, (char)108, (char)100,
(char)33, (char)10>, memory<0, 87, 100, 33, 10, 0, 0, 0, 0, 0> >
```

となります。72,101,108,108,111,32,87,111,114,108,100,33,10 を ASCII コードとして解釈すると、ちゃんと"Hello World!¥n"になります。やりました!!

1.3 小休止(ゼリーを食べる)

実は 8 月頃から、秋葉原などに行くとそれらしいお菓子[*5] を見掛けるようになりました。

WORD 編集部に、忘れもしないあの 8 月 22 日、90 箱近くこのお菓子が持ち込まれたのです。今も私の席の近くに山積みになっているのですが。朝昼晩食し、無くなれば秋葉原に買いに行くという行為が繰り返されています(繰り返される予定です)。

内容物は桃ペクチンゼリー(4 個)とおまけシール(ただし 1 柄)となっています。更に当たり付きで、当たるとポスター(ただし絵柄は選べない)が貰える仕様になっています。1 箱 200 円。

以下のような感想があります。

もっと働かなくちゃ……。もっと働いて、当たるまで買う……。それが生存本能ッ!!!!!!!!!!!!!!!!

冷やすと結構ヤバい。

このお菓子のお陰で Agda2 を捨てて Coq に移行出来そうな気がします。

なんで"ことりちゃんのカプリコ"にしなかったんですか orz

当たりが出たら筆者にください。

4.結論

C++のテンプレートが、チューリング完全であることを証明出来ました。

もっとも、テンプレートには色々とリアルな制限があるのですがこれをとっぱらった時には、つまり理想的にはチューリング完全になる、という事です。

それから、','命令は実装しませんでしたが、この命令はチューリングマシンを Brainf*ck プログ

*5http://www.movic.jp/info/pockets/index.html

ラムに変換する時には使わないので、チューリング完全である事を示すのに必要ないと判断して実装しませんでした。'.'命令と同じように、予め入力値をテンプレートパラメタとして渡しておいてあげる、などすればこれも実装可能でしょう。

5.後書き

今回テンプレートで Brainf*ck を実装出来た、という事は即ち**コンパイル時**に Brainf*ck のプログラムを実行させる事が出来る、という事になります。実行時間は、実行結果を出力したいのであれば 0 秒とまではいかないのですが、限りなく 0 にする事が出来ます。

それから今回は Brainf*ck のインタプリタのようなものをテンプレートを使って書いたのですが、一方 Brainf*ck のコードから等価な C++のコードをテンプレートメタプログラミングによって生成した場合、そのコードは C++の**コンパイラによって最適化**され、実行可能なコードを得る事が出来ます。つまり、Brainf*ck の高速なコードを吐くコンパイラを得るに等しい事に成ります。この辺のお話は、**Multi-stage Programming** と呼ばれるものとも少し関係してきます。興味がある人は、ググってください。

1.4 ラブライブのこと

ポエムはチラシの裏に書いてな!! という事らしくて、つまりここに書かれているのでこれはポエムじゃないって事です。

どうも順番が最後になってしまいましたが、そもそも、ラブライブとは何なのかを説明する必要がある気がします。いや、既に CD を聞いた皆さんには無用な話だとは思うのですが。

ラブライブ、厳密には"ラブライブ! school idol project"は、電撃 G’s magazine、Lantis、サンライズのメディアミックス型の企画です。 3 社はそれぞれ雑誌、音楽 CD、アニメーションを扱っている会社であって、それぞれの長所で売って行く、要するに分業ですね、という形を取っています。 電撃 G’s magazine の 2010 年 7 月号、8 月号、9 月号、10 月号の誌面上でプロローグなど含めて色々話が出ています。

公式サイト[*6] には何故か余り情報が出ていませんので、興味のある方は電撃 G’s magazine を Amazon などで入手するのが手っ取り早いと思います。

また、ユーザの意見を積極的に取り込んで行くという事に成っているらしく、さっそく人気投票&センターポジションを新たに選ぶ為の"いきなり真夏の総選挙"が先日行われました。既に終わってしまったので、今から投票する事は出来ませんが……。

ラブライブは(少なくとも見た目には)はじまったばかりです。まだ中の人も公開されていません(公開しなくても良いんじゃないかと思うのですが。ちなみに電撃 G’s magazine11 月号で発表らしいです)。ユーザ参加型とは言え、直接個人の意見がそのまま採用されていくというわけでは無いと思うのですが、それでもそういう機会が設けられているのは嬉しいです。こういう明らかな機会を与えられない中でも何とかしていく方が燃えるのですが、私は軟弱なのでこういうのでも良いかなと思ってしまいます。

ラブライブは、この先どうなっていくかは分かりません。だからこそ、私は今から、ここから応援して生きたいと思っています。目が離せませんね!!

取りあえずことりちゃんにカプリコ差し入れして欲しいですよ……。

*6http://gs.dengeki.com/lovelive/

情報科学類誌

WORD

From College of Information Science

非実在WORD編集部号

発行者　情報科学類長

編集長　中 裕太郎

製作・編集　筑波大学情報学群
情報科学類 WORD 編集部
（第三エリア C 棟 212 号室）

2010年9月　初版第一刷発行