平成 27 年 11 月 15 日
京都工芸繊維大学コンピュータ部

Lime 52

はじめに

コンピュータ部部長の小西です。このたびはお忙しい中、本学の学園祭にお越しいただき、我々コンピュータ部にご興味、ご関心を持っていただきありがとうございます。また、この Lime51 号を手に取ってくださったこと、部を代表いたしまして、厚く御礼申し上げます。この Lime は部員の活動の一部を記したものです。日々の活動の成果や部員個人の趣味全開なものまで内容は多岐にわたります。これを通じて、コンピュータ部のことを知っていただき、少しでも、我々の活動に興味を持っていただければ幸いです。最後に、この Lime の編集担当家原君、表紙の作成にあたった矢口君、忙しい中記事を書き上げた部員各位、並びに、OB・OG の方々を含め、日々 我々の活動を支えてくださっているすべての方に感謝し、はじめの挨拶とさせていただきます。

平成 27 年 11 月吉日
京都工芸繊維大学コンピュータ部部長　小西 健太

目 次

1 オレオレ言語の開発

情報工学課程 1 回生 松永大輝

1.1 はじめに

プログラミング言語はすでに星の数ほど存在していますが, やはり自作してみたくなるものです. この記事では, 3 月〜6 月頃に作ったプログラミング言語の説明を行います. 前半でその言語の特徴を, 後半で処理系の実装について説明します. ちなみに今回作った言語の名前は”soramame”です. 開発には C++を使用しました.

1.2 soramame 言語ひとめぐり

soramame 言語をざっくりと紹介します. まずはお決まりのものを.

```
fun main(){
    print("hello,world")
}

---出力---
hello,world
```

この言語は手続き型言語です. 文法は C や JavaScript を知っている人ならなんとなく雰囲気がつかめるであろうものとなっています. 実行は `main` 関数から始まります.

データ型・変数

```
data Point{ x:int; y:int } //構造体宣言 (トップレベルに記述)
```

```
var a:int=9
var b=5.1,c=[("java",true),("c",true),("python",false)]
var d=Point{x=12,y=200}
a=d.y;
print(c[2][0]) //python と出力される
b="kitcc" //コンパイルエラー「型に問題があります。二項演算子 = 左辺:double 右辺:string」が表示される
```

変数への再代入が可能です. 静的型付き言語で, ローカル変数は型推論します. 上の例では, b は double 型, c は string と bool のタプルのリスト ([(string,bool)]), d は Point 型であると推論します. 基本型に int, double, bool, string があります. また, 複合型にリスト ([T], 1 つの型, 可変長), タプル ((T1,T2,...), 異なる型, 固定長), 構造体があります. その他に関数型 (fun(T1,T2,...)=>T), 継続型 (continuation(T)), チャンネル型 (channel(T)) があります.

クロージャ

```
fun makecounter()=> fun()=>void{
    var n=0
    return fun(){ n=n+1; print_int(n); return }
}

fun main(){
    var n=10
    var a=makecounter(),b=makecounter()
    a(); b(); a(); b()
}

---出力---
1122
```

関数は第一級オブジェクトです. つまり, 変数に代入したり, 引数や返り値にできるということです. main 関数では makecounter 関数から返ってきた関数 (a,b に代入されている) を呼び出しています. レキシカルスコープを採用しているので, 変数 n は無名関数が定義されたときのものが使われます. 関数 makecounter からは関数ポインタではなく, クロージャ

が返ってきます. クロージャとは, 関数ポインタと環境のセットのことを言います. a,b それぞれ独立した環境を持っているため, カウントは独立して行われます.

継続

```
fun main(){
    var cont:continuation(void)
    print("do "); print("re "); print("mi ")
    callcc(c){ /*c に継続が代入されている*/ cont=c  }
    print("fa "); print("so "); print("ra ")
    cont()
}

---出力---
do re mi fa so ra fa so ra fa so ra fa so ra .....
```

継続も第一級オブジェクトです. 継続というのは残りの計算をオブジェクトにしたものです. C で大域脱出に使われる setjmp/longjmp では内側からの脱出しかできませんが, 継続はどこからでも呼び出せます [1]. 上の例は, do re mi と表示されたあと, 変数 cont に継続を代入しています. callcc というのはコンピュータクラブではなく call-with-current-continuation の略で, Scheme に倣いました. これは継続を取り出すためのものです. Scheme では関数ですが, soramame では専用の構文となっています. fa so ra と表示された後に, cont に入れておいた継続を呼び出すのでまた fa so ra の表示が始まります [2].

並行実行・チャンネル通信

```
//ボール投げ
fun main(){
    var player=[newchannel(bool,1),
                    newchannel(bool,1),newchannel(bool,1)]
    var player1:fun()=>void = fun(){
        player[0]?
        print("Player 1 catch!\n"); sleep(3000)
```

[1] 別のスレッドからでも呼び出せます.
[2] この後無限ループとなります.

```
        player[randitem([1,2])]!true
        player1()
    }
    var player2:fun()=>void = fun(){
        player[1]?
        print("Player 2 catch!\n"); sleep(1000)
        player[randitem([0,2])]!true
        player2()
    }
    var player3:fun()=>void = fun(){
        player[2]?
        print("Player 3 catch!\n"); sleep(2000)
        player[randitem([0,1])]!true
        player3()
    }

    async player1(); async player2(); async player3()
    player[0]!true
    sleep(100000)
}

fun randitem(list:[int])=>int{
    return list[rand()%(@?list)]
}

---出力---
Player 1 catch!
(3秒待つ)
Player 2 catch!
(1秒待つ)
Player 1 catch!
(3秒待つ)
Player 3 catch!
(2秒待つ)
Player 2 catch!
(1秒待つ)
Player 3 catch!
(2秒待つ)
```

```
.....
```

```
//ストリーム
fun main(){
    var c1=newchannel(int,1)
    var c2=newchannel(int,1)
    var c3=newchannel(int,1)
    async intgen(c1,1)
    async map(fun(x:int){return x*x},c1,c2)
    async filter(fun(x:int){return x%17==0},c2,c3)

    print_int(c3?); print("\n")
    print_int(c3?); print("\n")
    print_int(c3?); print("\n")
}

fun intgen(chan:channel(int),start:int){
    chan ! start
    intgen(chan,start+1)
}

fun filter(f:fun(int)=>bool,in:channel(int),out:channel(int)){
    var value=in?
    if(f(value)){ out!value }
    filter(f,in,out)
}

fun map(f:fun(int)=>int,in:channel(int),out:channel(int)){
    out ! f(in?)
    map(f,in,out)
}

---出力---
289
1156
2601
```

この言語は並行処理を書きやすくするための機能を備えています. 関数呼び出しの前に async というキーワードが付いている箇所がありますが, そうすることで別スレッドで関数が実行されます. Go 言語の goroutine は Go のランタイムが管理する軽量スレッドで, メモリ使用量が小さく大量に生成できるそうですが, これは単なるネイティブスレッドです.

プログラムが並列に実行されるにあたり, スレッド間のデータのやり取りや同期が重要となってきます. そのために soramame ではチャンネルというものが用意されています. チャンネルはスレッド間の多対多での通信手段を提供します. チャンネルを利用することによりセマフォやモニタのような排他制御も可能となります. このチャンネルは双方向であり, FIFO バッファを持ちます. バッファのサイズは 0 以上でチャンネルオブジェクトの生成時に指定します. [チャンネル]?で受信, [チャンネル]![値] で送信です. 受信時に値がなければブロックします. 送信時にバッファがいっぱいだったらブロックします.

上記の 1 つ目の例はボール投げをシミュレーションするもので, 図 1.1 のような構造となっています. player1〜3 はそれぞれ, 自身のチャンネルにボール (ここでは値 true) が送られてくるのを待ちます. 送られてくると, メッセージを表示し数秒待った後, 他の player をランダムに選びボールを送ります. そして自身を再帰的に呼び出すことでその動作を繰り返すようにしています. C 言語等ではこのような書き方をするとスタックオーバーフローになってしまいますが, soramame 言語では末尾呼び出し最適化が行われるため問題ありません (後述). player1〜3 を起動し, player1 にはじめのボールを投げて開始すると, ボール投げの様子が出力されてゆきます. main 関数を抜けてしまうと終了してしまうため, 最後に余分な待ち時間を設けています.

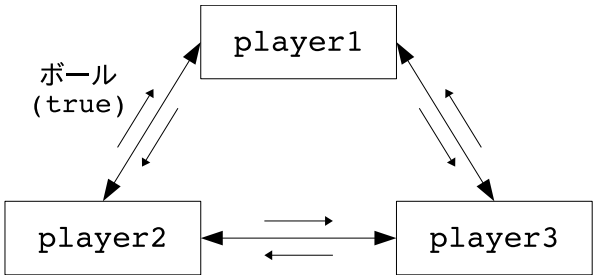

図 1.1: ボール投げの様子

2 つ目の例は, ストリームをチャンネルを用いて実現したものです. ストリームとはデータの流れを抽象化したものです. 例えばファイル入出力はストリームと見ることができます. この例ではチャンネルを用いてそれを表現しています. intgen 関数は, start から 1 ずつ増加させた数値をチャンネルへ送信します. つまり初項 start で公差 1 の等差数列を表現します. filter 関数は, in からの入力を述語 (真偽値を返す関数)f でフィルタし, out へ送信します. map 関数は in からの入力に f を適用した値を out へ送信します. これらの関数間でデータの通り道となるのがチャンネルです. main 関数では c1,c2,c3 の 3 つのチャンネルを通信路とし, 図 1.2 のようなデータの流れをつくっています.

intgen 関数は, 無限ループとなっていることから分かるように無限の数列を表現します.

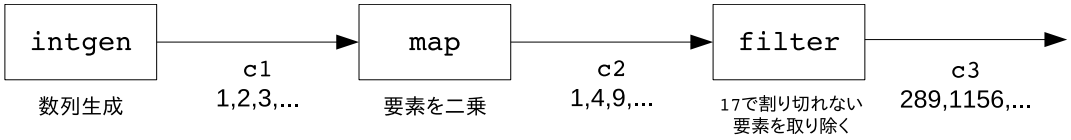

図 1.2: データの流れ

メモリは有限なので無限に続くデータの列を生成することは不可能ですが, 無限とは言っても実際に必要なのはその一部分です. データの列を一度に生成するのではなく, 必要とされた時点で必要な分だけ生成するようにすることで, 見かけ上無限の列を表現することができます. そのようなストリームを遅延ストリームと呼びます. ここではチャンネルの持つ「送信時にバッファがいっぱいだったらブロックする」性質を利用して遅延ストリームを実現しています. それぞれの関数はデータををひたすらチャンネルへ送信しようとしますが, バッファがいっぱいだとそこで一旦停止します. チャンネルへ値が要求され (?で受信され) バッファに空きができると 空いたバッファをまた埋めようとします. `map` や `filter` では入力が間に合っていない場合, チャンネルの性質により入力が来るまで待ちます. この例では, `main` 関数の終わりで末端のチャンネル `c3` から 3 回受信しています. 受信する度に徐々にデータが計算されていきます.

`intgen,filter,map` 関数の呼び出しにはそれぞれ `async` が付いていて, 別スレッドで実行されます. それぞれ並行して実行されないと, 一度ブロックしたらブロックしたままで進まなくなってしまうからです[3]. データの列が揃ってから次の工程へ進むのではなくデータを連続的に処理し, またそれぞれの処理は並列に実行されるので, これはパイプとよく似ています.

ユーザ定義演算子

```
fun main(){
    print_int(2^3^2);print("\n")
    var f= fun(x:int)=>int{ return x*x }
    var g= fun(x:int)=>int{ return x+2 }
    print_int( (f$g)(3) )
}
//累乗
fun ^ binary,right,50(x:int,y:int)=>int{
    return pow(x,y)
```

[3]今回チャンネルを用いて実現したからであって, 「遅延ストリームの実現には並行性が必要だ」という意味ではありません.

```
}
//関数合成 (int->int に限る)
fun $ binary,right,60(f:fun(int)=>int,g:fun(int)=>int)=>fun(int)=>int{
    return fun(x:int){ return f(g(x)) }
}

---出力---
512
25
```

演算子を自分で定義することができます. 関数定義と同じような構文ですが, 引数リストの前に演算子の情報を記述します. 単項演算子であるか二項演算子であるか, 結合規則, 優先順位, の順です.

組み込み関数

```
fun main(){
    glut_openwindow("Lissajous")
    glut_setdisplayfunc(fun(){
        glut_clear(); glut_begin_point()
        var a=4.0,b=3.0
        for(0.0,1000.0,1.0,fun(t:double){
            glut_color3i(255,0,0)
            glut_vertex2i(d2i(sin(a*t)*50.0)+100,
                    d2i(cos(b*t)*50.0)+100)
        })
        glut_end(); glut_flush()
    })
    glut_mainloop()
}

fun for(start:double,end:double,step:double,block:fun(double)=>void){
    if(start>end){return}
    block(start)
    for(start+step,end,step,block)
}
```

現在, 以下の組み込み関数が利用できます.

- 数学 (`sin`, `cos`, `tan`, `abs`, `rand` 等)
- 型変換 (`int` と `double` の相互変換)
- コンソール出力 (`print`, `print_int`, `print_double`, `print_bool`)
- 並列 (スレッドのスリープ, プロセッサのコア数取得)
- グラフィックス (GLUT を利用, 簡単な 2D グラフィックスとマウス・キーボード・タイマーイベント)

上記の例は, リサージュ曲線を描画するものです. `glut_setdisplayfunc` に無名関数を渡して描画関数を登録しています. 組み込み関数とは関係ない話ですが, 多くの言語で構文が用意されている for ループは残念ながら soramame にはありません [4]. しかし再帰とクロージャを使って関数で for ループを実現できます. 図 1.3 が実行結果です.

また, 図 1.4 は soramame で書いたテトリスの実行結果です. キーボードで操作します. 図 1.5 はライフゲームです. マウスで, セルの状態を変更したりシミュレーション速度を変更したりできます.

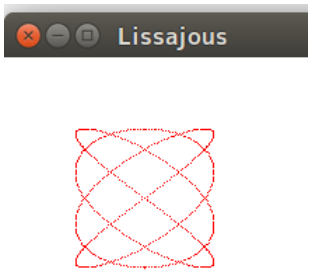

図 1.3: リサージュ曲線

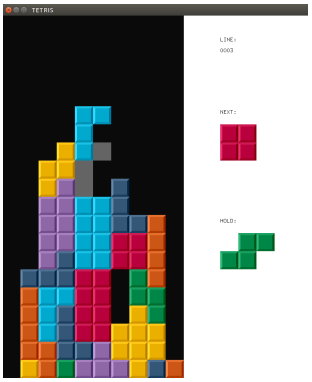

図 1.4: テトリス

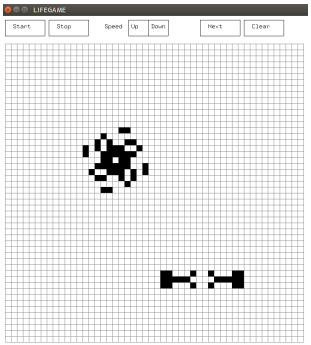

図 1.5: ライフゲーム

1.3 処理系

このセクションでは, 処理系の実装についての説明を行います. 今回作った処理系は, バイトコードを生成し, 仮想マシン上で実行する方式です. ソースコードを字句解析, 構文解析しながら構文木を生成し, 型検査を行った後にバイトコードを生成して仮想マシン上で実行します [5].

[4] while ループはあります.
[5] 生成したコードをファイルに保存できないので, 実行毎にこれらを行います.

再帰下降構文解析

再帰下降構文解析という手法を使えば比較的簡単に構文解析器を作れます. この手法では非終端子ごとに入力のトークン列を消費する関数を定義し, お互いに呼び出しあいます. 当初は再帰下降構文解析を使っていました. しかし, 文法規則と構文解析プログラムを分離できないため文法規則が複雑になるにつれてプログラムの見通しが悪くなってきました. そこで LR 構文解析と呼ばれる手法を使うことにしました.

LR 構文解析

LR 構文解析は, 文法規則から構文解析表を生成してそれを元に構文解析するという手法です. 再帰下降構文解析よりもプログラムはずっと複雑になりますが, 文法を変更する際に文法規則を書き換えるだけで済むため大変便利です. LR 構文解析器を生成するツールとして有名なものに yacc というソフトウェアがあります. それを使おうとしたのですが, ビルドがなかなかうまくいかずイライラし, 自作することにしました. ここからは LR 構文解析の簡単な説明を行います.

$$< whilestatement >::= "while"" (" < expression > ")"" {" < block > "}"$$

これは, while 文の BNF による文法規則です. `while` の後に左丸括弧, 式が来て, 右丸括弧, 左波括弧, 文の並びが来て, 右波括弧, であれば, $< whilestatement >$ とみなす, という意味です. 今回作成した構文解析器では, 文法規則を配列の形でソースコード内に直接記述するようにしました. 以下がその一部です.

```
SyntaxRule SYNTAXRULE[SYNTAXRULECOUNT]={
 (中略)
{{whilestatement,WHILE,LPAREN,expression,RPAREN,
          LBRACE,block,RBRACE,SYNTAXEND},while_reduce},
 (中略)
};
```

構文解析表の作成方法については省略しますが, 構文解析表は LR 構文解析の要となります. 構文解析表は, 「状態番号△の時にトークン○○が来たら, ☆☆しろ」という情報を持っています. 「トークン○○が来たら, 状態□へ移れ」は shift と呼ばれます. 様子見するといった感じです. 「トークン○○が来たら, ★番目の文法規則を適用して状態◇へ移れ」は, reduce と呼ばれます. 文法規則の右側の並びが揃ったので, まとめるということです. while 文の例でいえば, shift を続けていき, 途中で $< expression >$ や $< block >$ に reduce され

ながら, 最終的に $< whilestatement >$ に reduce されます. reduce される時に, その文法規則に対応した関数が呼ばれます. 上記のソースコードでは一番後ろの `while_reduce` が, $< whilestatement >$ へ reduce されるときに呼ばれる関数です. この関数には文法規則中の $< expression >$ や $< block >$ が渡されるためそれを元に構文木を構築していきます.

式の構文解析

式の構文解析で問題となるのは, 優先順位と結合規則です. これらの問題は, 文法規則を工夫することにより解決します. しかし, 今回作る言語は新たな演算子を定義できるようにしようと思っていました. 構文解析しながら動的に文法規則をいじって... 気が狂いそうです. 今回は, 式の構文解析を後から, 別の方法で行うという方法をとりました. はじめの構文解析で式を, 演算子と項の並びとして保持しておきます.

$$< expression >::= \{< primary > | < operator > | < parenexpr >\}+$$

$< primary >$ は数値や文字列, 識別子などです. この段階で優先順位や結合法則はまったく考慮されず, リストとして構文木に入れられます. また, 演算子定義を見つけるたびに演算子テーブルへ登録しておきます. そして型検査時に構文木をたどるタイミングで, 優先順位や結合法則を考慮した構文木に展開します. 操車場アルゴリズムを利用して中置記法から逆ポーランド記法へ変換した後, 構文木を作っています. このタイミングで定数畳み込みを行います.

型検査

静的型付け言語なので, コンパイル時に型の整合性をチェックします. 構文木をたどり, 木の深いところから型を決定していきます. そして記述されている型と一致するかを確認します. グローバル変数や関数定義には型を明記する必要がありますが, ローカル変数は型推論を行います. つまり, 得られた型の情報をそのまま使います. また C 言語等の言語では暗黙の型変換 (例えば, 整数と実数の間での変換) が行われることがありますが, soramame ではすべて明示的に型変換を行う必要があります [6]. これは暗黙の型変換による, 発見が難しいバグを生まないようにするためです.

コード生成

生成されるコードは仮想マシン (後述) のバイトコードです. この仮想マシンはスタックマシンであるため, 例えば $3 + 4 * 5$ だと,

[6] 組み込み関数 `i2d`(`int` から `double`), `d2i`(`double` から `int`) を使用できます.

```
ipush3 //整数 3 をプッシュ
ipush4 //整数 4 をプッシュ
ipush5 //整数 5 をプッシュ
imul //スタック上の 2 つの整数を掛けたものをプッシュ
iadd //スタック上の 2 つの整数を足したものをプッシュ
```

のようなコードが出力されます[7].

バイトコードにおいて, 変数の参照は「さかのぼるスタックフレームの数」と「スタックフレームの変数領域での変数の位置」の組で表されます. スタックフレームをさかのぼるというのは, ここでは, 呼び出し元へとさかのぼるのではなく, 「自身を生成した関数のスタックフレームへのポインタ」をさかのぼる, ということを意味します. つまり, 構文構造上いくつ外側のブロックに変数が存在するか, ということです.

```
fun main(){
    var x=12, y=34
    var f=fun(){
        print_int(x)
    }
    print_int(x)
    print_int(y)
}
```

main 関数の内側で定義された関数 f 内の print_int(x) の x は, ひとつ外側の main 関数で定義された x なので, (1, 0) と表されます. また, main 関数内の print_int(x) の x は, (0, 0) と表されます. print_int(y) の y は, main 関数の 2 つ目のローカル変数であるため, (0, 1) と表されます. レキシカルスコープを採用しているため, これらの値をコンパイル時に決めることができます. 変数の参照をこのように表すことで, 実行時に変数探索をしなくて済むのです.

仮想マシン

処理系には主にコンパイラ方式とインタプリタ方式があります. 今回は仮想マシンのバイトコードにコンパイルして, 仮想マシン上で実行させることにしました. 有名な仮想マシンに JVM や CLR 等がありますが, そんな立派なものを使いこなせるようになるよりも自分

[7]実際は定数畳み込みにより直接計算結果をプッシュするコードとなる場合もあります.

でしょぼい仮想マシンを作ったほうが楽しそうなので作ることにしました. とはいえどんな命令が必要となるのか, JVM の命令セットを参考にしました. コンスタントプールの仕組みも JVM のまねです. 今回作った仮想マシンは, スタックマシンです. オペランドスタックを使いながら演算を進めていきます. この仮想マシンのスタックフレームは以下の要素で構成されます.

- 変数領域
- プログラム・カウンタ
- オペランドスタック
- 関数へのポインタ
- 呼び出し元のスタックフレームへのポインタ
- 自身を生成した関数のスタックフレームへのポインタ

文字列や浮動小数点数, 関数等はコンパイル時に定数領域 (コンスタントプール) に保管されます.

継続

スタックフレームをたどり, オペランドスタックをコピーするという単純な方法を取りました. 継続を取り出すたびに大きなコストのかかる, おそらく最も効率が悪い実装です [8]. `makecontinuation` という命令で継続オブジェクトが生成され, `resumecontinuation` で呼び出します.

末尾呼び出し最適化

```
int main(){
    f();
    return 0;
}

void f(){
    printf("f");
    g();
}
```

[8] Lime34 の湯浅先輩の Scheme では継続を取り出すのにコストのかからないような実装となっています.

```
void g(){
    printf("g");
    f();
}
```

これはCで書いたプログラムです. これはすぐにスタックオーバーフローしてしまいます. ところが, これに相当するプログラムをsoramameで実行するとスタックオーバーフローすることなくfgfgfgfgfg...と無限にfgを出力します. ところで, これは次のように書き換えることができます.

```
int main(){
    goto f;
f:
    printf("f");
    goto g;
g:
    printf("g");
    goto f;
    return 0;
}
```

このように, 末尾呼び出しはgotoで書きなおすことができます. つまりスタックを消費する必要がないということです. soramameは末尾呼び出しで余分なスタックを消費しないようになっているのです.

以下のVM::Run関数は, 仮想マシンの核となる関数です.

```
//vm.cppの一部
VMValue VM::Run(shared_ptr<Flame> CurrentFlame,bool currflame_only){
    (中略)
    try{
        while (true){
            (中略)
            bytecode = OPERAND_GET;
            switch (bytecode){
```

```
            case ipush:
                v.int_value = OPERAND_GET;
                STACK_PUSH(v);
                break;
            case pushim1:
                v.int_value = -1;
                STACK_PUSH(v);
                break;
            (以下ひたすら case が続く)
            }
        }
    }catch (exception& ex){
        (中略)
    }
}
```

バイトコードの種類だけひたすら case で分岐しています. この仮想マシンでは, オペランドスタック上に引数を右から順番に積み, 関数へのポインタを積み, invoke 命令を呼ぶことで関数が呼び出されます. invoke 命令は 2 つの引数を取ります. 末尾呼び出しかどうかのフラグと, 別スレッドで実行するかどうかのフラグです. コンパイル時に末尾呼び出しだと分かると, 即値で印を付けておき実行時にそれを見て不要なものを記憶しないようになっています.

並列実行

同じように, 実行時に invoke 命令の引数をみて並列実行すべき (async 文を使った) だと分かれば, VM::Run を別スレッドで開始するだけです.

チャンネル通信

sendchannel/receivechannel 命令によって通信を行います. キューと条件変数を使い, 待ちスレッドを管理します. ひとつのチャンネルへ複数のスレッドから同時にアクセスされる可能性があるため, ロックを使って排他制御を行います.

1.4 おわりに

今回, はじめてプログラミング言語をつくりました. 最初に電卓のようなものを作ってから, 変数, 関数, クロージャ, 継続, 並行実行とチャンネル通信... といった具合に徐々に拡張

していきました. soramame 言語には自慢できる特徴が無いので, 今度はもう少し個性的な言語を作りたいです. また, この記事の中で GC について触れませんでした. GC は手を抜いてスマートポインタ (shared_ptr) を利用しているのですが, shared_ptr は参照カウント方式の GC であるため循環参照ができた場合解放されないという欠点があります. この処理系ではクロージャを作ると循環参照ができてしまうためこれは大きな問題となります. GC のこともちゃんと考えないといけないなと思いました.

soramame のソースコードは, https://bitbucket.org/matsud224/soramame で公開しています. また, 第一回 カーネル／ VM 探検隊@名古屋で発表させていただいた際のスライドが http://www.slideshare.net/matsud224/soramame にあります.

参考文献

[1] 前橋和弥, プログラミング言語を作る (技術評論社)

[2] LLVM によるプログラミング言語の実装, http://peta.okechan.net/blog

[3] 文法と言語-自由文脈文法と LR 構文解析-, http://vrl.sys.wakayama-u.ac.jp/SS/

[4] Scheme 演習, http://www-ui.is.s.u-tokyo.ac.jp/~hara2001/scheme

[5] Java アセンブラ「Jasmin」でバイトコードの世界を覗いてみよう, http://www.hakkaku.net/series/java アセンブラ「jasmin」でバイトコードの世界を覗いて

[6] 数理科学的バグ撲滅方法論のすすめ, http://itpro.nikkeibp.co.jp/article/COLUMN/20060915/248230/?s2p

2 数式を計算する計算機

情報工学課程 1 回生 西阪 和貴

2.1 概要

四則演算の数式を計算する計算機を c++で実装します．直接計算するのは難しいので，一旦「逆ポーランド記法」という方式に数式を変換し，その上で計算します．
例えば，(3+5)*2 と入力したら 16 と返すのを目標としています．

2.2 逆ポーランド記法とは

逆ポーランド記法とは，私達が普段口に出して数式を読み上げる時の順番で数式を書く方法です．

(例 1)
[算数の時の計算式] 3 + 5
[逆ポーランド記法] 3 5 +
[口に出して読む時] 3 に 5 を たす

(例 2)
[算数の時の計算式] (3 + 5) * 2
[逆ポーランド記法] 3 5 + 2 *
[口に出して読む時] 3 に 5 を 足して 2 を かける

この逆ポーランド記法はコンピュータにとっても解釈のしやすい方法なのです．

2.3 逆ポーランド記法の計算式を計算する

上でも書きましたが，計算は比較的簡単にできます．カッコを先に計算する，とか考える必要がないからです．前から読んでいって，数字なら記憶しておく，記号なら覚えていた数字を計算する．これだけです．

(例 1)
「3 5 +」を前から読んでいって
3…3 を覚える
5…5 を覚える
+…おぼえていた 3 と 5 を足した 8 を覚える (このとき 3 と 5 は記憶から消します)
最後に記憶に残っている 8 が答えになります.

(例 2)
「3 5 + 2 *」を前から読んでいって
3…3 を覚える
5…5 を覚える
+…おぼえていた 3 と 5 を足した 8 を覚える (このとき 3 と 5 は記憶から消します)
2…2 を覚える
*…おぼえていた 8 と 2 を掛けた 16 を覚える (このとき 8 と 2 は記憶から消します)
最後に記憶にのこっている 16 が答えになります.

以下は c++で実装したものです.

```
//既に逆ポーランド記法に変換している計算式を計算する
double calc(vector<string>& input) {

    stack<double> stack;

    if (input.empty()) {
        cout << "nothing to calc" << endl;
        return NAN;
    }

    for (int i = 0; i < input.size(); i++) {
        if (input[i] == "+") {
            //数字が 2 つ stack にないとおかしい
            if (stack.size() < (size_t)2) {
                cout << "syntax error" << endl;
                return NAN;
            }
            //stack から 2 つとってきて，その 2 つは stack から消して
            double hoge = stack.top();
```

```
            stack.pop();
            double piyo = stack.top();
            stack.pop();
            //計算して stack に push
            stack.push(piyo + hoge);
        }
        else if (input[i] == "-") {
            if (stack.size() < (size_t)2) {
                cout << "syntax error" << endl;
                return NAN;
            }
            double hoge = stack.top();
            stack.pop();
            double piyo = stack.top();
            stack.pop();
            stack.push(piyo - hoge);
        }
        else if (input[i] == "*") {
            if (stack.size() < (size_t)2) {
                cout << "syntax error" << endl;
                return NAN;
            }
            double hoge = stack.top();
            stack.pop();
            double piyo = stack.top();
            stack.pop();
            stack.push(piyo * hoge);
        }
        else if (input[i] == "/") {
            if (stack.size() < (size_t)2) {
                cout << "syntax error" << endl;
                return NAN;
            }
            double hoge = stack.top();
            stack.pop();
            double piyo = stack.top();
            stack.pop();
            stack.push(piyo / hoge);
```

```
        }
        //記号じゃなければ数字 (なはず) なので stack に push
        else {
            stack.push(atof(input[i].c_str()));
        }
    }

    //最後に覚えていた数字が答え
    return stack.top();

}
```

input は”13”とか”+”とかが並んだものです．数字と演算子をひとつずつ切り離して並べてあります．

上の説明で言った「記憶」はコード内では stack にあたります．数字は push で記憶に貯めて，演算子なら pop で 2 つ記憶から消して，計算したものを push で記憶に貯めます．

エラーだったら計算できませんでしたの意味で NAN(Not a Number) を返しています．

いくつか例を．数字と演算子を切り離すだけの関数を通してから上記の関数に計算させます．

入力: 5 3 +
出力: 8
入力: 2 3 + 4 *
出力: 20
きちんと計算できています．

2.4 逆ポーランド記法に変換する

逆ポーランド記法の計算式を計算することはできました．次は，その前段階として，算数の時の計算式を逆ポーランド記法に書き直す方法です．人間を 1 人用意して読み上げさせてもいいのですが，コンピュータにやらせます．

この変換はちょっと難しいです．

数字の順番は変わらないので，前から順番に読んでいって演算子の位置を変えます．

カッコでくくった部分は何よりも先に計算します．逆ポーランド記法では先に書かれたものを先に計算するので，カッコでくくった部分は前に出力します．同じように，カッコは掛け算や割り算より先，掛け算や割り算は足し算や引き算より先なので，カッコの中身は掛け

算や割り算より先に，掛け算や割り算は足し算や引き算より先になるように出力します.

c++で実装したものが以下です.

```
//逆ポーランド記法に変換する
void convert(vector<string>& input, vector<string>& output) {

    stack<string> stack;

    if (input.empty()) {
        cout << "nothing to calc" << endl;
        output.clear();
        output.push_back("NAN");
        return;
    }

    for (size_t i = 0; i < input.size(); i++) {

        if (input[i] == "+" || input[i] == "-" ||
            input[i] == "*" || input[i] == "/") {

            //掛け算の方が先，足し算は後
            while (stack.empty() == false &&
                   priority(input[i]) <= priority(stack.top()) ) {
                output.push_back(stack.top());
                stack.pop();
            }
            stack.push(input[i]);

        }
        else if (input[i] == "(") {
            stack.push("(");
        }
        //閉じカッコがあったら開きカッコまで遡っていって output に
        else if (input[i] == ")") {

            if (stack.empty()) {
                cout << "parenthesis error" << endl;
```

```
                output.clear();
                output.push_back("NAN");
                return;
            }

            while (stack.top() != "(") {
                output.push_back(stack.top());
                stack.pop();
                if (stack.empty()) {
                    cout << "parenthesis error" << endl;
                    output.clear();
                    output.push_back("NAN");
                    return;
                }
            }
            stack.pop();

        }
        //数字 (なはず) なのでそのまんま output に
        else {
            output.push_back(input[i]);
        }

    }

    while (stack.empty() == false) {
        output.push_back(stack.top());
        stack.pop();
    }

}
```

priority は，+と-なら 1 を，*と/なら 2 を，数字なら 0 を返すだけの関数です．カッコの中身は掛け算や割り算より先に，掛け算や割り算は足し算や引き算より先になるように，です．

2 つほど例を．入力も出力も string 型の配列ですが見えにくいので簡単にして書きます．
入力: 5 + 3

出力: 5 3 +
入力: (5 + 3) * 2
出力: 5 3 + 2 *
きちんと変換できています.

この変換をしてから前章の計算をすることで，(3+5)*2 のような数式を計算することができます.

2.5 他の工夫

入力から出力までひと通り実装して工夫した点です

- マイナス
  - – の記号は，引き算の演算子なのか符号としてのマイナスなのかの判別も必要です．逆ポーランド記法では特別な手段を使わないと符号としてのマイナスが表現できないからです．符号の時には~(チルダ) を使うなど，特別な方法を用いてもいのですが，今回は自分が思うように実装しました．文の最初または開きカッコの次に – のつく数字があったら”−1”など負の数として解釈し，それ以外は引き算の演算子として解釈します.

- 切り分ける
  - convert の引数は string の可変長配列です．数式をそのまま渡すのではなく，切り分けてから渡します．こうすることで負の数に対応しています.
    また，同時に数字，規定の演算子，カッコ以外の文字があったらエラーを出します．数字や演算子の間にスペースが入っていたら全部無視します.

2.6 実行結果

$(3+5)*2 \to 16$
当初の目標です．達成.
$() \to$ nothing to calc
$((() \to$ parenthesis error
エラーもちゃんとできてます.
$2.1/2.6 \to 0.807692307692$
整数なら整数として，小数なら小数として出力.
$-1231+1230 \to -1$
$3*(-1)+8 \to 5$
マイナスも大丈夫です.

$10 + (8 + (-2) * 2) - 2.3 * (-3/4 - 1) + 0.1414 * (-6.434/(-3.2315)) \rightarrow 18.3065310537$
長い計算式でもきちんと動作します.

2.7 最後に

まずは完成してよかったです．機能としては最低限のものですが，思っていた通りの形までこぎつけました.

思いつきでやると決めた割に思ったより苦労してしまいました．std::cout で整数か小数か判別して出力してくれるのと，慣れている方がいいだろうと思って c++で書きました.

エラーの処理汚いですね．初心者丸出しです．入力がきちんとした形式でない場合，カッコの数がおかしかったり計算が不可能な式が入力された場合にどのような処置をすればいいのかわからず，結果こんな感じになってしまいました．開きカッコと閉じカッコの数が合わないなど，入力された段階でわかる間違いがあれば計算より前にエラーとするなど，まだ工夫が必要です.

2.8 参考文献

- http://7ujm.net/etc/calcstart.html
- http://www.gg.e-mansion.com/ kkatoh/program/novel2/novel207.html

3 図書管理システム

情報工学課程 3 回生 小西 健太

3.1 はじめに

コンピュータ部にはたくさんの本があります。種類はプログラミング関係の本からネットワーク関係の本まで多岐にわたりますが、その冊数は大きなものとなっています。また、これらの本はコンピュータ部の部員であれば貸し出し帳に記載すればだれでも借りることができます。そんなコンピュータ部の書籍事情ですが、冊数が比較的多く、時折部室のお引越しもあるため、すべての本の把握が難しくなってきています。このため、今夏の開発として図書管理システムを作成することにしました。

3.2 理想の設計

今回の開発では当初の (理想の) 設計として、次の項目を設定しました。

- マルチプラットフォーム化を図るため使用する言語は Java とする。
- 実際に実行される際は Java アーカイブファイルとする。
- データの管理には SQL により操作可能なデータベースを用いる。
- 収蔵されている本の一覧の表示、検索ができる。またこれを PDF として出力できる。
- 貸し出されている本の一覧が表示できる。これを PDF として出力できる。
- 新しい本を新規に登録できる。
- 登録されている本を削除できる。但し、削除を示すデータを記録するのみとしデータベースからの削除は行わない。
- バーコードリーダーから本の JAN コードを読み込むことができこれにより本の貸し出し、返却が行える。
- 返却期日の迫っている、または期日が過ぎているユーザーに対して、自動的にメールを送信する。

- 貸し出し中の本を借りたい場合、その返却待ちを設定でき、返却されれば、返却を待っているユーザーに対してメールを送信する。
- ユーザーには標準ユーザーと管理者ユーザーの別を設ける。
- 自身の登録情報を変更できる。

また、管理者ユーザーのみが持つ機能として次の機能を設定しました。

- テーブル”ユーザー一覧”を適宜変更できる。
- 貸し出し履歴のテーブルを適宜変更できる。
- 使用するデータベースの場所を変更できる。

3.3 データベースの設計

今回のシステムを作成するためにはデータベースでのデータ管理が適切と考えました。このため、SQL で操作可能なデータベースとして関係データベースを設計しました。データベースは 4 テーブルからなり、第三正規形となるよう設計しました。作成したテーブルを次に示します。

3.3.1 テーブル Book

部室にある本を登録するためのテーブルです。データの要素として次を持ちます。

Manage_number
: 1 つの本に振られる重複の無い番号です。データベース上で本を一意に識別します。
INTEGER NOT NULL UNIQUE PRIMARY KEY

Book_name
: 本のタイトル。TEXT NOT NULL UNIQUE

Person_name
: 著者の名前です。TEXT NOT NULL

Publisher
: 出版社です。TEXT NOT NULL

ISBN10
: ISBN10。TEXT UNIQUE

ISBN13
: ISBN13。TEXT UNIQUE

JAN_code_1
JAN コード 1 つ目。INTEGER

JAN_code_2
JAN コード 2 つ目。INTEGER

Price
本の価格。INTEGER

Quantity
収蔵冊数。INTEGER NOT NULL

Registration_day
本が登録された日。TEXT NOT NULL

Update_day
本の情報が更新された日。TEXT NOT NULL

Registration_user
本を登録したユーザー。TEXT NOT NULL

Last_modified_user
最後に本の情報を更新したユーザー。TEXT NOT NULL

Delete_bit
本を削除した場合 1 とする。INTEGER

3.3.2 テーブル Rental_history

貸出履歴を記録するテーブルです。データの要素として次を持ちます。

Manage_number
本の識別番号。INTEGER NOT NULL CHECK(Manage_number¿=0)

User_name
本を借りているユーザーのユーザー名。TEXT NOT NULL

Rental_day
貸出日。TEXT NOT NULL

Due_day
返却予定日。TEXT NOT NULL

Return_day
実際の返却日。TEXT

Rental_number
データベース上で貸出情報を一意に識別します。

3.3.3 テーブル Rental_wating

返却待ちを記録するためのテーブルです。データの要素として次を持ちます。

Manage_number
本の識別番号。INTEGER NOT NULL

User_name
返却待ちをしているユーザーのユーザー名。TEXT NOT NULL

Wait_limit
返却待ちの期限。TEXT

3.3.4 テーブル User

ユーザーの一覧を記録するテーブルです。データの要素として次を持ちます。

UserName
ユーザー名。TEXT NOT NULL UNIQUE PRIMARY KEY

PassWord
ユーザー認証用のパスワード。TEXT

Name
ユーザーの氏名。TEXT NOT NULL

MailAddress
ユーザーのメールアドレス。TEXT

latest_login
最終ログイン日時。TEXT

Authority
管理者の場合 1 を、標準ユーザーの場合 0 を記録しています。TEXT NOT NULL

3.4 GUI について

3.4.1 GUI とは

GUI とはグラフィカルユーザーインターフェイスのことであり、一般のソフトウェアで広く使用されています。今回の開発では図書管理システムであることから大きな文字とわかりやすい操作が求められました。このことから次の点に注意し、設計しました。

- 文字は大きく読みやすいこと。
- ボタンをクリックした際の応答がわかりやすいこと。
- 常に戻るボタンを設け、操作の取り消しが容易であること。

ここで、実際に作成した GUI の一部のスクリーンショットを図 3.1〜3.7 に示します。

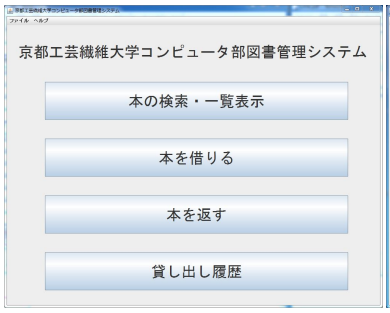

図 3.1: トップ画面

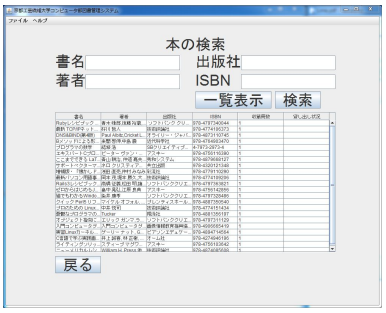

図 3.2: 本の検索

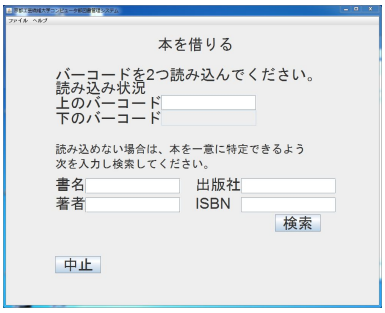

図 3.3: 本の貸し出し

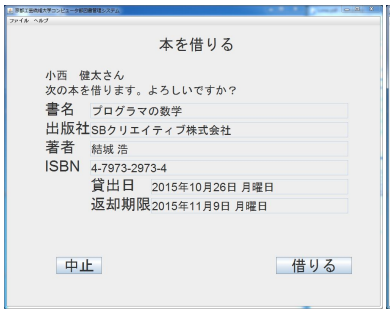

図 3.4: 貸し出し確認

図 3.5: 本の貸し出し状況

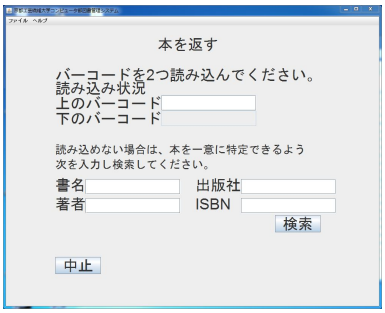

図 3.6: 本の返却

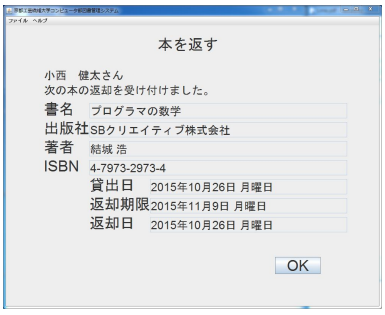

図 3.7: 本の返却確認

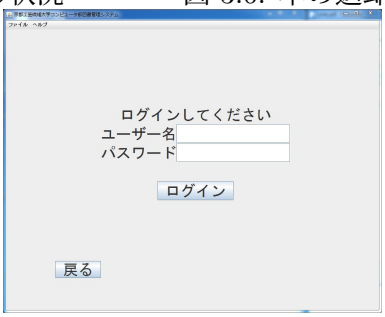

図 3.8: ログイン認証

3.4.2 GUI の設計と苦労した点

今回の GUI 設計にあたっては、パワーポイントを用いてプロトタイプを作成しました。これを基にレイアウトを決定し実装しました。任意の場所に任意の項目を配置するため Java のレイアウト機能を無効化し、座標指定で項目を配置するようにしました。配置に関してはこの方法で容易に決めることができましたが、図 3.3 のようにバーコードリーダーからの入力を受け付ける必要がある場合、初期ではどのテキストフィールドが選択されているか不定のため、スムーズな動作ができませんでした。ここで、これを解消する手段として、バーコードリーダーによる入力を受け付けるような画面では、バーコードリーダーの入力が入るべきフィールドに対して requestFocus() メソッドを呼び出すことによりカーソルを移動させておく方法を使用しました。また、バーコードリーダーでの読み取りが複数回並んで行われるような場合は、バーコードを一つ読むごとに次のフィールドに対し requestFocus() メソッドを呼び出し、スムーズな読み込みが行えるよう工夫しました。

また、ログイン画面では SQL の脆弱性を突き、ログインされないよう注意しました。

3.5 各機能の実装について

3.5.1 貸し出し機能の実装について

貸し出し機能はテーブル Book,Rental_history,User を操作することにより実装されています。実際の工程では次の過程を経ます。

1. ログイン認証を求め、成功した場合ユーザー名を獲得する。
2. 貸し出される本の JAN コードをバーコードリーダーより獲得する。または、書籍名等より本を一意に獲得する。
3. 入力された本が現在すべて貸し出されていないか、データベースに登録されているかを確認し貸し出し可能であることを確認する。
4. テーブル User よりユーザーの氏名を引き出し、氏名と本の確認画面を表示する。
5. 確認されれば、テーブル Rental_history にレンタル情報を書き込む。

3.5.2 返却機能の実装について

返却機能はテーブル Book,Rental_history,User を操作することにより実装されています。実際の工程では次の過程を経ます。

1. 返却される本の JAN コードをバーコードリーダーより獲得する。または、書籍名等より本を一意に獲得する。
2. 入力された本が実際に貸し出されていることを確認する。

3. 貸し出されているユーザーが 1 人の場合はこのユーザーの氏名をテーブル User から引き出し、テーブル Rental_history に返却情報を書き込み、返却完了画面を表示する。

4. 貸し出されているユーザーが複数いる場合はログイン認証を求め、どのユーザーからの返却なのかを獲得する。

5. ログインの結果、本を借りているユーザーからのログインであった場合は返却情報を書き込み、返却完了画面を表示する。

6. ログインの結果、該当する本を借りていないユーザーからのログインであった場合は、エラーとする。

3.5.3 本の検索・貸出履歴の実装について

本の検索機能はテーブル Book を、貸出履歴の機能はテーブル Book,Rental_history,User を用いて実装されています。各機能実行時に必要な情報を各テーブルから引き出し結合することにより機能します。

3.6 これからの開発について

冒頭の理想論に挙げたように、この開発では未実装の機能が多数あります。現在は、一通りの本の貸し出し、返却機能が使えるようになった程度です。しかしながら、冒頭に挙げた機能は現在設計・実装が済んでいるデータベースを使用することにより実現可能であると考えられます。このことから、これ以降の開発は、機能追加という形で実現できればと考えています。その一つとして新規の本の追加機能を現在作成していますが、すべての入力項目が埋められていないと正しく動作しないバグが発生しています。ISBN10 が割り当てられていない本も存在するので、これからの課題だと考えられます。このため、これからの目標は、こういったバグを排除しながら、機能を一つずつ実装していくことです。また、現在、認証用のパスワードは平文で保存さていますが、将来的には暗号化し保存できればと考えています。

3.7 おわりに

この図書管理システムは部室にいるときにふと思いついたものでした。本棚をみるとたくさんの本があり、今年の部室のお引越しでもその多さを実感しました。そして、部室にいったいどれくらいの本があるのか、どういった本があるのかよくわかっていない自分に気づきました。そこで、思いったったのが書籍の情報をデータベース化しようというものでした。そこから発想がぶっ飛んでいき、最終的に図書管理システムへと行きつきました。このシステムが実際に部として使用されることはないと思いますが、使ってみたいと思ってくれる人がいれば幸いです。

4 数当てゲーム

情報工学課程 １回生 矢口　喬脩

4.1 はじめに

最初にLime用に作ったプログラムは線形代数における行列の積の計算プログラムでした。確かに線形代数では役に立つでしょうが、松ヶ崎祭の展示としては楽しそうではなかったので、今回は数当てゲームを新たに作り、それを展示作品に選びました。内容的には情報工学課程１回生のソフトウェア演習より簡単なものになってしまった感じがします。プログラミング上級者のみなさまには退屈かも知れませんが、最後までお付き合いいただけたら嬉しいです。

来年、コンピュータ部の門をたたいてくれた１回生希望者には、行列の積の計算プログラムを配布予定です。（いらないと言われると悲しいのですが … 誰かもらってください !! ）

4.2 前置き

数を当てるゲームなので派手なグラフィックやかわいいキャラが採用されていません。いえ、したくてもできなかったのが実情です。学習の至らなさを痛感しました。

4.3 ゲーム内容説明

あらかじめ数当てプログラム（以下、本プログラム、と呼称します。）が設定した二桁の数字を当てていくという、いたってシンプルなゲームです。ただし、１ゲーム内の最大入力回数は５回であり、その回数未満で正解を導く必要があります。

4.4 なぜ　二桁なのか？

本プログラムはなぜ当てる桁数が二桁なのか？と疑問をもたれた方がいらっしゃるかもしれないので、説明しておきます。

簡潔に言うと、プレイヤーのゲームの拘束時間を短くするためです。

桁数が多くなるにしたがって、クリアまでの時間は長くなります。すると、たいしてプレイのメリットもないゲームへの集中力などそう長く続くわけもなく、そのゲームはプレイ

ヤーにとって、退屈な作業になってしまいます。では、本プログラムのように魅力の少ないゲームを遊んでもらうにはどうすればよいでしょうか。そう、プレイヤーがゲームに興味を失うまでの時間までにクリアか、ゲームオーバーという結果を返せばよいのです。そうすればプレイヤーは少なくともゲームに苦痛を感じることがなく、ゲームプレイできるでしょう。ゲームという限り、娯楽的要素であるので、苦痛を感じるゲームなど本末転倒であるわけですから。

4.5 初歩的技術でも楽しめる工夫を！

本プログラムもゲームの端くれを名乗る以上、プレイヤーを楽しませる方法を考えなければいけません。どうすれば本プログラムをちょっとでも面白いものにできるでしょうか。そもそもゲームの面白さとは何でしょうか。美麗なグラフィックでしょうか、それとも作りこまれたシナリオでしょうか。もちろんそれらも大切な要素です。しかし、私はもっと大切なものがあると考えました。それは、クリアできるか、できないかという手に汗握るようなドキドキ感だと思います。難所を間一髪ですり抜ける、その瞬間がゲームをして一番楽しい瞬間ではないでしょうか。　さて、本題に戻ります。本プログラムは美麗なグラフィックも作りこまれたシナリオも存在しません。なら、ドキドキ感を組み込めないか、そう考えました。そこで勝つか、負けるかギリギリまで競り合うように本プログラムの最大入力試行回数を調整しました。

4.6 なんとなく二分探索

大学の前期の授業で聞きかじった二分探索の考え方を参考にして最大入力試行回数を調整しました。以下、中央の値を入力画面で順次入力していく状況を想定します。

今回は 0 から 99 までの自然数 100 個のデータがあると考えることができます。そのとき、中央の値は 50 とすることができます。この 50 という値に対して High か Low かがヒントで示されます。仮に High とヒントで示されたとしましょう。そのとき、50 と 100 の中央の値は 75 となります。仮にまた High とヒントが示されると 75 と 100 の中央の値は 86 となります。仮にまた High とヒントで示されると 86 と 100 の中央の値は 93 になります。ここまで 4 回の入力試行が行われており、正解は 93 から 100、もしくは 86 から 93 の間に存在すると確認できたわけですから、五回目の非常に正解確率は高まっているわけです。しかし、正解の選択肢は 7 個あるわけですから、そう簡単には正解できません。もし 6 回入力試行が許されているとし、五回目の入力で High というヒントが与えられるとすると、93 と 100 の中央の値は 96 となり、93 から 96、または 96 から 100　に正解が含まれており、正解するのは五回目の入力より非常に簡単になってしまいます。つまり、五回目の入力を最大入力試行回数とすることで、適度なゲームバランスを実現できると考えて、本プログラムの最大入力試行回数を定めました。

4.7 不正入力対策

0 から 99 の自然数以外が入力されるとプログラムが警告を発します。仏の顔は三度までとも言いますので、三度の不正入力を行うと、プログラムが怒って言うことを聞いてくれなくなる、というのは冗談で、プログラムを強制終了します。

4.8 ソースコード

```
/*
 ================================================================================
 Author      : Yaguchi Takanobu   the member of KITCC
 ================================================================================
 */

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define ERROR -1
int main(void)
{
      int i,j; //ループ変数
      int data[100]; //ランダムな値を保管する配列
      int output_num; //選定したランダムな値を記憶する変数
      int num_play; //プレイ回数を記憶する変数
      int penalty=0; //不正入力回数を記憶する変数
      int answer; //プレイヤーが入力する値を記憶する変数

      srand(time(NULL));

      for(i=0;i<100;i++) //ランダムな値を生成する準備
     {
            data[i]= rand()%100;
     }

      output_num = data[rand()%100]; //0~99 の中から値を一つ決定。

      num_play = 5;

     //数字入力の繰り返し
```

```
      for(j=1;j<=5;j++){
            printf("チャンスはあと%d 回!!０から９９の整数値を入力してください:",num_play);
            scanf("%d",&answer);
            num_play--;
      //不正入力対策
            if(answer>99 || answer<0){
                  printf("不正入力値です。\n");
                  penalty++;
                  switch(penalty){
                  case 1: printf("当てる数字は二桁までですよ。
            三桁以上かマイナス値を入力してはいけませんよ \n"); break;
                  case 2: printf("同じミスを繰り返してはいけません。
            入力値は０から９９までです。\n"); break;
                  case 3: printf("プログラムが怒って
              言 う こ と を 聞 か な く な り ま し た 。冗 談 で す。( 笑 )
\n"); printf("\n");exit(ERROR); break;
                  }
       }
            //生成値と入力値の比較
            if(output_num == answer){
                  printf("正解!!おめでとう！\n");
                  break;
            }
            if(output_num > answer){
                  if(output_num / 10 > answer / 10){
                        printf("\n十の位を大きくしてみよう！\n");
                  }
                  else{
                        printf("\n一の位が違うよ\n");
                  }
                  if(output_num - answer<5){
                        printf("\n惜しい！　答えは入力値＋０~5\n");
                  }
            }
            if(output_num < answer){
                  if(answer / 10 > output_num / 10){
                        printf("\n十の位を小さくしてみよう!\n");
                  }
```

```
                else{
                    printf("\n一の位が違うよ\n");
                }
                if(answer - output_num < 5){
                    printf("\n惜しい！　答えは　入力値ー0~5\n");
                }
            }
        }
        printf("答えは%dです。\n",output_num);//答えの表示
        return 0;
}
```

4.9 推奨実行環境

Linux,Windowsともに実行できるはずです。Linuxでは端末上、Windowsはコマンドプロンプトで実行してください。ただし、Windows環境下ではバックスラッシュを円マークに置き換えてください。一応、テンキーを数字入力に用いるのがのぞましいです。統合開発環境としてEclipseを使うと楽にビルドできるはずです。

4.10 注意

文字、文字列入力に対し強制終了がかかります。この辺の対策は実装していないので、入力はテンキーが望ましいと上で示しました。文字、文字列を入力しても正解とはなりません。

4.11 謝辞

本プログラムをテストプレイし、貴重な意見を提供して頂いたコンピュータ部の矢倉君、北村君、アドバイスを下さった小西先輩、松永君、北出先輩、Limeに携わるすべての部員の皆様、Limeを手にとって下さった皆さんにこの場を借りてお礼申し上げます。

5 暗号化

情報工学課程 1 回生 矢倉　知幸

5.1 はじめに

今回私は可能な範囲で何が作れるかと思い、暗号化のプログラムを作成しました。クラブでの勉強会を参考にしつつ自分で考えたものなので、ソースコードが長くごちゃごちゃしていますが、最後まで読んでいただければ幸いです。

今回の暗号化プログラムは暗号化（encode）と暗号化した文章の解読（decode）の 2 つを行うことが出来ます。また文字コードは ASCII のコードを参考にしています。

5.2 ソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

#define STEP 4

void adjust(char word[], int number, int total);
void swap(char word[], int number, int x);

main(void){
  srand((unsigned int)time(NULL)); rand();
  int i, step, order, under, over = 1;
  int change[STEP], count = 0, passcode;
  char sentence[101] = {0};
```

ソースコードは上から順に区切って説明していきます。またプログラム中で登場する変数や文章は全て英語で表しています。adjust 関数、swap 関数は「変換実行」で使用します。STEP は「変換実行」の工程の数とパスコードの桁に関係しており、今回は 4 としています。

5.3 変換方法選択

```
printf("Choose encode or decode.\n");
printf("Encode...1/Decode...2 :");
scanf("%d",&order);
if(order != 1 && order != 2){
  printf("Error.\n"); return 0;
}
```

ここでは暗号化を行うか解読を行うかを選択します。order はこの先暗号化か解読かの識別に使います。指定された数字以外の数字を入力した場合はエラーを表示してプログラムを終了します。

5.4 文章入力

```
printf("Please don't use space_key in sentence.\n");
printf("Instead of space_key,please use under_bar.\n");
if(order == 1){
  printf("Input sentence less than 100 words.\n");
}
else{
  printf("Input code less than 100 words.\n");
}
fflush(stdout);
scanf("%100s",sentence);
for(i=0; sentence[i] != '\0'; i++){
  if(sentence[i] < '!' || sentence[i] > '~'){
    printf("Error,include unavailable word.\n");
    return 0;
  }
  count++;
}
```

文章入力の注意事項を出力後、文章を読み取ります。%100s とすることで、100 文字目までを読み取ることが出来ます。最後に入力した文字で使用できない文字がある場合はエラーを表示してプログラムを終了します。日本語は入力できませんので注意してください。count には文中の文字数が入ります。

5.5 パスコード

```
for(i=0; i<STEP; i++){ over *= 10; }
under = over / 10;
if(order == 1){
  for(i=0; i<STEP; i++){
    change[i] = rand() % 10;
    if(change[i] == 0){ i--; }
  }
}
else{
  printf("Input passcode:");
  fflush(stdout);
  scanf("%d",&passcode);
  if(passcode < under || passcode >= over){
    printf("Error,no this passcode.\n");
    return 0;
  }
  for(i=STEP - 1; i >= 0; i--){
    change[i] = passcode % 10;
    passcode /= 10;
    if(change[i] == 0){
      printf("Error,no this passcode.\n");
      return 0;
    }
  }
}
printf("\n");
```

このプログラムでいうパスコードとは次の「変換実行」に用いる数字です。今回のプログラムでは暗号化する場合は 1〜9 のランダムな 4 つの数字でパスコードを生成し、解読する場合は 4 桁のパスコードを入力します。誤ったパスコードを入力してしまうと誤った解読を行い、原文に戻りません。under と over はそれぞれ STEP の値によってパスコードの下限と上限を定めます。最後に存在しないパスコードが入力された場合はエラーを表示してプログラムを終了します。

5.6 変換実行... 上

```
step = (order == 1)? 0: STEP;
while(step >= 0 && step <= STEP){
```

```
switch(step){
  case 0: for(i=0; i < count; i++){
              if(order == 1){
                if(sentence[i] + ((change[step] * 3) % (i + 1)) > '~'){
                  adjust(sentence, i, sentence[i]
                                          + ((change[step] * 3) % (i + 1)));
                }
                else{ sentence[i] += ((change[step] * 3) % (i + 1)); }
              }
              else{
                if(sentence[i] - ((change[step] * 3) % (i + 1)) < '!'){
                  adjust(sentence, i, sentence[i]
                                          - ((change[step] * 3) % (i + 1)));
                }
                else{ sentence[i] -= ((change[step] * 3) % (i + 1)); }
              }
            }
            if(order == 1){ step++; }
            else{ step--; } break;
  case 1: if(order == 1){
              for(i=0; i + change[step] < count; i += 2){
                swap(sentence, i, change[step]);
              }
            }
            else{
              for(i = ((count - change[step]) % 2 == 1)
                      ? (count-1): (count-2);i - change[step] >= 0;i -= 2){
                swap(sentence, i, -change[step]);
              }
            }
            if(order == 1){ step++; }
            else{ step--; } break;
```

変換部分は長いので３つに分けます。今回の場合、変換は５つの工程（step）に分けて行われます。暗号化は「０→１→２→３→STEP」の順番で行い、解読はその逆の順番で行われます。adjust 関数は char 型の大きさによる不具合を防ぐためのものです。swap 関数は文章中の文字の順番を変えるために使用しています。

5.7 変換実行... 中

```
case 2: for(i=0; i<count; i++){
            if(i % change[step]){
              if(order == 1){
                if(sentence[i] - i < '!'){
                  adjust(sentence, i, sentence[i] - i % ('~' - '!' - 1));
                }
                else{ sentence[i] -= i; }
              }
              else{
                if(sentence[i] + i > '~'){
                  adjust(sentence, i, sentence[i] + i % ('~' - '!' - 1));
                }
                else{ sentence[i] += i; }
              }
            }
            if(order == 1){ step++; }
            else{ step--; } break;
case 3: for(i=0; i<count; i++){
            if(order == 1){
              if(sentence[i] + count % change[step] > '~'){
                adjust(sentence, i, sentence[i] + count % change[step]);
              }
              else{ sentence[i] += (count % change[step]); }
            }
            else{
              if(sentence[i] - count % change[step] < '!'){
                adjust(sentence, i, sentence[i] - count % change[step]);
              }
              else{ sentence[i] -= (count % change[step]); }
            }
            if(order == 1){ step++; }
            else{ step--; } break;
```

暗号化と解読を一つの switch 文に無理矢理詰め込んでいるので、コードが複雑になっています。特に加算と減算を暗号化と解読で逆転させるために if 文がかなり多くなっています。また STEP を変更して工程を増やす場合は case3 の続きに追加してください。

5.8 変換実行... 下

```
      case STEP: for(i=0; i<count; i++){
                   if(order == 1){
                     if(i % 2){
                       if(sentence[i] + change[i % STEP] > '~'){
                         adjust(sentence, i, sentence[i] + change[i % STEP]);
                       }
                       else{ sentence[i] += change[i % STEP]; }
                     }
                     else{
                       if(sentence[i] - change[i % STEP] < '!'){
                         adjust(sentence, i, sentence[i] - change[i % STEP]);
                       }
                       else{ sentence[i] -= change[i % STEP]; }
                     }
                   }
                   else{
                     if(i % 2){
                       if(sentence[i] - change[i % STEP] < '!'){
                         adjust(sentence, i, sentence[i] - change[i % STEP]);
                       }
                       else{ sentence[i] -= change[i % STEP]; }
                     }
                     else{
                       if(sentence[i] + change[i % STEP] > '~'){
                         adjust(sentence, i, sentence[i] + change[i % STEP]);
                       }
                       else{ sentence[i] += change[i % STEP]; }
                     }
                   }
                 }
                 if(order == 1){ step++; }
                 else{ step--; } break;
    }
  }
```

これで変換の工程は終了です。STEP の値を変える場合、この工程が暗号化する際に一番最後に実行されるようにしてください。

5.9 結果の出力

```
  if(order == 1){ printf("Enode...\n\n"); }
  else{ printf("Decode...\n\n"); }
  for(i=0; sentence[i] != '\0'; i++){
    printf("%c",sentence[i]);
    if(i % 20 == 19){ printf("\n"); }
  }
  if(count % 20 != 0){ printf("\n"); }
  if(order == 1){
    printf("\nPasscode:");
    for(i=0; i<STEP; i++){ printf("%d",change[i]); }
    printf("\n");
  }
  return 0;
}
```

変換結果は２０字置きに改行するようにしています。また暗号化を行った場合はパスコードを表示します。

5.10 使用した関数

```
void adjust(char word[], int number ,int total){
  if(total > '~'){
    word[number] = ('!' - 1) + (total - '~');
  }
  if(total < '!'){
    word[number] = ('~' + 1) - ('!' - total);
  }
}

void swap(char word[], int number, int x){
  char temp;
  temp = word[number];
  word[number] = word[number + x];
  word[number + x] = temp;
}
```

swap 関数内の、x、は適切な英単語が浮かばなかったため一時的に置いたものです。

5.11 使用出来る文字

使用することの可能な文字の一覧を記載しておきますので、遊ぶ際に参考にしてください。他の文字コードで行った場合に同じように動くかは保証できません。

表 5.1: 使用可能な文字

!	+	5	?	I	S	]	g	q	{
”	,	6	@	J	T	^	h	r	|
#	-	7	A	K	U	_	i	s	}
$	.	8	B	L	V	‘	j	t	~
%	/	9	C	M	W	a	k	u	
&	0	:	D	N	X	b	l	v	
’	1	;	E	O	Y	c	m	w	
(	2	<	F	P	Z	d	n	x	
)	3	=	G	Q	[	e	o	y	
*	4	>	H	R	\	f	p	z	

5.12 最後に

いかがでしたか？私は暗号化に関する知識がほとんど無かったので上手に暗号化できているか心配です。一番時間のかかった部分は変換部分でした。特に暗号化するので変換後の出力を見てもすぐには異常があるかを判別できないのでバグの修正にてこずりました。僕の検証した範囲ではバグは修正しましたが、まだ他にもあるかもしれませんのでご注意ください。

6 HIT & BLOW

情報工学課程 1回生 福井　寿行

6.1 はじめに

HIT&BLOWとは隠された数字を当てるゲームです。プログラミングを始めたばかりの人でも簡単に作れるゲームです。せっかくなので楽しめるようなゲームを作ろうと思い、色々と考えてみたのですが、僕の力で作れるのはこのゲームくらいだったのでHIT&BLOWをつくることにしました

6.2 ソースコード

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>


//数字の入力と有効な値かをチェック
void input(int p[]) {
	int flag, N, k;
	char buffer[256];

	while (1) {
		fgets(buffer, 256, stdin);
		flag = sscanf(buffer, "%d", &N);

		//文字入力がされていないかチェック
		if (flag == 0) {
			printf("数字以外は入力しないで下さい\n再入力してください\n");
			continue;
		}

		//有効範囲の数字かチェック
```

```
            flag = (N >= 123 && N <= 9999);
            if (flag == 0) {
                  printf("四桁の数字以外は入力しないでください\n 再入力してくだ
さい\n");
                  continue;
            }

            //四桁の数字を一ケタずつ格納する
            for (k = 0; k < 4; k++) {
                  p[3 - k] = N % 10;
                  N = N / 10;
            }

            //同じ数字が含まれていないかチェック
            flag = (p[0] != p[1] && p[0] != p[2] && p[0] != p[3] && p[1] != p[2] &&
  p[1] != p[3] && p[2] != p[3]);
            if (flag == 0) {
                  printf("同じ数字が含まれています\n 再入力してください\n");
                  continue;
            }

            break;
      }
      return;
}

int main(void) {
      int ans[4], num[4];
      int count = 0, hit = 0, blow = 0;
      int i, j;
      srand((unsigned)time(NULL));

      //答えとなる数字の生成
      //数字の重複がある場合は再生成
      while (1) {
            for (i = 0; i<4; i++) { ans[i] = rand() % 10; }
            if (ans[0] != ans[1] && ans[0] != ans[2] && ans[0] != ans[3] &&
  ans[1] != ans[2] && ans[1] != ans[3] && ans[2] != ans[3]) { break; }
      }
```

```
	printf(
	"数字当てゲーム\n"
	"隠された四桁の数字を当てよう\n"
	"○…場所も数字も一致　△…数字だけ一致\n"
	"※同じ数字が二個以上含まれることはありません\n");
	while (1) {
		printf("\n 四桁の数字を入力してください\n");
		count++;
		hit = 0;blow = 0;
		input(num) ;

		//hit と blow をカウント
		for (i = 0; i<4; i++) {
			if (num[i] == ans[i]) { hit++; }
		}
		for (i = 0; i<4; i++) {
			for (j = 0; j<4; j++) {
				if (num[i] == ans[j]) { blow++; }
			}
		}
		blow -= hit;

		//結果を表示
		if (hit == 4) {
			printf("正解です !! 所要回数%d 回\n", count);
			break;
		}
		else { printf("○…%d　△…%d\n", hit, blow); }
	}
	return 0;
}
```

6.3 コードについて

文字や有効範囲外の数字に対しては対策をしたのですが、数字＋文字の混合文などを入力すると弾けずに予期せぬ動作をすることがあります。多重ループなどは極力減らそうと思

い、関数などを使って減らしましたが、それでも不格好なコードになってしまっています

6.4 最後に

初めは、コンソールアプリでは視覚的にはあまり面白くないので、Dx ライブラリを使ってウィンドウアプリを作ろうと思ったのですが時間的にも実力的にも不足していたので作ることができませんでしたので、今年のところはこのような簡単なゲームですがご容赦ください。

7 n次正方行列の逆行列の計算プログラム

情報工学課程 1 回生 北村　馨

7.1 はじめに

僕たち一回生は前期の線形代数学の授業で、多次元の正方行列の逆行列の計算方法を習いました。しかしながらこの計算は手書きで行うには非常に面倒で、多次元の正方行列の逆行列を計算する際にはケアレスミスが非常に多くなります。そこで僕は今持っている c 言語の知識を使って任意の正方行列の逆行列を計算するためのプログラムを書こうと考えました。

7.2 計算方法

逆行列の計算には掃き出し法を用います。掃き出し法は行列の行基本変形と呼ばれる変形を繰り返すことによって計算する方法です。
以下、掃き出し法による逆行列計算の流れを具体例とともに説明する。

(i) 対象の行列 A の右に単位行列を並べた行列 B を定義する

$$A=\begin{bmatrix}2&-1&0\\1&0&-1\\2&-1&-1\end{bmatrix},B=[AE]=\begin{bmatrix}2&-1&0&1&0&0\\1&0&-1&0&1&0\\2&-1&-1&0&0&1\end{bmatrix}$$

(ii) 行列 B の第 1 行を何倍かして（例においては 1/2 倍して）、行列 B の主成分を 1 する。

$$B\Rightarrow\begin{bmatrix}1&-\frac{1}{2}&0&\frac{1}{2}&0&0\\1&0&-1&0&1&0\\2&-1&-1&0&0&-1\end{bmatrix}$$

(iii) 第 1 行の何倍かをほかの行（例においては第 2、3 行）に加えて、第 1 行の主成分を含む列（例においては第 1 列）のほかの成分を 0 にする。

$$\begin{bmatrix}1&-\frac{1}{2}&0&\frac{1}{2}&0&0\\1&0&-1&0&1&0\\2&-1&-1&0&0&-1\end{bmatrix}\Rightarrow\begin{bmatrix}1&-\frac{1}{2}&0&\frac{1}{2}&0&0\\0&\frac{1}{2}&-1&-\frac{1}{2}&1&0\\0&0&-1&-1&0&1\end{bmatrix}$$

(iv) 第 2,3,.. 行について (ii)(iii) を繰り返し行う。

$$\begin{bmatrix} 1 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & -1 & -\frac{1}{2} & 1 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & -2 & -1 & 2 & 0 \\ 0 & 0 & -1 & -1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & -1 \\ 0 & 1 & 0 & 1 & 2 & -2 \\ 0 & 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

行列 B に (ii)〜(iv) の操作を行った後の行列 B' が、B' = [EC]　の形になっていれば、行列 A は逆行列 C を持ちます。逆を言えば、B = [EC] の形になっていないならば行列 A は逆行列を持たないということになります。

今回書いたコードにおいては、入力された行列次第によってはステップ (ii) において 0 で割る可能性が生じるので、二つの行を入れ替える操作が必要な場合が存在します。また、第 i 行列より下の行列の、左から i 番目の成分がすべて 0 だった場合は、B' = [EC] の形にならないので、ここで操作を終了します。

以上の計算方法を基にして書いたコードが次になります。

7.3 ソースコード

```
#include<stdio.h>
#include<stdlib.h>


//２つの値を入れ替える関数
void swap(double *x,double *y){
    double temp;
    temp = *x;
    *x = *y;
    *y = temp;
}


int main(void){

    //行列の次数を入力
    int n;
    printf("次数を指定してください（１以上の自然数）:");
    scanf("%d",&n);
```

```
//次数が負の数だった場合の処理
if(n<=0){
    printf("１以上の自然数を入力してください！\n");
    return -1;
}

//行列計算のためのメモリを確保
double *matrix = malloc(sizeof(double)*n*n*2);

int m=n*2;//簡単のため、行列 B の列数 m を定義

//行列の各要素を入力と同時に行列 B を定義
printf("各要素を整数で入力してください\n");
int i,j;
for(i=0;i<n;i++){
    for(j=0;j<n;j++){
        printf("matrix[%d][%d]=",i+1,j+1);
        scanf("%lf",&matrix[m*i+j]);
        matrix[m*i+j+n] = 0;
    }
}
for(i=0;i<n;i++){
    matrix[i*m+i+n] = 1;
}

//入力した行列の確認
printf("入力された行列は、\n");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
        printf("%.3f\t",matrix[i*m+j]);
        }
    printf("\n");
    }

int p,q,r,s,t,u;
double head1,head2;

for(p=0;p<n;p++){
```

```
        //ステップ(ii)において0で割るのを防ぐための行を入れ替える処理
        for(u=p;u<n;u++){
            if(matrix[p*m+p] == 0){
                for(t=0;t<m;t++){
                    swap(&matrix[p*m+t],&matrix[u*m+t]);
                }
            }
        }

        //逆行列が存在しない場合の処理
        if(matrix[p*m+p] == 0){
            printf("この行列の逆行列は存在しない。\n");
            return 0;
        }

        //ステップ(ii)
        head1 = matrix[p*m+p];
        for(q=p;q<m;q++){
            matrix[p*m+q] = matrix[p*m+q]/head1;
        }

        //ステップ(iii)
        for(r=0;r<n;r++){
            if(r != p){
                head2 = matrix[r*m+p];
                for(s=p;s<m;s++){
                    matrix[r*m+s] = matrix[r*m+s]-matrix[p*m+s]*head2;
                }
            }
        }
    }

        //逆行列の出力
        printf("この行列の逆行列は、\n");
        for(i=0;i<n;i++){
            for(j=0;j<n;j++){
            printf("%.3f\t",matrix[i*m+j+n]);
            }
```

```
            printf("\n");
        }

    free(matrix);

    return 0;
}
```

7.4 最後に

この文章においては、掃き出し法による逆行列の計算についての数学的背景の説明は割愛させていただきました。申し訳ないですが、気になった方は『入門線形代数』（著者：三宅敏明氏）等の参考書を手に取っていただけると幸いです。

7.5 参考文献

三宅敏明 (1991) 『入門線形代数』 培風館.

8 Arduinoを使ったゲーム用入力デバイス

情報工学課程 2 回生 川崎真

8.1 はじめに

Arduinoとは、Amtel AVRマイコンなどを搭載したワンボードマイコンと統合開発環境の総称である。Arduinoはハードウェア・ソフトウェアの両方について、設計・ソースコードなどの主要情報が公開されている。このため、互換機の販売が行われており、互換機の自作も可能である。

今回は、純正のArduinoボードのArduino LeonardoのUSB通信機能を用いて、ゲーム用のアナログ入力デバイスを制作した。

8.2 ハードウェア

今回の入力デバイスは、3つのスライダーによるアナログ入力と3つのボタンを備えている。

ハードウェアは、片側を半田付けしたジャンパワイヤで可変抵抗・押スイッチをArduino Leonardoに接続したものである。

スライドボリューム3個はそれぞれArduino Leonardoの$+5V$出力ピン、アナログ入力ピン、グラウンドピンに接続されている。これによって、抵抗分圧をArduino Leonardoで読み取る事ができる。

スイッチ3個はデジタル入出力ピンとグラウンドピンに接続されている。後述する通り、マイコン側でプルアップしており、押すと電圧が下がるようになっている。

回路は図8.1の通りである。

筐体はアルミ製ケースを用い、ドリルを用いて穴を開けた。外観は図8.2を参照。

8.3 ソフトウェア

今回のプログラムは、USB の HID デバイスクラスとして動作させるライブラリを用いる。プログラムは読みこんだスライドボリュームとボタンの値を関数で設定するだけで、ライブラリが通信のコントロールなどを全て行う。

今回用いたスケッチ（Arduino でのプロジェクトのこと）は、パッチを Arduino の開発環境のライブラリに当てて、アナログ 7 軸・ボタン 32 個に対応している。

ライブラリは以下からダウンロードできる。

https://github.com/Cryocrat/LeonardoJoystick

```
JoyState_t joySt;    // Joystick state structure

void setup()
{
  pinMode(0, INPUT_PULLUP);
  pinMode(2, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);

  joySt.xAxis = 0;
  joySt.yAxis = 0;
  joySt.zAxis = 0;
  joySt.xRotAxis = 0;
  joySt.yRotAxis = 0;
  joySt.zRotAxis = 0;
  joySt.throttle = 0;
  joySt.rudder = 0;
  joySt.hatSw1 = 0;
  joySt.hatSw2 = 0;
  joySt.buttons = 0;

}

void loop()
{
  // Set Axes value
  joySt.xAxis = map(analogRead(0), 0, 1023, 0, 255);
  joySt.yAxis = map(analogRead(2), 0, 1023, 0, 255);
  joySt.throttle = map(analogRead(4), 0, 1023, 0, 255);
```

```
  // Set button value
  joySt.buttons = 0;
  joySt.buttons |= digitalRead(0) ? 0 : 1;
  joySt.buttons |= digitalRead(2) ? 0 : 2;
  joySt.buttons |= digitalRead(4) ? 0 : 4;

  // Call Joystick.move
  Joystick.setState(&joySt);
}
```

setup関数はハードウェアの接続時の初期化処理である。ここでは、デジタル入出力ピンのプルアップ入力モードの設定と、送信データの変数の初期化を行っている。

loop関数は初期化後に呼び出される無限ループのルーチンで、スライドボリュームとボタンからの入力をUSB接続ライブラリに渡している。前半は、スライドボリュームからの電圧の値を512段階に変換し、送信データにセットしている。後半は、ボタンの値を送信データにセットしている。プルアップ方式のため、ボタンが押された時にdigitalReadの返り値が0になることに注意。

8.4 おわりに

Arduinoは簡単なデバイスを自作する時に便利である。

PC接続のあるデバイスも、スタンドアロンのデバイスも自作できる。また、今回は用いなかったが、温度センサ・SDカードソケット・無線接続カードなどの各種拡張ハードウェアとそのライブラリも豊富である。

また、ハードウェア仕様が公開されているので、自作の電子工作作品を互換機にすることによってArduinoのインフラを利用することもできる。

私は、プログラムを書ける人にとってArduinoは魅力的な電子工作パーツであると思う。

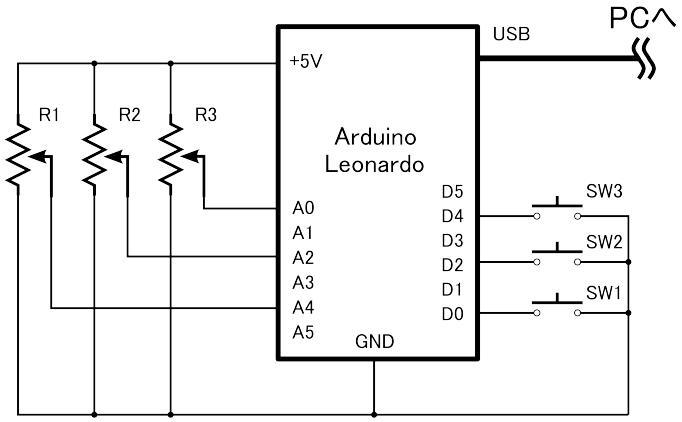

図 8.1: 回路図

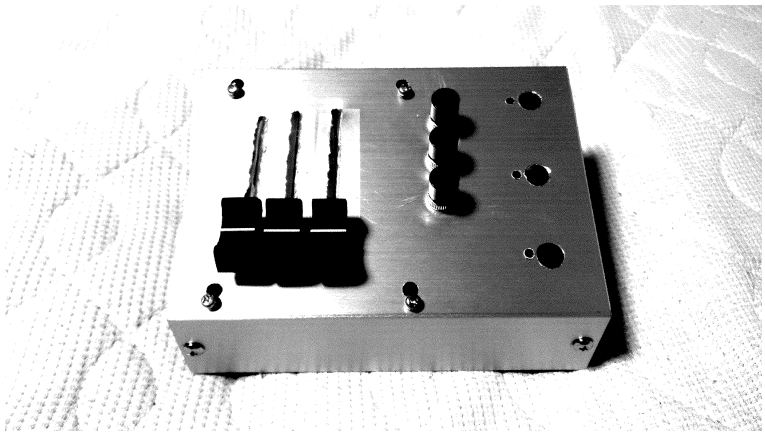

図 8.2: 筐体の外観

編集後記

編集担当の家原です. 今回で 52 冊目の Lime になります. 松ヶ崎祭自体が開催から危ぶまれるようなこともあり、制作から編集までなかなかハードな日程になっていましたが、今年も無事発行することができました. 今年は合宿がなく、作る人もすくなかったため記事が少なめですが、楽しんでもらえると幸いです. 私個人としては、レベルの高い記事が多いと感じ、編集していてとても面白かったです. 最後にこの冊子を手にとっていただき、さらに最後まで目を通してもらいありがとうございました.

平成 27 年 10 月 27 日
編集担当　家原　瞭

Lime Vol.52

平成 27 年 11 月 15 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/