C 言語復習トレーニング@グラフィックス演習 2016　担当　安永卓生 2016/09/28

本課題は、グラフィックス演習に向けて、C 言語の復習及び複数ファイルのコンパイルの予習、及び、実際の演習のためのライブラリ（関数の作成）のために実施するものである。本課題の提出及び内容を成績の最終評価に２０％の割合で加味する。

2016 年 12 月 6 日 13 時までにすべての課題を終了し、ムードル上に提出すること。

※ 提出期限：2016 年 12 月 6 日 13 時

---

◎　課題の実施に関しては次の順序で実施せよ。

１．　ソースコードの管理や共同開発の基礎を学ぶために Git(GitBucket)というツール（補遺 1 参照）を利用する。

(ア) 学科の GitBucket サーバに新たなプロジェクト（レポジトリ）を作成する。

① http://edu-git.bio.kyutech.ac.jp/gitbucket/ にログインする。アカウントは、端末の ID(以下、UserID として記述する)、パスワードと共通である。

② New repository（新しいリモート・レポジトリ）を、practiceGraphics という名前で、private repository（登録者だけが使用するレポジトリ）として作成する。

③ Settings/Collaborators に安永(UserID: tacyas)を加える。

※ 今後、新たなソース管理（グラフィックス演習中に個人、及び、チームでの開発を行う）では、新たなレポジトリを作製する。

※ グループで作成する際には、リーダが一名この作業を行い、Collaborators として班員を加える。その際も安永（UserID: tacyas）を加えること。

２．　SSH キーを設定する

(ア) ssh-keygen コマンドを使って、パブリックキーを作製する

（Git Bash を立ち上げる）

① ＄ssh-keygen

途中で聞かれるキーを保存するファイルは、そのまま Enter キーを押す。

途中で、パスフレーズを聞かれるので、適当なパスフレーズを入力。パスワードとは異なるものが望ましい。

② 学科の GitBucket サーバーにパブリックキーを設定する。

$ cat .ssh/id_rsa.pub

この画面に出た文字を全てコピー（マウスで選択して、ウィンドウの上部バーの編集・コピー）

GitBucket のウィンドウの右上（▼）Account Settings を押し、その後、SSH Keys を押す。

Add an SSH key の Key の上でペースト。Title はそれぞれのホスト毎に分かり易い名前をつける。

※この作業は、異なる環境(ホームディレクトリが異なる)毎に実施する必要がある。

3．　開発を行うディレクトリに移動し、ローカルレポジトリとして、クローンを作成する。

$ git clone git clone ssh://git@edu-git.bio.kyutech.ac.jp:29418/UserID/practiceGraphics.git

※ userID の所はそれぞれのユーザー毎に異なる

※ practiceGraphics というディレクトリができあがっている。この後、以下に指定している全てのプログラムは、この practiceGraphics というディレクトリの下に作成すること。

注1）　git の使い方については、補遺 1 を参照すること。

利用するコマンドは最低限次の通り

① リモートレポジトリからクローンを作製する

$ git clone xxxxxxxxxxxxxxxxx

② インデックスに登録

$ git add .

③ ローカルレポジトリに登録

$ git commit

④ リモートレポジトリに登録

$ git push

⑤ リモートレポジトリからローカルレポジトリに登録

$ git fetch

⑥ ワーキングツリーとマージ

$ git merge

- ローカルレポジトリにワーキングツリーの内容が一旦登録されていることが前提

注 2）上記のコマンドを利用することにより、リモート・レポジトリの最新版をいつでも手に入れることができる。例え、ローカル・レポジトリを消しても、最新版をプッシュ(アップロード)しておけば、git clone コマンドを使って復元することができる。

注 3）異なる環境に用意する場合も同様である。

4．　最終的に、一つのファイルにアーカイブし、moodle のグラフィクス演習 2016 の指定した提出場所にアップロードせよ。

practiceGraphics のディレクトリの親ディレクトリにおいて、以下のプログラムを実行し、作成した practice-xxxxxx.tgz ファイルをアップロードせよ。xxxxxx は、学生番号とせよ。

$ tar cvzf practice-xxxxxx.tgz practiceGraphics

加えて、自分自身の技術レベル、特に、開発能力をチェックし、フィードバックすることは開発能力を高めるために必須である。そこで、開発時間を想定する能力をチェックすることにする。

自己点検書として、ムードル上にアップした下記のような様式（ワード形式のファイルが用意されている）にしたがって、全ての作業に関して、開発を始める前に、どの位の時間を要すると予め考えたか、また、実際には、開発にどのくらい時間を要したかを記し、そのファイルを作成記録として、同様にムードル上にアップせよ。ファイルフォーマットは、オープンオフィスもしくは、ワードファイル形式とせよ。

---

予習課題の自己点検書

学生番号　　氏名

| 課題番号 | 開発に想定した時間（A[分]） | 開発に要した時間（B[分]） | B ÷ A × 100 (%) | 開発コード行数 | 想定とずれた理由 |
|---|---|---|---|---|---|
| 課題 I | | | | | |
| 課題 II | | | | | |
| 課題 III | | | | | |
| 課題 IV | | | | | |
| 発展課題 I | | | | | |
| 課題 V | | | | | |
| 発展課題 II | | | | | |

予習課題の実施に関して、自分自身のスキルに関する評価を下記に記せ。

①□　　3年前期までに学んだ内容を全て理解しているとしたときを１０とすると、自分自身のスキルはいくらか。○をつけよ。

0　1　2　3　4　5　6　7　8　9　１０

②　問題を感じた場合、どこの学びが特に弱いと感じたか、科目名及び技術スキルを記述せよ。

科目名　　技術スキル

---

練習 1 （9 月 28 日）：実際に Git を使って、ソースコードの開発を行ってみよう。

clone で作製されたディレクトリにおいて、下記の出力ができる hello.c を作製せよ。
Cygwin コンソールを使えば、emacs/cc が利用できる。

```
$ hello
Hello C-World
```

１．完成したら、プログラムを add/commit/push し、リモートレポジトリに登録せよ。
２．GitBucket のサービス上で、確かにファイルが登録されていることを確認せよ、
３．異なるディレクトリにクローンを作製する。
　　ホームディレクトリ（/cygdrive/z/）の直下に、other を作製し、そのディレクトリの中にクローンを作製せよ。
４．新しくつくったワーキングツリーにある hello.c を変更して
　　Hello C-World
　　How are you doing ?
と出力されるように変更せよ。加えて、Readme.txt というファイルを作成して、その中に
practice1: hello.c
と記述せよ。

５．　完成したら、プログラムを add/commit/push し、リモートレポジトリに登録せよ。

６．　元のディレクトリ側に移動せよ。WhatToDo.txt というファイルを作成して、その中に
practice1: git exercise
と記述せよ。その後、add/commit を実行せよ。

７．　リモートレポジトリから最新のファイルを fetch/merge せよ。

８．　４で行った変更が 6 に反映していることを確認せよ。

９．　add/commit/push を実行せよ。

１０．other 以下のディレクトリに移動して、最新版に更新し、そのように変更されていることを確認せよ。

課題 I（ファイルの取り扱いの復習）　ex1.c というファイルを作成し、ex1 という実行形式をコンパイルして作成する。コンパイルの仕方は以下のようになる。

```
$ cc  -c ex1.c                  # ex1.c をコンパイルし、ex1.o を作成。
$ cc ex1.o  -o ex1 -lm          # ex1.o を必要なライブラリとリンクする。
                                # -lm は/lib/libm.a をリンクする意味。
                                # m は math の意味で数学関数を使う場合に必要である。
                                # libxxx.a とリンクする場合には、-lxxx とする。
```

○プログラム ex1 の仕様

$ ./ex1  ファイル名  startx starty endx endy

とすると、「ファイル名」で指定したファイルを開き、そこに(startx, starty)から(endx, endy)に線をひく main 関数を含む ex1.c を作成せよ。startx, starty, endx, endy のこれらの 4 つの数字は浮動小数点表示の実数として入力するものとする。

具体的には、指定されたファイルに

```
%! PS-Adobe-3.0
startx starty moveto
endx endy lineto
stroke
showpage
```

と、出力できればよい。

※　注意：実行時の引数は main 関数の引数となる。通常、main 関数は、

```
int main(int argc, char* argv[])
```

と宣言する。このとき、argc がコマンドも含めた引数の数、argv が引数の内容を文字列で表現している。今回の場合、　　argc=6　であり、

```
argv[0]          # ./ex1　が文字列として格納
argv[1]          #ファイル名が文字列として格納
argv[2]          # startx で指定した数字が文字列として格納。
argv[3]          # starty で指定した数字が文字列として格納。
argv[4]          # endx で指定した数字が文字列として格納。
argv[5]          # endy で指定した数字が文字列として格納。
```

が格納されてる。argv[2]～avg[5]は、内部では文字列ではなく、浮動小数点として取り扱いたいので、atof 関数を用いて変換するようにすること。

※　ポストスクリプトファイル：　startx, starty endx, endy は実際には、特定の数値を表すことになる。単位は、ポイントと呼ばれ、72 ポイントが 1 インチ（25.4mm）になる。このファイルの形式をポストスクリプトと呼ぶが、それらの詳細な意味は、補遺 2 のポストスクリプトに関する情報を参考にせよ。

課題 II　（関数の復習）下記に示すように、bondDraw 関数を作成し、main 関数から呼び出すことにより、課題 I と同じ動作をする main 関数を含む ex2.c を作成せよ。

線を描く関数

```
void
bondDraw(FILE* fpt, float startx, float starty, float endx, float endy)
```

は、bond.c と名付けられたファイルに記述せよ。この関数を呼び出すと、第一引数のファイルポインタで指定されたファイルに、

```
startx starty moveto
endx endy lineto
stroke
```

が書き出されるものとする。返り値は必要ない。ファイルに書き出された startx 等の変数は、fprintf 関数を利用し、書式%f を用いて出力された浮動小数点である。

この関数のプロトタイプ宣言（型宣言）は、PDB.h の中に記述する。
ex2.c 及び bond.c において、挿入(include)することで、関数の型の確認ができる。
PDB.h は、

---

```
#ifndef PDB_H  //PDB_H が定義がされていない時 endif までのコードが意味がある
#define PDB_H  //PDB_H を定義する
  この間に関数宣言や構造体宣言などを書き込むこと。
#endif                  // #ifndef や#ifdef を閉じる。
```

---

とせよ。
このインクルードファイルは、ex2.c, bond.c のそれぞれで、

```
#include “PDB.h”
```

として、挿入できる。

今回の課題では、ファイルが分かれているために、分割コンパイル・リンクが必要である。具体的には以下のようになる。

```
$ cc –c ex2.c
$ cc –c bond.c
$ cc ex2.o bond.o –o ex2  -lm
```

課題Ⅲ　（構造体の復習）　下記に示す関数 bondDraw2 を使って、課題 I と同じ動きをするプログラム ex3 を作成せよ。

ex3 の main 関数は、ex3.c に記述し、実行形式 ex3 を作成せよ。

課題 II の線を書く関数 bondDraw を利用して、構造体が使える関数

```
void bondDraw2(FILE* fpt, Bond l)
```

を bond2.c の中に、関数 bondDraw2 から、関数 bondDraw を呼び出す形で実装せよ。また、下記に示した構造体 Bond を PDB.h の中に作成し、bond2.c でこの構造体が使用できるように#include 文を用いて挿入せよ。

PDB.h での構造体の宣言

```
#ifndef PDB_H
#define PDB_H
typedef struct Atom Atom;          // 構造体の型宣言
struct Atom {
    float x;
    float y;
    float z; // 今回は用いないが、将来の 3 D座標のために宣言しておく
};
typedef struct Bond Bond;
struct Bond {
    Atom start;
    Atom end;
};

extern void bondDraw(FILE* fpt, float startx, float starty, float endx, float
endy);
extern void bondDraw2(FILE* fpt, Bond l);

#endif /* PDB_H */
```

課題Ⅳ（配列の復習）補遺 3 の PDB ファイルの説明を参考にして、ex4 を、下記の手順に従って開発せよ。

１．　構造体の宣言

下記の構造体 arrayPDB を PDB.h の中に定義し、

```
typedef struct arrayPDB arrayPDB;
struct arrayPDB {
    int    numAtom; /* ATOM で指定された原子の個数 */
    int     numCAAtom;      /* α炭素の数（点の数） */
    Atom    *CA;            /* α炭素の点の座標(x, y, z) */
};
```

とせよ。

２．　関数 pdbRead の開発(pdbRead.c)

ファイル pdbRead.c の中に、関数 int arrayPDBRead(FILE* fpt, arrayPDB *pdb)を作成せよ。この関数は、ファイルポインタで指定されたファイル（ＰＤＢフォーマット）から、構造体 arrayPDB に変数を読み込む関数である。その動作の仕様は次のように設定する。

（１） 一度、ファイルを最初から最後まで読み込み、元素の個数 numAtom, α炭素の数 numCAAtom をカウントする。一行一行の読み込みには、fgets 関数を利用する。実際には、先頭の 6 文字が”ATOM　“で指定された行（レコード）の行数、及び、その中で、AtomName(12－１５文字目)が” CA “となっている行の行数をカウントする。。

（２） α炭素の数にしたがって、メンバ変数 CA を配列として用いるために、malloc 関数もしくは、calloc 関数を用いて配列領域を確保せよ。

（３） fseek 関数を用いて、ファイルの先頭まで戻る。fseek 関数を用いると、ファイルの開閉をしなくても、ファイルポインタを先頭に移動できる。

（４） 改めて、ファイルの最初から最後まで読み込み、α炭素の座標をメンバ変数 CA として読み込むこと。

３．　関数 bondCADraw の開発(bondCA.c)

ファイル bondCA.c の中に、関数 int bondCADraw(FILE* fpt, arrayPDB pdb)を作成せよ。この関数は、PDB 型の変数 pdb に格納された Atom 型をもつ点を結ぶ線を描くための関数である。下記のような出力を行う。moveto/lineto の命令の行数は全体で numCAAtom である。最後に stroke 命令を実行することで線が引かれる。

```
X0 Y0  moveto
X1 Y1  lineto
```

```
…
Xn-1 Yn-1  lineto
stroke
```

４．main 関数の開発

これまで用いた arrayPDBRead/bondCADraw を用いて、タンパク質のα炭素を繋いだ画像を出力するための main 関数を ex4.c に作成せよ。

その際、

$ ex4 PDB ファイル名　PS 出力ファイル名

とすると実行できるものとせよ。

下記のように、分割コンパイル、リンクできるものとする。

```
$ cc –c bondCA.c –o bondCA.o
$ cc –c pdbRead.c –o pdbRead.o
$ cc –c ex4.c –o ex4.o
$ cc ex4.o bondCA.o pdbRead.o –o ex4 –lm
```

発展課題Ⅰ　現在のプログラムでは、画像が左下角に描かれてしまう。また、大きさも非常に小さい。自由な位置に、自由な大きさで配置できるように、二つの引数 originx, originy, scale を与え、タンパク質の原点が(originx, origin)に scale 倍されて表示されるようにせよ。

実行時の形式は、

$ ex4-1 PDB ファイル名　PS 出力ファイル名　originx　　oringy　　scale

とし、与えた原点の周りに scale 倍された画像がでるようせよ。

実装方法としは、PDB の原子の位置を平行移動し、拡大したものに変換した後に、bondCADraw を呼び出すか、もしくは、bondCADraw に、originx, originy, scale の 3 つの引数を加えた関数を実装せよ。後者が望ましい。

課題Ⅴ（リストの復習）課題 IV と同じ機能を、ex5 として、リストを用いて実現するものとする。この場合には、最初に個数を数えなくても、実現できる。

下記の構造体 PDB を PDB.h の中に定義し、

```
typedef struct recordPDB recordPDB;
struct recordPDB {
    Atom    atom;                           /*  原子の座標  */
    recordPDB*      nextAtom;               /*  次の ATOM へのポインタ  */
    recordPDB*      nextCA;         /*  次の CA へのポインタ  */
};

typedef struct PDB PDB;
struct PDB {
    int numAtom;                            /*  原子の個数  */
    int numCA;                              /*  ＣＡの個数  */
    recordPDB* top;                 /*  先頭の原子 */
    recordPDB* topCA;                       /*  先頭の CA */
    recordPDB* current;                     /*  現在の原子 */
    recordPDB* currentCA;           /*  現在の CA 原子 */
};
```

とせよ。

（１） ファイル pdbRead.c の中に、関数 int pdbRead(FILE* fpt, PDB *pdb)を作成せよ。この関数は、ファイルポインタで指定されたファイルから、構造体 PDB にデータを読み込み、格納するための関数である。

それぞれの各行（レコード）に対応して、recordPDB の構造体が対応する。これは単方向リスト構造となっており、次のように宣言されている。

| | |
|---|---|
| Atom | それぞれの原子の座標を格納する。 |
| nextAtom | 全ての場合において、次の原子を示すポインタ。<br>最後の原子には、NULL が格納されている。 |
| nextCA | その原子がα炭素である場合には、<br>次のα炭素のポインタが指定されている。<br>そうでない場合、また、最後のα炭素には NULL が格納されている。 |

また、全体を格納している PDB の構造体には、次のものが格納されている。

numAtom 全ての ATOM の個数
numCA　　全ての α 炭素の個数
top　　最初の原子のポインタ、格納前には NULL
topCA　　最初の α 炭素のポインタ、格納前には NULL
current 現在アクセス中の原子の情報。
currentCA　　現在アクセス中の α 炭素の情報

○　下記が読み込みのためのアルゴリズムである。

一行読む
RECORDNAME が"ATOM　"か、ATOMNAME が" CA "かを確認する。
"ATOM　"である場合には、numAtom を 1 増加させ、
新しい recordPDB 型の領域を malloc で確保し、その原子の座標を読み込む。
必要に応じて、nextAtom を設定する。
" CA "である場合には、numCA を 1 増加させ、topCA，nextCA の値を設定する。
最後の原子、α炭素には、nextAtom, nextCA を NULL を挿入し、次の行に移動する。

（２）上記の構造体 PDB のポインタを送ると、それを課題 IV(2)と同様に出力できる bondCADraw2(FILE* fpt, PDB *pdb)という関数を bondPDB2.c 中に作成せよ。

（３）（１）、（２）の関数を利用して、ex5.c の中に main 関数を設計し、実装せよ。

下記のかたちで、コンパイル、リンクができること。

```
$ cc –c bondPDB2.c –o bondPDB2.o
$ cc –c pdbRead.c –o pdbRead.o
$ cc –c ex5.c –o ex5.o
$ cc ex5.o bondPDB2.o pdbRead.o –o ex5 –lm
```

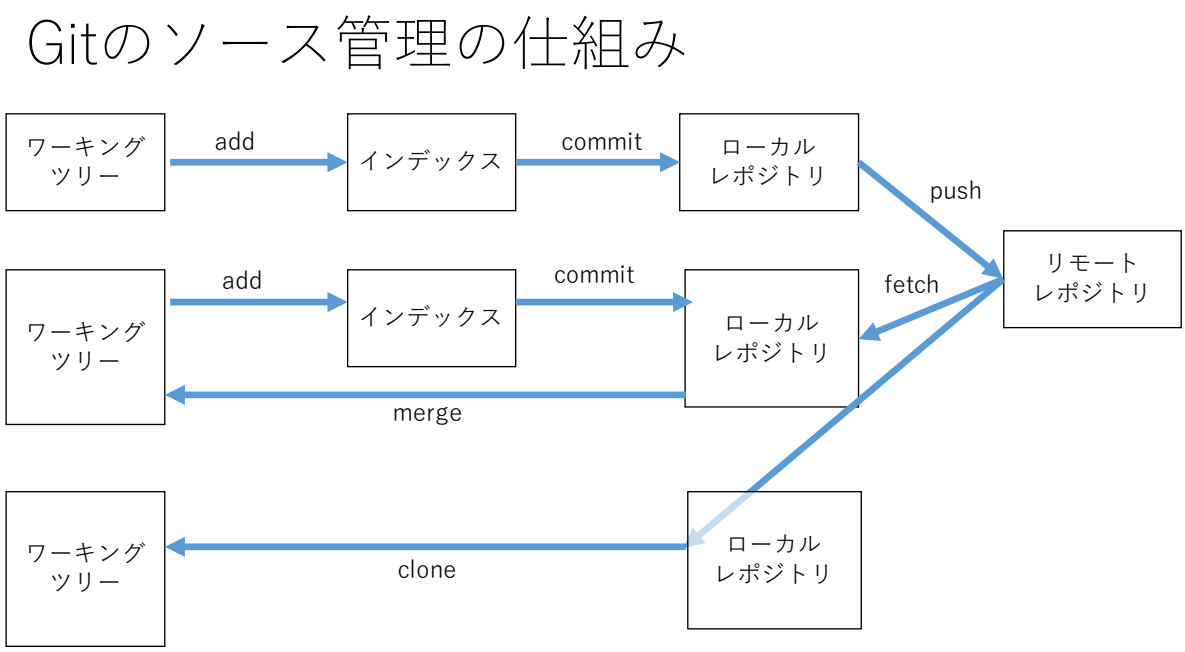
発展課題 II　発展課題 I の後者のように、originx, originy, scale により画像の位置と大きさが変更できるように、bondCADraw2 を拡張せよ。

補遺 1 ：git の使い方

　git は分散型のソースコードの管理・共有化のためのツールです。ソースコードの変更履歴を管理して、いつでも元に戻したり、最新版のコードに変更したりするために利用されます。加えて、大人数で開発する場合には、それぞれの開発者の環境（ディレクトリ）に開発コードが置かれますが、それぞれの変更や新規開発をお互いに教諭する必要があります。

　まず、ファイルが格納される場所が 4 箇所あることを理解しましょう。

ワーキング・ツリー：現在開発を行っているディレクトリ
インデックス：一時的に変更されたファイル等を格納しておく所
ローカル・レポジトリ：開発中のホスト上に格納されている場所（個人でのソース管理）
リモート・レポジトリ：他者と共有するためにファイルが格納されている場所




clone: リモートレポジトリから、手元のホストにワーキングディレクトリを作成します。
add: 変更したファイル群を一時的に格納します。
　　容易に変更を戻すことができ、履歴が残りません。
commit: ローカル・レポジトリに登録します。
　　まとまった変更があった毎に登録するのが好ましいです。
push: リモート・レポジトリに登録します。
fetch: リモートレポジトリから、ローカルレポジトリにコードをダウンロードします。
merge: ローカルレポジトリの間で commit されているものと比較して、ソースを自動的にまとめます。もし、うまく自動的にまとめられないときは CONFLICT というメッセージを出して、ソースコードに両者のコードを併記します。

これ以外にも、過去のものに戻したり、ブランチ（他のバージョン）をつくったりといった機能を持っています。

だれが管理しているリポジトリもリモートレポジトリを通して、merge していくことが出来ます。また、全ての情報をローカルレジストリにもつことがデフォールトです。そのため、特別に有意なレポジトリを仮定しなくて良いので、分散型と呼ばれます。共同開発を行う上で、重要な要素であるため、最近はこの分散型のソース管理システムが利用されるようになってきました、

下記のサイトが詳しく書かれていますので、参考にしてみて下さい。
https://osdn.jp/magazine/09/02/02/0655246
https://osdn.jp/magazine/09/03/16/0831212

今回は、学科の中で、GitBucket というサイトを立ち上げ、その場で管理を行っています。多くのオープンソース型開発のツールは、github、SourceForge、OSDN といった無償（機能によっては有償）のサイトが立ち上がっており、自由にオープンソースソフトウェアを開発できるようになっています。

補遺 2：PostScript 形式のファイル

今回の演習では、PostScript (以下、ＰＳと呼ぶ)というプリンタ向け言語を使って、お絵かきをする一連のプログラムを作成している。まず、簡単に説明をしておく。スタックを用いた言語である。詳細は、http://tutorial.jp/graph/ps/psman.pdf にあるので、参考にせよ。

下記に、今回の演習の中で最低限必要な命令を用いた参考プログラムを用意しておく。これをポストスクリプトに対応したプリンタで直接表示したり、Ghostscript というプログラムで画面に表示することが可能である。

後者に関しては、作成したファイルを下記のプログラムで実行することができる。

$ gs -sDEVICE=x11 ファイル名

また、

$ gs -sDEVICE=x11

として実行すると、下記の命令を一つひとつ実行していくことで画面に表示できる。

| | 解説（単位はポイント、１／７２インチ） |
|---|---|
| %! PS-Adobe-3.0 | ポストスクリプトファイルであることの宣言 |
| 10 10 moveto | ペンの位置を(10, 10)に移動する。 |
| 72 72 lineto | 現在のペンの位置から、(72,72)に線を引くことにする |
| stroke | 実際に線を引く |
| 72 10　moveto | ペンの位置を(72, 10)に移動する |
| 0　72 rlineto | 現在のペンの位置から、ベクトル(0, 72)で移動して線を引く |
| stroke | 実際に線を引く |
| 100 10 moveto | ペンの位置を(100, 10)に移動する |
| 72 0　rlineto | 現在のペンの位置から、ベクトル(72, 0)で移動して線を引く |
| 0 72　rlineto | 現在のペンの位置から、ベクトル(0, 72)で移動して線を引く |
| -72 0 rlineto | 現在のペンの位置から、ベクトル(-72,0)で移動して線を引く |
| 0 -72 rlineto | 現在のペンの位置から、ベクトル(0,-72)で移動して線を引く |
| closepath | ペンの開始と終点を閉じる |
| stroke | 実際に線を引く |
| showpage | 一頁分の終了 |

上記のファイルを gs をつかって表示すると、２本の線と、ひとつの四角形が描かれる。

補遺 3：ＰＤＢ形式のファイル

全ての課題に渡って、下記に示すタンパク質などの構造を示すＰＤＢファイルが取り扱うための関数を作りながら実際の演習で用いるものとする。

ここで用いるＰＤＢファイルは、ムードル上にアップしているファイルを利用せよ。

このＰＤＢファイルの中には、下記の ATOM で始まる行のような記述がされている。

```
0         1         2         3         4         5         6         7
01234567890123456789012345678901234567890123456789012345678901234567890123456789
ATOM     86  CA  ARG    11      -2.455   1.706  24.211  1.00 17.72      1AAK 146
```

PDB 形式のファイルにおいては、一行がレコードとして取り扱われており、１行 80 文字(0-79)＋改行コードとして決まっている。先頭の６文字がそのレコードの意味を示す。この６文字(0-5)が”ATOM　“（後ろ２文字が空白）であれば、タンパク質に含まれる原子を、”HETATM”であれば、それ以外の分子の原子を示している。

ここで示したレコードは、”ATOM　“で始まっているので、タンパク質原子の座標が入っている行であることが分かる。これ以外の情報も、それぞれ先頭の６文字に従って、このファイルの中に格納されており、それぞれのレコードのフォーマット異なる。詳細なフォーマットは、http://deposit.rcsb.org/adit/docs/pdb_atom_format.html を参考にせよ。ただし、先頭が１文字目になっているので、Ｃ言語とは１だけインデックスが異なるので注意せよ。

今、先頭の６文字 (0-5) が”ATOM　“（後ろ２文字が空白）で始まり、原子の名前を表現する４文字(12-15)が、” CA “（CA の前後に空白）となっている行だけに注目すると、タンパク質のうちα炭素だけの情報を抜き出すことができる。それぞれの原子の位置は、ｘ座標（%8.3f, 30-37 文字目)、ｙ座標(%8.3f, 38-45 文字目）、ｚ座標(%8.3f, 46-53 文字目)を読み出すことで利用できる。