

グラフィックス演習

生命情報工学科・3年次第4クォータ・演習科目

【目的】

生命活動を理解する上で、コンピュータ上で可視化する技術は必須である。特に、3次元構造を表現することができれば、その理解は進む。特に、OpenGLを用いた三次元グラフィックスは、世の中で多く利用されている手法の一つである。そこで、C言語を用いて、三次元グラフィックスを利用した分子表示プログラムの作成ができる技術と知識を身につけることを目的として、演習を行う。加えて、今回は、大規模プログラムの開発のための分割コンパイル、makeを利用したソースコード管理、gitを用いたソースのバージョン管理などを学ぶと共に、演習の一部をグループで行うことで、より実践に近いプログラム開発を実践する。

【実施内容】

下記に実施内容を示す。

1. 予習課題（C言語の復習、PDBファイルの取り扱い）（個人）
2. 第1週：makeを用いたソースコード管理とアクティビティ図（個人）12/6
3. 第2週：三次元分子表示の調査（個人）とPDBからの情報抽出（グループ）12/13
4. 第3週：OpenGLのプログラミング（個人）12/20
5. 第4週以降：分子表示プログラム（グループ）1/10, 1/17, 1/24, 1/31, 2/7, 2/21（最終提出）

【提出物一覧】

1. 予習課題（補遺1）
2016年12月6日（火曜日）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 13:00（13:00より説明会）
 - ☐ ソースコード(tgz形式) (ムードル上) 12/6
 - ☐ 開発状況報告書（フォーマット有、pdf形式）(ムードル上) 12/6
 - ☐ 自己点検書（フォーマット有、配付）(12/6日当日)
2. 個人で対応するべきもの
 - Makefileをもちいた予習課題の拡張（第1週課題）
 - ☐ ソースコード(tgz形式) (ムードル上) 12/13
 - ☐ アクティビティ図（PDFもしくはpng形式）(ムードル上) 12/13
 - ☐ 開発状況報告書（フォーマット有、配付）(ムードル上) 12/13
 - RasMol/chimera等の比較調査（第2週課題）
 - ☐ 調査報告（課題有、フォーマット自由）(ムードル上) 12/20

グラフィックス演習

●PDB からの情報抽出（第 2 週）

- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー（フォーマット有/PDF
- ☐ ）（ムードル上）12/20

●OpenGL を用いた課題に関するもの（第 3 週）

- ☐ ソースコード(tgz 形式）（ムードル上） 1/10
- ☐ ベストショット画像(png 形式）（ムードル上）1/10
- ☐ 要件定義書(PDF 形式）（ムードル上）1/10
- ☐ 開発状況報告書（フォーマット有、配付）1/10

●分子表示ソフトウェアに関するもの（第 4 週以降）（全てムードル上、PDF）

- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー 1（フォーマット有）1/10～1/17
- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー 2（フォーマット有）1/17～1/24
- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー 3（フォーマット有）1/24～1/31
- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー 4（フォーマット有）1/31～2/7
- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー 5（フォーマット有）2/7～2/14
- ☐ 週報（個人）・グループのメンバーの活動のピアレビュー終（フォーマット有）2/14～2/21

3. グループで対応するべきもの、グループ活動報告に関するもの

●PDB からの情報抽出（第 2 週）

- ☐ グループでのソースコード（ムードル上）12/20
- 要件定義書（ムードル上）12/20
- 活動報告書（フォーマット有）

●分子表示ソフトウェア（第 4 週目以降）

- ☐ ソースコード(tgz 形式）（ムードル上）2/21
- ☐ 要件定義書・工程表 1（ムードル上）1/17
- ☐ 要件定義書・工程表 2（ムードル上）1/24
- ☐ 要件定義書・工程表 3（ムードル上）1/31
- ☐ 要件定義書・工程表 4（ムードル上）2/7
- ☐ 要件定義書・工程表 5（ムードル上）2/14
- ☐ 要件定義書最終・工程表最終(PDF 形式）（ムードル上）2/21
- ☐ マニュアル（表示画像を含むこと）(PDF 形式）（ムードル上）2/21

4. 自己点検書・ルーブリック評価の提出（全 7 回）

（全員が参加する毎回、出席確認を含めた形で、印刷物への記述により提出（補遺 4、5 参照）

【演習の進め方】

毎回（出席が必要な回は全部で6回＋まとめ）、指定された火曜日の3－4限に新しい内容の説明、開発状況の報告を行ってまいります。次回以降は、指定の教室にて実習を行います。その際、USBメモリにて、開発したプログラムを持ってきて下さい。その際、Linuxを利用できるWindowsノートを各班に1台、貸し出す予定です。

グループでの活動は、個々のグループの活動方針に任せます。所属研究室の予定等と調節しながら、グループでの活動を進めて下さい。週に1コマは集合できる時間をつくり、その時間をフォーラム上に情報を展開して下さい。海外留学している場合でも、情報の共有は図るようにして下さい。集まらない場合、また、集合した場合のいずれも、「活動、ソースコード、意見」は、ムードル上の各グループ毎のフォーラムにて共有してください。個人戦の内容に関してもフォーラムを通して、互いが議論することを推奨します。評価者は、それをみて活動状況を確認し、評価します。加えて、ソースの共有の為に、gitというコマンドとそのためサーバを利用します。これは、ソースコードの共有、共同開発、バージョン管理のために用いられているものです。こちらの活動状況も確認します。今後の予定（全体での動き）を下記に示します。

■第1回：makeを用いたソースコード管理とアクティビティ図（個人）（12月6日）

【個人戦＋グループ活動】

予習課題のためのmakefileの作成

予習課題IVのbondCADraw2関数のアクティビティ図の作成

（第2回実施日13時までに提出）

Windows/OS-X上で実行するLinux環境の構築（富重さん対応）

■第2回：三次元分子表示の調査（個人）とPDBからの情報抽出（グループ）（12月13日）

【団体戦】PDBファイルの情報からプログラムを作成する（仕事の手分けと実装準備）

- 重心、分子を囲む直方体の大きさ、分子の大きさ、密度、表面等の計算。
- グループで作成したものを途中経過として、提出してまいります。

（12月20日13時までに途中経過＋進捗状況として提出）

【個人戦】chimera/rasmolを使った分子表示グラフィックスについての調査研究

- 実際に利用した結果
- 分子表示の種類、GUIに関する調査

（第3回実施日13時までに一旦提出、2月21日13時までに最終提出）

■第3回：OpenGLのプログラミング（個人）（12月20日）

【個人戦＋グループ活動】OpenGLを使ったグラフィックス入門：OpenGLによる作品

グループで実際の演習の内容について連携するが、同一作品は認められない。

（第4回実施日13時までに一旦提出、最終日13時までに最終提出）

グラフィックス演習

■第4回：分子表示プログラム（グループ）（1月10日）

【団体戦】分子グラフィックスについての要件定義書の作成

機能、実装方法などの検討、アクティビティ図によるアルゴリズム表示

■第5回：分子表示プログラム（グループ）~~（1月17日）~~

【団体戦】分子グラフィックス実装の開始

機能、実装方法などの再検討、アクティビティ図によるアルゴリズム表示

~~1月24日はグループで自由に演習可能~~

■第6回：分子表示プログラム（グループ）（1月31日）

【団体戦】分子グラフィックス実装（活動の中間報告）

機能、実装方法などの検討

2月7日、2月14日はグループで自由に演習可能

随時、要件定義書（進捗報告含む）を更新掛けたものをアップロードすること

■最終日：まとめ

最終提出：締め切り日（2月21日）

【団体戦】最終成果物（分子グラフィックス）

（ソースコード、要件定義書、マニュアル、画像：ムードルに提出）

【個人戦】分子グラフィックスに関する調査研究（レポート；ムードルに提出）、

OpenGL による作品（ソースコード、画像、概説；ムードルに提出）

注意）【週報（個人）＋ピアレビュー】により、グループ活動における互いの評価を実施します。これは、グループの活動状況を把握するために必須の事です。

【要件定義書・中間（最終）報告】

毎週、ムードルに提出することにより、これにより、進捗状況を把握することが出来ます。

※ フォーマット（ピアレビュー、週報、要件定義書等）は、ムードル上にそのフォーマットを置いているので、ダウンロードして入力し、PDF 型でアップロードすること。

【第1週】makeを用いたソースコード管理とアクティビティ図（個人）

今回は、分割コンパイル、ソースコードの管理のためにmakeを用いた管理を理解し、かつ、ソースコードの流れを共有するための手法としてUMLのアクティビティ図を学ぶことが目的です。

【予習】

make、及び、アクティビティ図について、予め学んでおくことが望ましい。ムードル上に、当日使用するパワーポイントの例がアップロードされているので、それをみて確認しておくこと。

【課題と演習】

● 分割コンパイルとmake（補遺6）

課題1【個人戦】makefileの設計

予習課題全てのプログラムがmakefileにより作成できるようにして下さい。

修正するプログラム内容があれば、修正してもよいです。また、発展課題に取り組んでもよいです。

これにより、ex1～ex5 及び発展課題がコンパイルできるようなmakefileを作成して、予習課題と同様にtar コマンドを用いてアーカイブして、moodle 上の所定の場所にアップして下さい。

\$ make

\$ make test

により、それぞれ「コンパイル・リンク」により実行形式が作成され、あるいは、テストコマンドが動き、その結果(ex1.txt～ex5.txt)が出力できるようにしなさい。グループ内で情報は共有してもよい。

【グループ演習課題】 演習当日に実施するグループ演習課題

ex4, ex5 について、補遺6で示したmakefileをどのように書き換えればよいかを考えてみましょう。

pdbRead.c, bondCA.c, bondCA2.c を、libPDB.a として一つのライブラリにまとめて利用する様に設計してみましょう。

● アクティビティ図（補遺 7）

課題 2 【個人戦】

予習課題における bondCADraw2 関数の動作をアクティビティ図で表現し、提出して下さい。

手書きの図を取り込むことでも問題ありませんが、下記の asah* community というソフトウェアは、無償の UML モデリングツールです。こちらを利用したアクティビティ図の作成も可能です。

<http://astah.change-vision.com/ja/product/astah-community.html>

【グループ演習課題】 演習当日に実施するグループ演習課題

次の関数（calcFactorial）の流れをアクティビティ図を用いて記載して下さい。制御（アクションノード）には、可能であれば、何をしている行を記し、分からない場合は、その行の命令をそのまま記述せよ。

```
int calcFactorial(int n)
{
    int v;
    int i;

    if(n<0) {
        v = 0;
    } else {
        v = 1;
        for(i=0; i<=n; i++) {
            v *= i;
        }
    }
    return v;
}
```

【第2週】グループ活動1 PDB ファイルから情報を自由に取り出すプログラムの開発

グループでプログラム群を共同開発してもらいます。1～10の基本課題と11の発展課題からなります。最終的に作成するPDBからの分子表示ソフトウェアの準備（チーム開発準備、PDBファイルからの情報抽出準備）を目的とした活動です。

【予習】

予習課題を十分に見直し、以下に示す作業がどのような手順となるかを予め想定しておきましょう。

【課題と演習】

● git を利用したソースコード管理

配付資料及びムードル上の資料に基づいて、git とよばれるソースコード管理に関する手法を解説する。その後、グループ内で git の利用に関して、演習を行う。

- ・ ソースのダウンロード(clone)
- ・ ソースのアップロード(push)
- ・ ソース開発途中での登録(add)
- ・ ソースの登録(commit)
- ・ 他のソースとのマージ(fetch, merge, pull)
- ・ ブランチの切替(checkout)

● 課題内容（詳細は下記）

- 1～2：構造体に付加的な情報を加えます。
- 3～10：PDB が示す分子から情報を抽出します。
- 11：PDB が示す分子の情報から、原子密度を計算します。

【本日の作業】

1～10に関して、アクティビティ図をグループのメンバーで確認し、その開発方針を確認し合っておくこと。

（1）グループでの課題の確認

1～11までの課題を分担して下さい。

- ※ それぞれの課題はそれぞれ担当の一人のひとが担当して下さい。実装する方法や内容に関して、相談することはよいですが、単に、だれかが開発したものをコピー／ペーストのみを行う事は決して行わないで下さい。

グラフィックス演習

- ※ 本日、作成予定の PDB.h 内の構造体（メンバー変数）及びそれを用いる関数群について、そのアクティビティ図（略図）を作成することで、互いが開発する流れを理解しておいて下さい。
- ※ 今回、開発した関数は全て、ライブラリとしてまとめます。libPDB.a に含まれるように makefile を変更して下さい。
- ※ 評価としては、個人の評価に、グループ全体の活性度、完成度が付加されます。

（２） 互いの開発状況は、フォーラム（グループ別）にて、情報の交換を行い、開発したものは、途中、フォーラム上に添付ファイルとしてアップロードしてください。

- ※ 他の方のプログラム上のバグを発見した場合には、その旨、フォーラムにアップし、修正すべき箇所を指摘するなどして、グループ全体で改善を図ってください。

（３） 締め切りまでに、終了したところまで、開発報告書（裏面参考）を担当者がまとめ上げ、ソースコードをディレクトリ毎アーカイブしたものをアップロードしなさい。

開発報告書（例）

開発内容	開発想定時間	開発時間	担当者	状況

担当者として決めるべき事：

役割

○全体のリーダー：全ての課題に影響

- ・ 全体の進捗状況を確認し、工程表、要件定義書、ソースコードをまとめてアップロードする。
- ・ 早期に **Makefile の改善**し、フォーラムにアップロードすることで、全体のプロジェクトが動くようにすること。

○全体のサブリーダー：全ての課題に影響

- ・ リーダーの動きを補佐し、リーダーに何かあるときは工程表、要件定義書、ソースコードのアップロードを代行すること。
- ・ 開発すべき関数及び構造体が入った、**PDB.h を早期に開発（課題 1、2）**し、フォーラムにアップロードし、作業工程に各々が入れるようにする。
（ソースコードの全体の管理者といえる）

グラフィックス演習

以下の担当については、リーダー、サブリーダーが実施してもよい。難易度を★（最大3）で示す。

○担当・課題8：課題9～10に影響 ★★★

・課題8を担当（pdbRead関数の拡張）

各課題担当：課題3～7、課題9～10、課題11（発展課題）

課題3 分子中心 ★

課題4 半径 ★

課題5 最小 ★

課題6 最大 ★

課題7 出力 （課題3～6によるまとめ） ★★

課題9 残基の種類 ★★

課題10 元素の種類 ★★

課題11 密度計算（発展課題）★★★

元素の種類に応じて、密度分布を計算する関数を用いて計算

三次元空間をボクセルに分けることで表示可能

グラフィックス演習

■具体的な課題提案（団体戦）

● 【団体戦】 PDB ファイルの情報からプログラムを作成する（仕事の手分けと実装準備）

【本日の演習】

これまで作成してきたライブラリ (libPDB.a) を利用し、拡張して新しいプログラムを作成します。このプログラム作成を分担して作成してもらいます。今日は、3 以降について、それぞれの関数の流れをアクティビティ図として、略記しながら、グループで作成すべきプログラムを記せ。

全てのプログラムは、\$ make によりコンパイル・リンクが行われ、作成されるように makfile を修正せよ。今回作成する main 関数を除く、新たな関数は全て、libPDB.a にアーカイブされるようにせよ。

1. PDB のメンバー変数の拡張

```
(ア) Atom Center;           // 分子中心の座標を挿入する
(イ) float maxRadius;       // 中心から尤も遠い原子までの距離を格納する
(ウ) Atom 型 min;           // x, y, z 座標のうち、尤も小さい座標が格納されている。
(エ) Atom 型 max;           // x, y, z 座標のうち、尤も大きい座標が格納されている。
```

2. Atom のメンバー変数の拡張

```
(ア) char resName[4];       // 残基名（3文字表記）が格納
(イ) int resNumber;         // 残基番号が格納される
(ウ) float tempFactor;     // 温度因子が格納される
(エ) float occupancy;      // 占有度が格納される
(オ) char atomName[5];     // 原子名が格納される
(カ) char element[3];      // 元素記号が格納される
```

3. 分子の中心の計算

```
void lpdbCenterCalc(PDB* pdb);   lpdbCenterCalc.c に実装、PDB.h に宣言。
    pdb ファイルから読み込んできた分子の中心を求め、Center に格納する。
プログラム : pdbCenterCalc PDB ファイル 出力ファイル
    pdbCenterCalc.c に作成された main 関数から、必要な関数を呼びだして、
    出力ファイル内に中心座標を出力するプログラムを作成せよ。
```

4. 中心からの尤も遠い原子までの距離の計算

```
void lpdbSizeCalc(PDB* pdb);     lpdbSizeCalc.c に実装、PDB.h に宣言。
    Center が求まっているとして、その中心から尤も遠い原子を求め、
    その距離を maxRadius に格納する。
```

グラフィックス演習

プログラム : pdbSizeCalc PDB ファイル 出力ファイル

pdbSizeCalc.c に作成された main 関数から、必要な関数を呼びだして、
出力ファイル内に最大距離を出力するプログラムを作成せよ。

5. 尤も小さい座標の格納

void lpdbMinCalc(PDB* pdb); lpdbMinCalc.c に実装、PDB.h に宣言。

全ての原子について、もっとも小さい x, y, z のそれぞれの座標を求め、
min に格納する。

プログラム : pdbMinCalc PDB ファイル 出力ファイル

pdbMinCalc.c に作成された main 関数から、必要な関数を呼びだして、
出力ファイル内に最小座標を出力するプログラムを作成せよ。

6. 尤も大きい座標の格納

void lpdbMaxCalc(PDB* pdb);

全ての原子について、もっとも大きい x, y, z のそれぞれの座標を求め、
max に格納する。

プログラム : pdbMaxCalc PDB ファイル 出力ファイル

pdbMaxCalc.c に作成された main 関数から、必要な関数を呼びだして、
出力ファイル内に最小座標を出力するプログラムを作成せよ。

7. PDB 座標の情報の出力

void lpdbInfoPrint(FILE* fpt, PDB* pdb);

3～6 で計算した全ての情報を指定したファイルポインタに出力する関数

プログラム : pdbInfoPrint PDB ファイル 出力ファイル

pdbInfoPrint.c に作成された main 関数から、必要な関数を呼びだして、
出力ファイル内に 3～6 の全ての情報を記述するプログラム。
フォーマットは下記を参考にすること

center: 10 20 30

maxRadius: 100

min: -80 -20 -10

max: 70 80 30

8. pdbRead 関数の拡張

拡張した下記の変数を、座標値と共に読み込んでくる関数に拡張する。

```
(ア) char resName[4];           // 残基名（3文字表記）が格納
(イ) int resNumber;            // 残基番号が格納される
(ウ) float tempFactor;         // 温度因子が格納される
(エ) float occupancy;         // 占有度が格納される
(オ) char atomName[5];         // 原子名が格納される
(カ) char element[3];          // 元素記号が格納される
```

9. 残基毎の数のリストを印字

resName に格納されている残基の種類に応じて、 α 炭素の数をカウントし、その数を次の様に表示する。アミノ酸残基の順序は問わない。20種のアミノ酸以外はOTHとしてカウントする。また、詳細なフォーマットは、問わないが、アミノ酸残基名の後ろにコロンとして、分離する。

```
ALA:    5
GLY:    10
...
```

10. 元素毎の数のリストを印字

全ての原子について、element に格納されている元素の種類に応じて、原子の数をカウントし、その数を次の様に表示する。原子の順序は問わない。通常元素に含まれないC, H, N, O, P以外の元素は、Xとしてよい（全て表示してもよい）。また、詳細なフォーマットは、問わないが、元素記号の後ろにコロンとして、分離する。

```
C:      500
H:      1000
...
```

11. 密度の計算（発展課題）

原子の密度について、pdb2dns6.c 及び、lpdb2dsn6.c を参考にしながら、作成してみよ。これらのプログラムは、Atomにあたる構造体の宣言方法などが異なっており、そのまま使用することは困難である。原子密度の計算方法などを参考にせよ。

グラフィックス演習

● 【個人戦】 chimera/rasmol を使った分子表示グラフィックスについての調査研究

- 実際に利用した結果
- 分子表示の種類、GUI に関する調査

【個人調査課題 1】（第 2 週目の課題）

PDB (Protein Data Bank) 上の興味のあるタンパク質をひとつ選択し、RasMol/Chimera のふたつの分子表示ソフトウェアを用いて、そのタンパク質分子を表示せよ。

また、タンパク質を RasMol/VMD のもつ様々な表示形式、色で表示し、下記の点に着目して比較せよ。更に、そのそれぞれのソフトウェアに問題点がある場合に、どのような改善が可能かについて指摘せよ。このとき、ふたつのソフトウェアが分子表示ソフトウェアであることを意識して議論せよ。

下記に記した課題に沿って、レポートを作成し、提出すること。

1. 表示ソフトウェアに関し、着目すべき点を記述せよ。

- 分子表示に使われてる 3 次元描画方法：注目すべき点
 - 透視投影、平行投影
 - シェーディング
- GUI (Graphical User Interface)
 - マウスによる分子操作
 - メニュー、ウィンドウの配置, プルダウンメニューの形式
 - キーボードによるショートカット操作
- CUI (Character User Interface) の有無
 - コマンド入力の方法、マクロコマンドの存在
- その他、気になったところ

2. 作成した図を添付せよ。

選択したタンパク質の名前、ID を明記の上、学籍番号に応じて、下記の三枚の絵を作成し、レポートに添付せよ。図には、その図のタイトルと説明をつけること。また、図の番号を設定し、レポート本文中から引用すること。

2. 1 RasMol を使って、分子表示。学籍番号下二桁を 6 で割ったあまりに 1 を加えた数に応じて以下の表示を実施し、図を作成せよ。

1. Backbone
2. Sticks
3. Spacefill
4. Ball&Sticks
5. Ribbons
6. Cartoons

グラフィックス演習

また、それぞれどのような分子表示法か、すべてについて簡単に説明せよ。

2. 2. Chimera を使って、分子表示。学籍番号下二桁を 5 で割ったあまりに 1 を加えた数に応じて以下の表示。

1. 全原子 (sphere 表示)
2. 全原子 (ball & stick 表示)
3. α 炭素のチェーントレース (Stick 表示)
4. 表面表示 (Surface)
5. リボン (rounded)

また、それぞれどのような分子表示法か、すべてについて簡単に説明せよ。

3. Chimera により、密度の等高面を表示せよ。

- Moodle 上から、密度計算のプログラムをダウンロードし、解凍せよ。
- 解凍したディレクトリ内で下記のコマンドを実行し、できあがったファイルを chimera を用いて表示し、密度表示すること。

```
$ cd ダウンロードしたディレクトリ
$ tar xvzf pdb2dsn6.tar.gz
$ cd src
$ make depend
$ make
$ ./pdb2dsn6 ダウンロードしたファイル名 出力ファイル名.ds6 5
```

出力ファイル名.ds6 を chimera で読み込む。その際、suffix に.ds6 としておくと、DSN6 フォーマットであることを自動的に認識できる。

全員 : Chimera による等高面表示 : 設定した等高面の値を付記すること。

【第3週】個人活動 OpenGL のプログラミング

三次元グラフィックスのプログラミングとして、OpenGL を用いたプログラミング基礎を学ぶことが目的である。下記の課題に従い、演習を行う。最終的な分子グラフィックスにつながる技術を学ぶ。

【予習】

OpenGL については、前期のコンピューターグラフィックスBで学んでいる。その際の教科書等を参考にせよ。必ず、当日持ってくること。また、当日の説明資料が、ムードル上にアップされているので、参考にしておくこと。

【課題と演習】

● OpenGL の理解 : glut_sample プログラム群の理解

1. 配付資料中の glut_sample, glut_sample1, glut_sample2, glut_sample3, glut_sample4, 及び、makefile に関して、解説後、各グループ毎にそれぞれのプログラムの動作に関して、配付資料中にまとめる。
2. 配付資料中の OpenGL に関する設問に関して、各グループ毎に整理し、自己点検書に整理して提出する。

● 演習問題: glut_sample5 の作成

glut_sample.c, glut_sample2.c, glut_sample3.c, glut_sample4.c を例しながら、以下の機能をもつ glut_sample5 を作成しなさい。

1. 下記の機能の要件定義表（下記）を作成せよ。

機能番号	機能（機能の略記）	ファイル名	関数名	予定時間	作成時間	完成度	備考（何をしたかを略記）
1	Makefile の作成	makefile	-	0:10	0:15	100%	glut_sample5 への対応
2	透視投影の実現	glut_sample5.c	reshape	0:10	0:10	100%	平行投影からの変更
3	animation モード：変数宣言	myGUI.h/glut_sample5.c	-	0:15	0:05	100%	外部変数の宣言 movable/velocity

2. Makefile の修正

1. glut_sample4.c のファイルを glut_sample5.c としてコピーしなさい。
2. glut_sample5.c から glut_sample5 が作成されるように、Makefile を修正しなさい。
※ この段階では、glut_sample5 は glut_sample4 と同じ動き方をする。
3. その後、glut_my_mouse.c, glut_my_key.c, glut_sample5.c 等を変更し、下記の仕様を満たす glut_sample5 を作成して下さい。
※その結果として、glut_sample4 に関しても、mouse/key の機能に変更されるものとする。

3. 画像表示のモードは次のものを満たすように変更する

1. 透視投影モード

※ glut_sample/glut_sample2/glut_sample3 と同じ

※ glut_sample4 は平行投影モードなので注意が必要

4. 回転し続けるアニメーションの仕様変更

1. 現在の仕様確認

`theta += 0.3*turn;`

0.3: 回転速度

turn: 回転の向き

2. 仕様変更

`theta += movable*turn*velocity;`

movable: 0 or 1

1: 回転、0:停止

(キーボード s により制御、交互に変化)

turn: -1 or +1

+1/-1 (キーボード r により制御、交互に変化)

velocity: 浮動小数点(float 型)

浮動小数点として、正負も許可する。マウスにより、速度を制御

5. キーボードによる GUI の設計

1. キーボードコールバック関数が受ける情報を fprintf を使い画面(console)に出せるようにする。

押されたキー、押された時のマウスの位置が引数としたコールバック関数が呼ばれる。

2. q または Q を押すことによって、プログラムが終了(exit(EXIT_SUCCESS)を呼び出す)する。

3. r を押すことによって、回転している右回転、左回転が切り替わる。

※ glut_sample 等の左ボタンクリックと同じ機能である。マウスの機能の変更により、左ボタンクリックでは変更できなくなる。

※ このとき、回転速度は変化せず、向きだけが変化する。3で指定した turn の+1/-1 を変更する。

※ if を利用しないで+1/-1 を切り替える方法を考えてみよう。(発展課題)

4. s を押すと、回転と停止を交互に繰り返す(トグルボタン)。

※ 3で指定した movable の 0/1 を変更する。

※ if を利用しないで、0/1 を切り替える方法を考えてみよう。(発展課題)

6. マウスによる回転の制御

1. マウスコールバック関数が受ける情報を `fprintf` を使って画面(console)にだせるようにする。
押されたマウスボタンの種類、そのときの状態、その時のマウスの位置が引数としたコールバック関数が呼び出される。
2. マウスの左ボタンにより、回転速度(velocity)を変更する。
 - ※ 元々の左ボタンの機能（回転の向き）はなくなる。
3. 左ボタンを押した最初の位置から現在のマウスの位置までの x 座標の変化量に比例して、回転速度(velocity)を変化させる。
 - ※ 比例係数は各自、適切な量を設定せよ。(0.01~0.1 を想定している)
 - ※ 押したところより、右側にずれた時には、velocity を正。
 - ※ 押したところより、左側にずれた時には、velocity を負。
 - ※ 回転するかしないかは、キーボード s によって制御される movable 変数により決定する。
 - ※ 回転していれば、回転するモードとする。
 - ※ 停止していた場合には、movable 変数を 1 として、回転する。
 - ※ ただし、マウスを押した直後は、変化量が 0 のため、velocity=0 となり、停止したようにみえる。
 - ※ その後、左右にマウスを動かすと回転を始める。
4. 右ボタンの機能は、`glut_sample3`/`glut_sample4` と同じ機能をもつ。
5. SHIFT+左ドラッグの機能は、`glut_sample3` と同じ機能（roll の機能）を実装する。
(発展課題)
 - ※ `glut_sample4` では、機能を失っているので、`glut_sample3` を参考にする事。
6. 中ボタンを押して、上下に動かすと ZoomIn/Out を行うことができる。(発展課題)
 - ※ `glut_sample4` では、機能を失っているので、`glut_sample3` を参考にする事。

7. ポリゴン画像の生成

1. 適当なポリゴンを一枚以上加え、適当な絵を作成する。
2. 現在のポリゴン（四角錐 3 つ）は、消しても良い。
3. アニメーション機能は残し、idle 状態でも動き続ける画像とせよ。

【提出物】

1. 適切な画像のスナップショットをとって、課題提出（スナップショット）に提出して下さい。
2. ソースコードを、src2 ディレクトリ毎、全て一つのファイルにアーカイブして、課題提出（ソースコード）に提出して下さい。
3. 要件定義書、開発報告書（フォーマット有、配付）を作成して、提出して下さい。

【第4週】グループ活動：分子表示プログラムの作成 1

これまで学んできたC言語によるプログラミング技術、ソフトウェア工学、及び、三次元グラフィックスのための技術を用いて、生命情報工学に有用な分子表示ソフトウェアを作成する。

【予習】

ここまでの技術要素を全て復習しておくこと。

【課題と演習】

● 配付資料に添付した参考プログラムである myTest に関する解説を聞き、その後、グループ毎で、当該プログラムの動作に対して確認を行う。開発はすべて src3 のディレクトリの下で実施すること。myTest の一部の機能を利用して開発することで問題はない。

● 第4週目 本日の課題

作業：【団体戦】分子グラフィックスについての要件定義書の作成

機能、実装方法などの検討、アクティビティ図によるアルゴリズム表示

本日の作業：要件定義書の作成と分担

1. 今回のプロジェクト（分子グラフィックス：簡易 RASMOL）のリーダー、サブリーダーを決定せよ。各会議では、書記を決め、議事録を作成すること。Moodle 上での情報交換でも十分である。

（ア）リーダーの仕事は、プロジェクト全体が動くための仕組み作りと工程管理にある。

（イ）サブリーダーの仕事は、リーダーの補佐であり、作業の流れを確認する。

2. 開発する分子グラフィックスソフトの名前、もしくは、開発コード名を決定せよ。

3. 作成する分子グラフィックスの機能を Rasmol や chimera の機能から想定（同じでなくてもよい）し、リストアップせよ。

（ア）共通に要求する機能に関しては、リストに加えること。

（イ）添付の要件定義書（本日、一部提出）を利用しながら、実装する、もしくは、実装したい要件を洗い出す事。

（ウ）1 月中に作成可能な機能だけではなく、あると望ましい機能についてもリストすること。

4. 上記で列挙した機能を分類し、それぞれの機能を関数に割り当て、関数毎に、担当を分担する。

グラフィックス演習

- (ア) 次回までにそれぞれの機能に対して、アクティビティ図を作成するための、分担せよ。次回までにアクティビティ図を作成し、提出する。
 - (イ) 各人が担当となる機能（関数）として2つ以上を担当すること。
 - (ウ) 最終的に、作成する担当とは異なっても良い。もちろん、担当者が同じであるほうが対応しやすい。
 - (エ) 1月に入り、実際にプロジェクトが進行する中で、途中の変更や複数分担もよしとする。その旨、最終的に提出する要件定義書等にその旨を記述すること。
-

5. 各々の機能の実装を検討する中で、必要となる共通変数の洗い出しを行う事。必要となる外部変数以外にも、想定されるものはリストアップしておくこと。

共通変数に関しては【要件1】にそれを記述する。正式なリストアップは、アクティビティ図を作成する中で、必要となる情報を洗い出す作業を通して実施する。

【第5週】グループ活動：分子表示プログラムの作成2

仕様の決定、担当の再確認を行う。

【予習】

前回作成した要件定義書、及び、アクティビティ図を参考にしながら、今回導入する仕様及び、Chimera/Rasmolの仕様の眺めながら、望むべき機能について調査しておく。

【課題と演習】

第5週目 本日の課題

作業：【団体戦】分子グラフィックスについての要件定義書の確認と開発内容の確認

機能、実装方法などの再検討、アクティビティ図によるアルゴリズムの確認

本日の作業：要件定義書及び分担の再検討

1. 今回のプロジェクト（分子グラフィックス：簡易 RASMOL）の担当を再度確認する。各会議では、書記を決め、議事録を作成すること。Moodle 上での情報交換でも十分である。
 - (ア) リーダーの仕事はプロジェクト全体が動くための仕組み作りと工程管理にある。
 - (イ) サブリーダーの仕事はリーダーの補佐であり、作業の流れを確認する。
 - (ウ) 書記の役割は会議の議事を取り、お互いの意識を統一することにある。
2. 作成された要件定義書及びアクティビティ図を確認しながら、最終的に作成する分子グラフィックスの機能とそれらに関わる共通変数、API 及びその開発担当を再度確認すること。

グラフィックス演習

- (ア) 書記は、清書用に1セット（共通変数用、API用）別に取りに来て下さい。必要であれば、更にとって下さい。トップの所にそれぞれ枝番を打って下さい。
- (イ) まず、自分自身が担当しているものに関して、互いに確認しながら、リストアップして下さい。
- (ウ) 各人が担当となる機能（関数）として2つ以上を担当すること。
- (エ) プロジェクトが進行の中で、途中の変更や複数分担もよい。その旨、提出する要件定義書等にその旨を記述すること。今回の打合せで変更があった場合は、その旨を記すこと。
- (オ) 書記は、全体をまとめて下さい。リーダー、サブリーダーは、その内容を確認して署名して下さい。他のメンバーも内容の確認をして下さい。

- 注意：必要となる変数をリストアップしておかないと作成が困難になります。
- これらのファイルの写真等を取り、ムードル上のグループ毎のミーティングに挙げて下さい。担当はだれでも結構ですが決めておいて下さい。

3. 最終的に作成したプログラムのマニュアルを作成する。マニュアル作成の分担と作成方法を確認する。マニュアルには、作成したアクティビティ図の最終版も添付し、そのソフトウェアの構造がみえるようにすること。

- 注意：全体で1つのファイルとしてマニュアルを提出してもらうので、そのための分担、作成環境などの摺り合わせも行っておくこと。

【第6週】グループ活動：分子表示プログラムの作成2

仕様の決定を行う。

【予習】

進捗状況を確認しながら、担当の変更、協働作業等を含めて作業工程を整理しておく。

【課題と演習】

第6週目 本日の課題

作業：【団体戦】分子グラフィックスについての要件定義書の確認と開発内容の確認

機能、実装方法などの再検討、進行状況の確認

本日の作業：要件定義書及び分担の再検討

グラフィックス演習

1. 今回のプロジェクト（分子グラフィックス：簡易 RASMOL）の担当を再度確認する。各会議では、書記を決め、議事録を作成すること。Moodle 上での情報交換でも十分である。
（ア）リーダーの仕事はプロジェクト全体が動くための仕組み作りと工程管理にある。
（イ）サブリーダーの仕事はリーダーの補佐であり、作業の流れを確認する。
（ウ）書記の役割は会議の議事を取り、お互いの意識を統一することにある。
2. 作業の工程表と問題点・解決方法をリストアップする。
3. 問題となる点の解決方法をグループもしくは、我々との確認のなかで解決しておく。

【まとめ】

【団体戦】最終成果物（分子グラフィックス）

（ソースコード、要件定義書、マニュアル、画像：ムードルに提出）

【ピアレビュー＋進捗状況】（最終週の活動報告＋最終報告）

グループ活動における互いの評価（ムードルに提出）

フォーマット（ピアレビュー＋進捗状況）は、ムードル上に置いているので、ダウンロードして入力して提出。

【当日の課題】

グループ課題の提出状況の確認

個人での提出課題の確認とアンケート、自己点検書の作成

グラフィックス演習

補遺 1：予習課題（C言語復習課題）：C言語による開発とその開発時間の見積もり

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10%の割合で加味する。

◎ 指定した課題Ⅰ～Ⅴにプログラムを、下記の手順に従い、開発せよ。

- (1) prpractice というディレクトリを作成し、そのディレクトリ practice の下で、課題Ⅰ～Ⅴのすべてのプログラムを作成し、提出せよ。
- (2) 開発にあたっては、下記のフォーマットに従い、開発時間の見積もりと現実の開発時間を把握し、下記の表に加えて、自分自身のC言語開発に関してレビューを記述した、レポートを提出せよ。

● 開発したプログラムの提出方法

・手順

- ① practice のディレクトリの親ディレクトリ移動
- ② 以下のプログラムを実行し、practice-xxxxxx.tgz ファイルを作成せよ。このとき、xxxxxx は、学生番号とせよ。

```
$ tar cvzf practice-xxxxxx.tgz practice
```
- ③ 上記で作成したファイルをアップロードせよ。

● 開発状況報告書（開発時間の見積もりと評価）

下記の様式（Moodle 上にテンプレート）に従い、全てのプログラム開発に関して、その開発を始める前に、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、最終的にPDF形式に変換して提出すること。

課題番号	開発に想定した時間(事前) (A[分])	開発に要した時間(事後) (B[分])	想定ずれ $B \div A \times 100$ (%)	開発コード量 (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					

● 自己点検書

自己点検書（配布物、もしくは、ムードル上よりダウンロード）に記述して、演習当日に提出すること。

グラフィックス演習

課題Ⅰ（ファイルの取り扱いの復習）プログラム ex1

【仕様】

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

として、実行することができ、その結果は次の様になる main 関数を含む ex1.c を作成せよ。

・「ファイル名」で指定したファイルが作成され、その内部には、(startx, starty)から(endx, endy)に線をひくポストスクリプト言語によるプログラムが含まれること。

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、出力すること。このとき、startx, starty, endx, endy は入力した数値、それ以外はそのまま文字列であるとする。

注意 1) 実行時の引数において、ファイル名は文字列、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。

注意 2) ポストスクリプト言語の最初と最後に関しては、下記のように設定する（補遺 2 参照）。この後の全ての演習において、main 関数にて、最初と最後の行を出すように設定すること。

```
%! PS-Adobe-3.0
....
....
showpage
```

【実行結果】

```
$ ./ex1 ex1.txt 100 200 300 400
```

として実行した結果 ex1.txt を、同じディレクトリに出力した状態で保持せよ。

【参考】

※ コンパイル方法

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。

※ 実行時の引数

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（補遺 2 参照）

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。ポストスクリプトファイルに関する詳細な意味は、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）プログラム ex2 と関数 bondDraw

【仕様】

下記に示すように、PDB.h に関数プロトタイプ宣言を、bondDraw 関数を実装せよ。この関数を呼び出すことにより、課題Ⅰと同じ動作をする main 関数を ex2.c 内に実装せよ。

● 関数のプロトタイプ宣言

関数のプロトタイプ宣言（型宣言）は、PDB.h の中に記述する。PDB.h の名前の由来は、補遺 3 を参照にせよ。今後の展開の中で利用することになる。

bondDraw のプロトタイプ宣言は、

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

とする。extern は、実装したファイル以外で利用することを意味している。すなわち、ex2.c 及び bond.c の両者において利用する。

設定した PDB.h は、それぞれ ex2.c, bond.c のファイルの中で、挿入(include)することで、関数の型の確認ができる。

具体的には、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、挿入し、型の宣言を行う。

また、具体的に、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#define を閉じる。
```

ととして設定し、関数宣言を行うこととせよ。

● 線を描く関数 bondDraw の実装

下記の関数の

```
void
bondDraw(FILE* fpt, float startx, float starty, float endx, float endy)
{
    // ここに具体的な実装を行う。
}
```

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とする。

● 分割コンパイル

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なになる。

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

注意 1) 今後、複数人での開発や大規模な開発を行う為には、分割コンパイル・リンクは必須の技術である。

【実行結果】

```
$ ./ex2 ex2.txt 105 205 305 405
```

として実行した結果 ex2.txt を、同じディレクトリに出力した状態で保持せよ。

課題III (構造体の復習) プログラム ex3 と関数 bondDraw2

【仕様】

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。

● 構造体の宣言とプロトタイプ宣言

下記に示した構造体 Bond を PDB.h の中に実装せよ。

PDB.h での構造体及びプロトタイプの宣言

```
#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 */
```

注意 1) このとき、ex2 のコンパイルも可能であることを確認する事。今後も、PDB.h は、改変されるが、そのいずれを通して、これまでのプログラムが全てコンパイル・リンクできることを前提とする。

● 関数 bondDraw2 の実装（構造体の利用）

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。

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)
{
    // この中で実装する。
}
```

を bond2.c の中に、関数 bondDraw2 から、前回、開発した関数 bondDraw を呼び出す形で実装せよ。

● 分割コンパイル

今回の場合には、次の様にして、分割コンパイルが可能である。

```
$ cc -c ex3.c
$ cc -c bond.c
$ cc -c bond2.c
$ cc ex2.o bond.o bond2.o -o ex3 -lm
```

【実行結果】

```
$ ./ex3 ex3.txt 110 210 310 410
```

として実行した結果 ex3.txt を、同じディレクトリに出力した状態で保持せよ。

課題Ⅳ（配列の復習）プログラム ex4 と関数 arrayPDBRead/bondCADraw

【仕様】

補遺 2 の PDB ファイルの説明を参考に、プログラム 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(補遺 3)）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、原子の個数 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）の行数が原子の個数となる。
 - ※ 先頭の 6 文字が”ATOM “で始まるもののうち、AtomName(12-15 文字目)が”CA “となっている行の行数が α 炭素を示す行数である。
- (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。


```
fseek(fpt, 0L, SEEK_SET);
```
- (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。座標が行のどこに記述しているかは、補遺 3 を参照すること。

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。このとき、(X0, Y0)~(Xn-1, Yn-1)までが numCAAtom で指定された n 個の α 炭素の座標のうち、(x, y)座標のみを示したものとする。得られる画像としては、z 軸方向に投影した画像となる。

```
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。

\$ 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。

● 分割コンパイル

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
```

【実行結果】

121p.pdb は、ムードル上から、テスト用ファイルとしてダウンロード出来る。

\$./ex4 121p.pdb ex4.txt

として実行した結果 ex4.txt を、同じディレクトリに出力した状態で保持せよ。

発展課題 I (応用) 平行移動と拡大縮小

【仕様】

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、自由な大きさに配置できるように、二つの引数 `originx`, `originy`, `scale` を与え、タンパク質の原点が(`originx`, `originy`)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 oriny 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。

実装方法としては、PDB の原子の位置を、まず、平行移動し、次に、拡大したものに变换した後に、`bondCADraw` を呼び出すか、もしくは、`bondCADraw1` として、`bondCADraw` を拡張して、`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。

後者の場合は、ファイル `bondCA.c` 注の、関数 `bondCADraw` を拡張して、`int bondCADraw(FILE* fpt, arrayPDB pdb, float originx, float originy, float scale)` として、作成せよ。

すなわち、

```
x = (pdb.CA->x + originx)*scalex;  
y = (pdb.CA->y + originy)*scaley;
```

として、新しい座標を設定できるものとする。

【実行結果】

121p.pdb は、モデル上から、テスト用ファイルとしてダウンロード出来る。

```
$ ./ex4-1 121p.pdb ex4-1.txt 30 30 10
```

として実行した結果 `ex4-1.txt` を、同じディレクトリに出力した状態で保持せよ。

課題 V（リストの復習）プログラム ex5 と関数 pdbRead

【仕様】

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;                  /* C A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 の構造体に対応する。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。 最後の原子には、NULL が格納されている。

グラフィックス演習

nextCA その原子が α 炭素である場合には、
次の α 炭素のポインタが指定されている。
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。

-
- ① 一行読む。
 - ② 空行でなければ、③、空行ならば終了。」
 - ③ RECORDNAME が "ATOM" かどうかを確認し、RECORDNAME が、"ATOM" である場合には、④
 を実行、そうでなければ、①に戻る。
 - ④ 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。
 - ⑤ 最初の原子では top を、かつ、現在の原子を current として、④で設定した領域として設定する。
 - ⑥ さらに、ATOMNAME が "CA" である場合には、⑦を次項、そうでなければ、⑧に進む。
 - ⑦ numCA を 1 増加させ、最初の α 炭素では topCA を、かつ、現在の α 炭素を currentCA として、
 設定する。
 - ⑧ nextAtom, nextCA に NULL を設定し、最後の原子、 α 炭素として、次の行に移動する（①に戻
 る）。
-

グラフィックス演習

(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
```

【実行結果】

```
$ ./ex5 121p.pdb ex5.txt
```

として実行した結果 ex5.txt を、同じディレクトリに出力した状態で保持せよ。

発展課題 II (応用) 平行移動と拡大縮小

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、bondCADraw 2 を拡張するように、設計・実装せよ。

【実行結果】

```
$ ./ex5-1 121p.pdb ex5-1.txt
```

として実行した結果 ex5-1.txt を、同じディレクトリに出力した状態で保持せよ。

グラフィックス演習

補遺 2 : PostScript 形式のファイル

今回の演習では、PostScript (以下、P S 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したり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

グラフィックス演習

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。

基本的なコマンドは次の通り。

moveto	ペンの移動
lineto	絶対的な移動として、ペンで線を引く
rlineto	相対的な移動として、ペンで線を引く
closepath	ペンの始点と終点を閉じる。
stroke	実際に、ペンで線を引く

グラフィックス演習

補遺3：PDB形式のファイル1

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
012345678901234567890123456789012345678901234567890123456789
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。

文字列から、浮動小数点をもとめるには、fgetsにより、1行分を文字列として読み込み、その後、その読み込んだ文字列から、sscanfもしくは、atofを用いて、浮動小数点として取り込むことができる。

グラフィックス演習

補遺4 グラフィックス演習（第1週目）

学生番号

氏名

グループの学生の一覧

	ファーストネーム	メモ（自己紹介）
1.本人		
2		
3		
4		
5		

【学習環境】

自宅や大学での学習環境について調査します。下記の問いに答えて下さい。

1. 自宅に利用できるコンピュータ

ある なし あるが自由に使えない

ある場合、その OS（複数可）（ ）

ネット接続 可（方法： ）
不可

2. 大学に持ち込むことができるノート PC

ある なし 上記と同じ

ある場合、その OS（複数可）（ ）

ネット接続 可（方法： ）
不可

3. ムードルへのアクセス方法

（ ）

4. 準備してもらいたい学習環境があれば、下記に記述して下さい。

グラフィックス演習

補遺 5

本日の課題の評価

項目	評価	1 Excellent	2 Good	3 Fair	4 Poor
複数ファイルの開発の手法の理解		makefile を自由に作成することができる。	makefile を読み取り、そのプログラムの作成の手順が読み取れる。	make の必要性は理解できた。	make の必要性が理解できない。
アクティビティ図によるプログラムの流れの理解		自由に、プログラムの流れを表現できる。	条件分岐、繰り返しといった構造化プログラミングの流れを表現できる	アクティビティ図から、流れを読み取れる。	アクティビティ図から、流れを読み取れない。
C 言語によるプログラミングの理解		データ構造やアルゴリズムを駆使したプログラム開発が可能である。	予習課題程度であれば、自分だけで記述できる。	予習課題の I-III は自分だけで記述できる。	予習課題の I-III を自分だけで解くことが困難である。
今後の課題の現状調査					
PDB ファイルの操作		PDB 内にあるタンパク質の原子座標などの情報を使って新たな情報を作り出せる。	今回開発した予習課題に加えて、PDB のフォーマットの意味が分かれば自由に情報を取り出せる。	今回開発した予習課題の関数を利用できる。	今回開発した予習課題の関数の意味が分からない。
OpenGL による三次元グラフィックス		OpenGL を使ったユーザーインターフェースをもつソフトウェアを記述できる。	OpenGL を使って 3 次元物体を記述できる。	OpenGL を使ってプログラミングが可能である。	OpenGL とは何かを知らない。
最終成果物の理解		分子グラフィックスを自由に作成できる		目標とするプログラムが理解できた。	何を作成するべきかが理解できていない。

本日のグループ型の演習についての意見を、下記の語句を選択した後、下記に述べて下さい。

学習する上で有効

有効性を感じない

未だよく分からない

その他、本グラフィックス演習に関する意見・質問、要望があれば記述して下さい。

グラフィックス演習

補遺6 グラフィックス演習（第1週目）：makeによる流れ

----- makefile の実際 -----

```
.SUFFIXES: .c .o .a
```

```
-include "depend.inc"
```

```
SOURCE=bond.c ¥  
        bond2.c
```

```
.c.o:  
        cc -c *.c -o *.o
```

```
all: ex1 ex2 ex3
```

```
ex1: ex1.o  
        cc $.o -o $@ -lm
```

```
ex2: ex2.o bond.o  
        cc $.o bond.o -o $@ -lm
```

```
ex3: ex3.o libBond.a  
        cc $.o -o $@ -lm -L./ -lBond
```

```
# -----
```

```
libBond.a: bond.o bond2.o  
        ar rv $@ bond.o bond2.o
```

```
# OR
```

```
libBond.a: $(SOURCE: .c=.o)  
        ar rv $@ bond.o bond2.o
```

```
# -----
```


グラフィックス演習

test: test-ex1 test-ex2 test-ex3

test-ex1:

./ex1 ex1.out 0 0 10 20

test-ex2:

./ex2 ex2.out 10 20 30 40

test-ex3:

./ex3 ex3.out 10 20 30 40

depend::

cc -M *.c > depend.inc

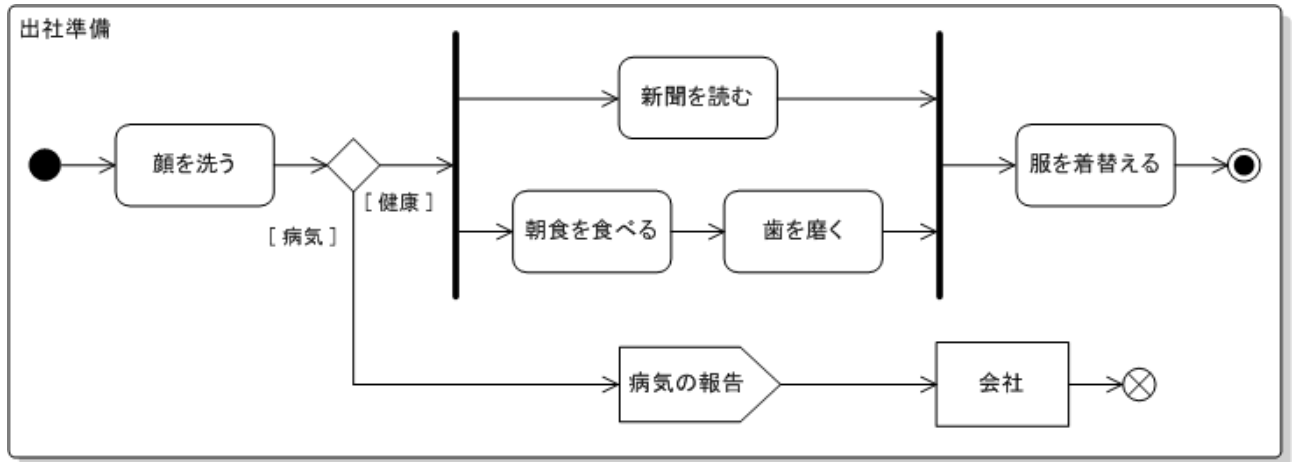
clean::

rm *.o *.a ex1 ex2 ex3

グラフィックス演習

補遺7 グラフィックス演習（第1週目）：アクティビティ図による流れ

<http://www.itsenka.com/contents/development/uml/activity.html> より引用






朝の出社準備

構成要素

要素	表示形式	意味
初期ノード		スコープ内で開始を表します。
最終ノード		スコープ内で終了を表します。
アクションノード		制御を表します。
デシジョンノード / マージノード		条件によるフロー分岐（デシジョンノード）もしくは、複数のフローの合流（マージノード）を表します。
フォークノード / ジョインノード		複数のフローが非同期に実行される（フォークノード）、もしくは、複数の非同期処理が終了する（ジョインノード）ことを表します。

グラフィックス演習

データストア		データーを保持するための記憶領域や装置を表します。
受信ノード		シグナル、またはイベントの発生の待機を表します。
送信とアクションのノード		オブジェクトにシグナルを送信する制御を表します。

グラフィックス演習

(補遺 8) グラフィックス演習 (第 2 週目)

学生番号

氏名

グループの学生のリスト (参加者のみ)

	ファーストネーム	分担した開発内容について
1. 本人		
2		
3		
4		
5		

グラフィックス演習

本日の課題の評価

項目	評価	1 Excellent	2 Good	3 Fair	4 Poor
複数ファイルの開発の手法の理解		makefile を自由に作成することができる。	makefile を読み取り、そのプログラムの作成の手順が読み取れる。	make の必要性は理解できた。	make の必要性が理解できない。
アクティビティ図によるプログラムの流れの理解		自由に、プログラムの流れを表現できる。	条件分岐、繰り返しといった構造化プログラミングの流れを表現できる	アクティビティ図から、流れを読み取れる。	アクティビティ図から、流れを読み取れない。
C 言語によるプログラミングの理解		適切なデータ構造やアルゴリズムを駆使したプログラム開発が可能である。	リストなどのデータ構造を利用したプログラムを作成できる。	アクティビティ図が指定されれば、プログラムを記述できる。	構造化プログラミングの実現方法が分からない。
PDB ファイルの操作		PDB 内にあるタンパク質の原子座標などの情報を使って新たな情報を作り出せる。	今回開発した予習課題に加えて、PDB のフォーマットの意味が分かれば自由に情報を取り出せる。	今回開発した予習課題の関数を利用できる。	今回開発した予習課題の関数の意味が分からない。
今後の課題の現状調査					
OpenGL による三次元グラフィックス		OpenGL を使ったユーザーインターフェースをもつソフトウェアを記述できる。	OpenGL を使って 3 次元物体を記述できる。	OpenGL を使ってプログラミングが可能である。	OpenGL とは何かを知らない。
最終成果物の理解		分子グラフィックスを自由に作成できる	OpenGL の理解を通して、分子グラフィックスの	目標とするプログラムが理解できた。	何を作成するべきかが理解できていない。

本日のグループ型の演習についての意見を、下記の語句を選択した後、下記に述べて下さい。

学習する上で有効

有効性を感じない

未だよく分からない

その他、本グラフィックス演習に関する意見・質問、要望があれば記述して下さい。