1　課題 1

1.1　課題 1-1

BCC に含まれるプログラム開発を行う際に役立つコマンドラインツールを示す．

1.1.1　grep

ファイルや標準入力に含まれる文字列を検索する．このコマンドは以下のようにして使用する．

```
grep [-<オプション>] <検索文字列> [<ファイル>...]
```

また，検索文字列に指定できる正規表現を表 1 に示す．

表 1　検索文字列正規表現

記号	説明
^	行頭
$	行末に一致
.	任意の 1 文字
*	前の文字が任意の数現れたあと任意の文字がゼロ個以上現れるものに一致
+	前の文字が任意の数現れたあと任意の文字が 1 個以上現れるものに一致
{}	複数の文字や表現のグループ化
[]	かっこ内に現れる任意の 1 文字にのみ一致
[^]	かっこ内に現れない任意の 1 文字に一致
[-]	かっこ内に現れる-で区切られた文字の範囲の任意の 1 文字に一致
\	次の文字をワイルドカードではなくそのままの文字として扱う

1.1.2　touch

ファイルのタイムスタンプを更新する．指定されたファイルが存在しない場合は，新規ファイルが作成される．また，*や?などのワイルドカードを含むファイル名の指定もできる．このコマンドは以下のようにして使用する．

```
touch [<オプション>] <ファイル名> [<ファイル名>...]
```

1.2　課題 1-2

ローカル変数は宣言された関数内でのみ参照でき，記憶域期間は関数が終了するまでとなる．対してグローバル変数はプログラム内のどの関数からでも参照でき，プログラムが終了するまで内容が保持される．スタティック変数は宣言された関数内でのみ参照できる点ではローカル変数と同じだが，初期化は一回しか行われず，関数が終了しても内容が保持される性質を持つ．

1.3　課題 1-3

星型正多角形を描き回転させるプログラムをソースコード 1 に示す．半径 1 の円に沿った頂点を一つずつ飛ばして線を結んで描画することで，星型正多角形を描くように実装した．

ソースコード 1　星型正多角形の描画

```
1  #define N_VERTEX 7
2  double rot = 0.0;
3
4  void display(void) {
5      int i;
6      double theta, dt, x, y;
7      if (N_VERTEX % 2 == 1) {
8          printf("星型正%d角形は存在しません\n", N_VERTEX);
9          exit(0);
10     }
11     glClear(GL_COLOR_BUFFER_BIT);
12     glColor3d(1.0, 0.0, 0.0);
13     dt = 2.0 * M_PI / N_VERTEX;
14     theta = rot;
15     glBegin(GL_LINE_LOOP);
16     for (i = 0; i < N_VERTEX; i++) {
17         x = cos(theta);
18         y = sin(theta);
19         glVertex2d(x, y);
20         theta += 2 * dt;
21     }
22     glEnd();
23     glutSwapBuffers();
24     rot += 3 * M_PI / 180.0;
25 }
26
27 static void timer(int dummy) {
28     glutTimerFunc(10, timer, 0);
29     glutPostRedisplay();
30 }
```

1.4　課題 1-4

ソースコード 1 を変更して，完全グラフを描き回転させるようにしたプログラムをソースコード 2 に示す．各頂点から線で結ばれていない他の頂点へ線を引き，完全グラフを描画するように実装した．

ソースコード 2　完全グラフの描画

```
1    glBegin(GL_LINE_STRIP);
2    for (i = 0; i < N_VERTEX-1; i
         ++) {
3      theta = rot + dt*i;
4      for (delta = i+1; delta <
           N_VERTEX; delta++) {
5        x = cos(theta);
6        y = sin(theta);
7        glVertex2d(x, y);
8        x = cos(theta-delta*dt);
9        y = sin(theta-delta*dt);
10       glVertex2d(x, y);
11     }
12   }
```

1.5　課題 1-5

カージオイドのグラフを描画するプログラムの描画部をソースコード 3 に示す．また，サイクロイドの描画部分をソースコード 4，4 尖点の内サイクロイドの描画部分をソースコード 5 に示す．カージオイドと 4 尖点の内サイクロイドの描画部分においては三角関数の呼び出しがなるべく少なくなるように工夫した．それぞれのプログラムにおいて，ソースコード 6 のようなコードで目盛りの文字を描画した．（軸の長さや文字の位置，「PI」の表示は適宜変更した．）それぞれの描画結果を図 1，2，3 に示す．

ソースコード 3　カージオイドの描画部分

```
1  double x, y, theta, cos_theta;
2
3  glColor3d(1.0, 0.0, 0.0);
4  glBegin(GL_LINE_STRIP);
5  //グラフ描画
6  for (theta = 0; theta <= 2 *
       M_PI; theta += 0.05) {
7      cos_theta = cos(theta);
8      x = cos_theta * (1 +
           cos_theta);
9      y = sin(theta) * (1 +
           cos_theta);
10     glVertex2d(x, y);
11 }
12 glEnd();
```

ソースコード 4　サイクロイドの描画部分

```
1  for (theta = 0; theta <= 2 *
       M_PI; theta += 0.05) {
2      x = theta - sin(theta);
3      y = 1 - cos(theta);
4      glVertex2d(x, y);
5  }
```

ソースコード 5　4 尖点の内サイクロイドの描画部分

```
1  for (theta = 0; theta <= 2 *
       M_PI; theta += 0.05) {
2      x = cos(theta);
3      x = x * x * x;
4      y = sin(theta);
5      y = y * y * y;
6      glVertex2d(x, y);
7  }
```

ソースコード 6　目盛り文字の表示

```
1
2  char cMinus, cAbs;
3  glColor3d(0.0, 0.0, 0.0);
4  // X軸の目盛り文字
5  for (x = 1; x <= 2.0; x += 1.0)
       {
6      glRasterPos2d(x - 0.05,
           -0.2);
7      //文字をセット
8      cMinus = x < 0 ? '-' : ' ';
9      cAbs = '0' + abs(x);
10     //文字を描画
11     glutBitmapCharacter(
           GLUT_BITMAP_8_BY_13,
           cMinus);
12     glutBitmapCharacter(
           GLUT_BITMAP_8_BY_13,
           cAbs);
13 }
14 // Y軸の目盛り文字
15 for (y = -1; y <= 1; y += 2) {
16     glRasterPos2d(-0.275, y -
           0.05);
17     //文字をセット
18     cMinus = y < 0 ? '-' : ' ';
19     cAbs = '0' + abs(y);
20     //文字を描画
21     glutBitmapCharacter(
           GLUT_BITMAP_8_BY_13,
           cMinus);
22     glutBitmapCharacter(
           GLUT_BITMAP_8_BY_13,
           cAbs);
23 }
```

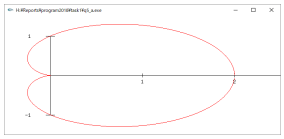

図 1　カージオイドグラフの出力

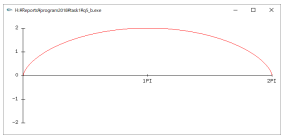

図 2　サイクロイドグラフの出力

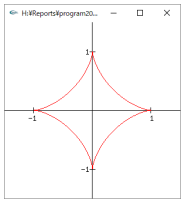

図 3　4 尖点の内サイクロイドグラフの出力

2　課題 2

2.1　課題 2-1

正四面体 ABCD について $\triangle BCD$ の重心 G を原点 O と一致させ，GA を x 軸，GB を y 軸とする座標系を考える．正四面体の一辺の長さを w とするとき．各頂点の 3 次元座標は以下のようになる．

$$\text{点 } A : \left(\sqrt{\frac{2}{3}}w, 0, 0\right) \qquad \text{点 } B : \left(0, \frac{\sqrt{3}}{3}w, 0\right)$$

$$\text{点 } C : \left(0, -\frac{\sqrt{3}}{6}w, \frac{1}{2}w\right) \qquad \text{点 } D : \left(0, -\frac{\sqrt{3}}{6}w, -\frac{1}{2}w\right)$$

2.2　課題 2-2

前問の正四面体を原点 O を中心とする半径 1 の球に内接させるために，三角錐 (正四面体) の重心を G' と置くと，三平方の定理より式 (1) となる．この方程式について代入し，w について解くと式 (3) となる．

$$GB^2 + GG' = BG'^2 \tag{1}$$

$$\left(\frac{\sqrt{3}w}{3}\right)^2 + \left(\frac{1}{3}\right) = 1 \tag{2}$$

$$w = \sqrt{\frac{2\sqrt{6}}{3}} \tag{3}$$

よって，各頂点の座標は次のようになる．また，この座標はテキストリスト 14 と等しい．

$$\text{点 } A : (1, 0, 0) \qquad \text{点 } B : \left(-\frac{1}{3}, \frac{2\sqrt{2}}{3}, 0\right)$$

$$\text{点 } C : \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, \frac{\sqrt{6}}{3}\right) \qquad \text{点 } D : \left(-\frac{1}{3}, -\frac{\sqrt{2}}{3}, -\frac{\sqrt{6}}{3}\right)$$

2.3　課題 2-3

キー入力によって台座の回転と，各関節の傾斜角を制御できるロボットアームのプログラムの変更部をソースコード 7 に示す．符号なし short 型の変数の各ビットをフラグとすることで，メモリの削減を図った．

ソースコード 7　制御可能なロボットアーム

```
1  /*キー ID*/
2  #define KEY_D 1
3  #define KEY_F 2
4  #define KEY_G 4
5
6  #define KEY_W 8
7  #define KEY_S 16
8
9  #define KEY_E 32
10 #define KEY_R 64
11 #define KEY_T 128
12
13 //押されたキーフラグ
14 unsigned short pressed = 0;
15
16 static void timer(int dummy) {
17   glutTimerFunc(10, timer, 0);
18
19   if (pressed & KEY_S) {
20     scale += 0.1;
21     glMatrixMode(GL_PROJECTION);
22     glLoadIdentity();
23     glOrtho(-scale, scale, -
           scale, scale, -10, 10);
24     glutPostRedisplay();
25     pressed = 0;
26   } else if (pressed & KEY_W) {
27     scale -= 0.1;
28     if (scale <= 0) scale = 0;
29     glMatrixMode(GL_PROJECTION);
30     glLoadIdentity();
31     glOrtho(-scale, scale, -
           scale, scale, -10, 10);
32     glutPostRedisplay();
33     pressed = 0;
34   } else if (pressed) {
35     //押されたボタンに応じて
           1度ずつ回転
36     rotAng[0] += (pressed &
           KEY_D) != 0 ? 1 : 0;
37     rotAng[1] += (pressed &
           KEY_F) != 0 ? 1 : 0;
```

```
38      rotAng[2] += (pressed &
            KEY_G) != 0 ? 1 : 0;
39      rotAng[0] -= (pressed &
            KEY_E) != 0 ? 1 : 0;
40      rotAng[1] -= (pressed &
            KEY_R) != 0 ? 1 : 0;
41      rotAng[2] -= (pressed &
            KEY_T) != 0 ? 1 : 0;
42      glutPostRedisplay();
43      pressed = 0;
44    }
45  }
46
47  //フラグセット
48  void keyin(unsigned char key,
        int x, int y) {
49    switch (key) {
50      case 'd':
51        pressed |= KEY_D;
52        break;
53      case 'f':
54        pressed |= KEY_F;
55        break;
56      case 'g':
57        pressed |= KEY_G;
58        break;
59      case 'e':
60        pressed |= KEY_E;
61        break;
62      case 'r':
63        pressed |= KEY_R;
64        break;
65      case 't':
66        pressed |= KEY_T;
67        break;
68      case 'w':
69        pressed |= KEY_W;
70        break;
71      case 's':
72        pressed |= KEY_S;
73        break;
74      case 'q':
75      case 'Q':
76        exit(0);
77      default:
78        break;
79    }
80  }
```

2.4 課題 2-4

タイマーコールバックを使って，アームロボットを制御するプログラムをソースコード 8 に示す．Z ボタンのフラグを新たに定義し，Z ボタンを押すことでダンスの再生・一時停止ができるようにした．三角関数を用いることでなめらかな動作になるように工夫した．

3 課題 3

3.1 課題 3-1

空間中の任意の 3 点 $P_i = (x_i, y_i, z_i), i = 1, 2, 3$ から $\triangle P_1P_2P_3$ の単位法線ベクトルをすべて求める．まず，$\triangle P_1P_2P_3$ の辺のうち，2 つの辺のベクトルを求める．

$$\overrightarrow{P_1P_2} = \vec{P_2} - \vec{P_1}$$
$$\overrightarrow{P_2P_3} = \vec{P_3} - \vec{P_2}$$

この 2 つのベクトルから全ての単位法線ベクトル $\vec{n}$ を求める式は以下のようになる．

$$\vec{n} = \pm \frac{\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}}{|\overrightarrow{P_1P_2} \times \overrightarrow{P_2P_3}|}$$

3.2 課題 3-2

空間中の任意の 3 点 $P_i = (x_i, y_i, z_i), i = 1, 2, 3$ が与えられたとき，$\triangle P_1P_2P_3$ の 3 点を反時計回りにたどる向きのベクトルから単位法線ベクトルを求めると，その単位法線ベクトルは面の表側を向くので，これによりその面の裏表を判別できる．

3.3 課題 3-3

Phong のモデルでの環境反射による光の強度 I_a は単純に環境反射係数と環境照明の積で表される．拡散反射による光の強度 I_d は拡散反射係数 k_d，光源への方向ベクトル L, 物体表面の方向ベクトル N, 光源の拡散反射成分 i_d とすると $I_d = k_d(L \cdot N)i_d$ で計算される．鏡面反射による光の強度 I_s は鏡面反射係数 k_s，鏡面反射方向ベクトル R，視点への方向ベクトル I，光源の鏡面反射成分 i_s，物体の光沢度 α としたとき $I_s = k_d(R \cdot I)^\alpha i_s$ で計算される．

これらの値から光の強度 I_p が $I_p = I_a + I_s + I_s$ と計算されるモデルが Phong の反射モデルである．

3.4 課題 3-4

正四面体をスムースシェーディングで表示するプログラムをソースコード 9 に示す．頂点が含まれるすべての面の法線ベクトルから平均を得ることで，頂点の法線ベクトルを求めた．

3.5 課題 3-5

テキストのリスト 27 を参考に，アームロボットを白色の光源でシェーディング表示するように変更したコードの追加部をソースコード 10 に示す．

4 感想

3 次元の描画プログラムを作成することはもっと難しいものだと考えていたが，OpenGL を使うことで簡単な物体なら手軽に描画できることがわかった．また，回転情報や移動情報が蓄積されることで，オブジェクト同士の連結処理もスムーズに書けることに感動した．

5 今後の改善案

コラム程度でも画像表示の方法やヒントが書いてあると，全体の総合課題の完成度が上がるのではないかと思います．

ソースコード 8　ダンスをするロボットアーム

```
1
2  #define KEY_Z 256
3
4  //ダンスフラグ
5  char danceToggle = 0;
6  //方向転換までの長さ
7  const int danceLength = 20;
8  //回転速度
9  const double rotSpeed = 10;
10
11 static void timer(int dummy) {
12     //回転方向転換までのタイマー
13     static int danceTimer;
14     //回転方向
15     static short dir_rot[3];
16     int i;
17     glutTimerFunc(10, timer, 0);
18     /*中略*/
19     } else if (pressed & KEY_Z) {
20         danceToggle = !danceToggle;
21         danceTimer = danceLength;
22     }
23     /*中略*/
24     if (danceToggle) {
25         if (danceTimer < danceLength) {
26             for (i = 0; i < 3; i++) {
27                 //なめらかに回転(cosをPIシフトして回す)
28                 rotAng[i] += rotSpeed * dir_rot[i] *
29                 (cos(danceTimer/(double)danceLength*M_PI*2+M_PI)+1)/10.0;
30                 //回転角の制限
31                 if (rotAng[i] > 90) {
32                     rotAng[i] = 90;
33                 } else if (rotAng[i] < -90) {
34                     rotAng[i] = -90;
35                 }
36             }
37             danceTimer++;
38         } else {
39             //回転方向の切り替え
40             for (i = 0; i < 3; i++) {
41                 dir_rot[i] = rand() % 2 * 2 - 1;
42             }
43             danceTimer = 0;
44         }
45         glutPostRedisplay();
46     }
47     pressed = 0;
48 }
```

ソースコード 9　正四面体のスムースシェーディング

```
GLdouble vP[4][3] = {{1.000, 0.0000, 0.000},
                     {-0.333, 0.943, 0.000},
                     {-0.333, -0.471, 0.816},
                     {-0.333, -0.471, -0.816}};
GLdouble nV[4][3];
GLdouble vert_n[4][3];
int tP[4][3] = {{0, 1, 2}, {0, 3, 1}, {0, 2, 3}, {1, 3, 2}};
int reversTP[4][3] = {{0, 1, 2}, {0, 1, 3}, {0, 2, 3}, {1, 2, 3}};

void display(void) {
    int i, j;
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 3; j++) {
            //各頂点に法線ベクトルを設定
            glNormal3dv(vert_n[tP[i][j]]);
            glVertex3dv(vP[tP[i][j]]);
        }
    }
    glEnd();
    glutSwapBuffers();
}

void init(void) {
    int i, j;
    GLdouble v1[3], v2[3], tmp[3];
    GLdouble n;
    /*中略*/
    //各面の法線ベクトルを求める
    for (i = 0; i < 4; i++) {
        vect_sub(vP[tP[i][0]], vP[tP[i][1]], v1);
        vect_sub(vP[tP[i][1]], vP[tP[i][2]], v2);
        cross(v1, v2, tmp);
        n = vect_norm(tmp);
        vect_scale(1.0 / n, tmp, nV[i]);
    }
    //頂点の法線ベクトルを求める
    for (i = 0; i < 4; i++) {
        vect_sum(nV[reversTP[i][0]], nV[reversTP[i][1]], tmp);
        vect_sum(tmp, nV[reversTP[i][2]], tmp);
        n = vect_norm(tmp);
        vect_scale(1.0 / n, tmp, vert_n[i]);
    }
}
```

ソースコード 10　アームロボットのシェーディング表示

```
GLfloat lP1[] = {0.0,1.0,2.0,1.0};
GLfloat lC1[] = {1.0,1.0,1.0,1.0};

void init(void) {
    /*中略*/
    glEnable(GL_LIGHTING);
    glLightfv(GL_LIGHT1, GL_POSITION, lP1);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, lC1);
    glLightfv(GL_LIGHT1, GL_SPECULAR, lC1);
    glEnable(GL_LIGHT1);
    glEnable(GL_CULL_FACE);
    /*中略*/
}
```