学籍番号202113517

所属 情報学群情報メディア創成学類 3年

氏名 高橋 健太郎

# 課題1

## プログラム

#include <cstdlib>

#include <cmath>

#include <vector>

*// 2次元ベクトルを扱うためのクラス*

**class** Vector2d

{

**public:**

**double** x, y;

Vector2d() { x = y = 0; }

Vector2d(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

**void** set(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

*// 長さを1に正規化する*

**void** normalize()

{

**double** len = length();

x /= len;

y /= len;

}

*// 長さを返す*

**double** length() { return sqrt(x \* x + y \* y); }

*// s倍する*

**void** scale(**const** **double** s)

{

x \*= s;

y \*= s;

}

*// 加算の定義*

Vector2d operator+(Vector2d v) { return Vector2d(x + v.x, y + v.y); }

*// 減算の定義*

Vector2d operator-(Vector2d v) { return Vector2d(x - v.x, y - v.y); }

*// 内積の定義*

**double** operator\*(Vector2d v) { return x \* v.x + y \* v.y; }

*// 代入演算の定義*

Vector2d **&**operator=(**const** Vector2d **&**v)

{

x = v.x;

y = v.y;

return (\*this);

}

*// 加算代入の定義*

Vector2d **&**operator+=(**const** Vector2d **&**v)

{

x += v.x;

y += v.y;

return (\*this);

}

*// 減算代入の定義*

Vector2d **&**operator-=(**const** Vector2d **&**v)

{

x -= v.x;

y -= v.y;

return (\*this);

}

*// 値を出力する*

**void** print() { printf("Vector2d(%f %f)\n", x, y); }

};

*// マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる*

Vector2d operator-(**const** Vector2d **&**v) { return (Vector2d(-v.x, -v.y)); }

*// ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる*

Vector2d operator\*(**const** **double** **&**k, **const** Vector2d **&**v) { return (Vector2d(k \* v.x, k \* v.y)); }

Vector2d operator\*(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x \* k, v.y \* k)); }

*// ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる*

Vector2d operator/(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x / k, v.y / k)); }

**int** main(**int** argc, **char** **\*\***argv)

{

*// =====================================================*

*// 2次元ベクトルクラス Vector2d の使い方の例*

*// =====================================================*

*// 2次元ベクトル(1, 2)の作成*

Vector2d v0(1, 2);

*// 作成した2次元ベクトルのx,y座標値を確認*

v0.print();

*// 2次元ベクトル(2, 4)の作成*

Vector2d v1(2, 4);

*// ベクトルの加算*

Vector2d v2 = v0 + v1;

*// 加算した結果の確認*

v2.print();

*// ベクトルの減算*

Vector2d v3 = v1 - v0;

*// 減算した結果の確認*

v3.print();

*// ベクトルの長さの確認*

printf("v3.length() = %lf\n", v3.length());

*// ベクトルのスカラー倍*

v3 = 5.0 \* v3;

*// スカラー倍した結果の確認*

v3.print();

*// ベクトルの長さの正規化*

v3.normalize();

*// 正規化した結果の確認*

v3.print();

*// ベクトルの長さの確認*

printf("v3.length() = %lf\n", v3.length());

*// =====================================================*

*// std::vector を要素数を変更可能な配列として使う例*

*// =====================================================*

std::vector<Vector2d> vec; *// 配列の宣言 （Vector2d 型のオブジェクトを格納できる）*

vec.push\_back(v0); *// 配列の末尾に v0 を追加*

vec.push\_back(v1); *// 配列の末尾に v1 を追加*

vec.push\_back(v2); *// 配列の末尾に v2 を追加*

printf("vec.size() = %lu\n", vec.size()); *// 配列に入っている要素数を確認*

*// 配列の先頭要素を取得*

Vector2d firstElement = vec[0];

*// 取得した要素の値を出力*

printf("firstElement=(%lf, %lf)\n", firstElement.x, firstElement.y);

*// 全ての要素を出力*

for (**unsigned** **int** i = 0; i < vec.size(); i++)

{

printf("vec[%d]=(%lf, %lf)\n", i, vec[i].x, vec[i].y);

}

*// ★課題：以下にコメント文で指示する内容のプログラムコードを追加すること*

*// (1) vec を空にする*

vec.clear();

*// (2) 次の2次元ベクトルを vec に格納する*

*// (5.0, 2.0), (3.2, -2.3), (4.1, 9.2), (-2.0, 4.0), (0.0, -2.7)*

vec.push\_back(Vector2d(5.0, 2.0));

vec.push\_back(Vector2d(3.2, -2.3));

vec.push\_back(Vector2d(4.1, 9.2));

vec.push\_back(Vector2d(-2.0, 4.0));

vec.push\_back(Vector2d(0.0, -2.7));

*// (3) 上記の2次元ベクトルを全て加算した結果を Vector2d vecSum に格納する*

Vector2d vecSum(0, 0);

for (**unsigned** **int** i = 0; i < vec.size(); i++)

{

vecSum += vec[i];

}

*// (4) vecSum の内容(x,yの値)と、vecSum の長さを出力する*

vecSum.print();

printf("vecSum.length() = %lf\n", vecSum.length());

*// Visual Studio でコンソールがすぐに閉じないようにするため*

*//system("pause");*

return 0;

}

## 実行結果

Vector2d(1.000000 2.000000)

Vector2d(3.000000 6.000000)

Vector2d(1.000000 2.000000)

v3.length() = 2.236068

Vector2d(5.000000 10.000000)

Vector2d(0.447214 0.894427)

v3.length() = 1.000000

vec.size() = 3

firstElement=(1.000000, 2.000000)

vec[0]=(1.000000, 2.000000)

vec[1]=(2.000000, 4.000000)

vec[2]=(3.000000, 6.000000)

Vector2d(10.300000 10.200000)

vecSum.length() = 14.495861

# 課題2

## プログラム

#include <cstdlib>

#include <cmath>

#include <vector>

#include <GLUT/glut.h>

*// 2次元ベクトルを扱うためのクラス*

**class** Vector2d

{

**public:**

**double** x, y;

Vector2d() { x = y = 0; }

Vector2d(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

**void** set(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

*// 長さを1に正規化する*

**void** normalize()

{

**double** len = length();

x /= len;

y /= len;

}

*// 長さを返す*

**double** length() { return sqrt(x \* x + y \* y); }

*// s倍する*

**void** scale(**const** **double** s)

{

x \*= s;

y \*= s;

}

*// 加算の定義*

Vector2d operator+(Vector2d v) { return Vector2d(x + v.x, y + v.y); }

*// 減算の定義*

Vector2d operator-(Vector2d v) { return Vector2d(x - v.x, y - v.y); }

*// 内積の定義*

**double** operator\*(Vector2d v) { return x \* v.x + y \* v.y; }

*// 代入演算の定義*

Vector2d **&**operator=(**const** Vector2d **&**v)

{

x = v.x;

y = v.y;

return (\*this);

}

*// 加算代入の定義*

Vector2d **&**operator+=(**const** Vector2d **&**v)

{

x += v.x;

y += v.y;

return (\*this);

}

*// 減算代入の定義*

Vector2d **&**operator-=(**const** Vector2d **&**v)

{

x -= v.x;

y -= v.y;

return (\*this);

}

*// 値を出力する*

**void** print() { printf("Vector2d(%f %f)\n", x, y); }

};

*// マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる*

Vector2d operator-(**const** Vector2d **&**v) { return (Vector2d(-v.x, -v.y)); }

*// ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる*

Vector2d operator\*(**const** **double** **&**k, **const** Vector2d **&**v) { return (Vector2d(k \* v.x, k \* v.y)); }

Vector2d operator\*(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x \* k, v.y \* k)); }

*// ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる*

Vector2d operator/(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x / k, v.y / k)); }

*// ================================================================================================*

std::vector<Vector2d> g\_ControlPoints; *// 制御点を格納する*

*// 表示部分をこの関数で記入*

**void** display(**void**)

{

glClearColor(1.0, 1.0, 1.0, 1.0); *// 消去色指定*

glClear(GL\_COLOR\_BUFFER\_BIT); *// 画面消去*

*// 制御点の描画*

glPointSize(5);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// 制御点を結ぶ線分の描画*

glColor3d(1.0, 0.0, 0.0);

glLineWidth(1);

glBegin(GL\_LINE\_STRIP);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// ★ ここにベジェ曲線を描画するコードを追加する*

if (g\_ControlPoints.size() >= 4)

{

if ((g\_ControlPoints.size() - 4) % 3 == 0)

{

glColor3d(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_STRIP);

Vector2d P(0, 0);

for (**unsigned** **int** i = 0; i <= g\_ControlPoints.size()-4; i += 3)

{

for (**double** t = 0; t <= 1.0; t += 0.1)

{

P = pow(1.0 - t, 3) \* g\_ControlPoints[i] + 3 \* t \* pow(1.0 - t, 2) \* g\_ControlPoints[i + 1] + 3 \* pow(t, 2) \* (1.0 - t) \* g\_ControlPoints[i + 2] + pow(t, 3) \* g\_ControlPoints[i + 3];

glVertex2d(P.x, P.y);

printf("i= '%d', ", i);

printf("t= '%f', ", t);

printf("P=(%lf, %lf)\n", P.x, P.y);

}

}

glEnd();

}

}

glutSwapBuffers();

}

**void** resizeWindow(**int** w, **int** h)

{

h = (h == 0) ? 1 : h;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

*// ウィンドウ内の座標系設定*

*// マウスクリックの座標と描画座標が一致するような正投影*

glOrtho(0, w, h, 0, -10, 10);

glMatrixMode(GL\_MODELVIEW);

}

*// キーボードイベント処理*

**void** keyboard(**unsigned** **char** key, **int** x, **int** y)

{

switch (key)

{

case 'q':

case 'Q':

case '\033':

exit(0); */\* '\033' は ESC の ASCII コード \*/*

default:

break;

}

glutPostRedisplay();

}

*// マウスイベント処理*

**void** mouse(**int** button, **int** state, **int** x, **int** y)

{

if (state == GLUT\_DOWN)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:

*// クリックした位置に制御点を追加*

g\_ControlPoints.push\_back(Vector2d(x, y));

break;

case GLUT\_MIDDLE\_BUTTON:

break;

case GLUT\_RIGHT\_BUTTON:

*// 末尾の制御点の削除*

if (!g\_ControlPoints.empty())

{

g\_ControlPoints.pop\_back();

}

break;

default:

break;

}

glutPostRedisplay(); *// 再描画*

}

}

*// メインプログラム*

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

{

glutInit(&argc, argv); *// ライブラリの初期化*

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE); *// 描画モードの指定*

glutInitWindowSize(800, 800); *// ウィンドウサイズを指定*

glutCreateWindow(argv[0]); *// ウィンドウを作成*

glutDisplayFunc(display); *// 表示関数を指定*

glutReshapeFunc(resizeWindow); *// ウィンドウサイズが変更されたときの関数を指定*

glutKeyboardFunc(keyboard); *// キーボードイベント処理関数を指定*

glutMouseFunc(mouse); *// マウスイベント処理関数を指定*

glutMainLoop(); *// イベント待ち*

return 0;

}

## 実行結果

スポーツゲーム, スポーツ が含まれている画像

自動的に生成された説明

# 課題3

## プログラム

#include <cstdlib>

#include <cmath>

#include <vector>

#include <GLUT/glut.h>

*// 2次元ベクトルを扱うためのクラス*

**class** Vector2d

{

**public:**

**double** x, y;

Vector2d() { x = y = 0; }

Vector2d(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

**void** set(**double** \_x, **double** \_y)

{

x = \_x;

y = \_y;

}

*// 長さを1に正規化する*

**void** normalize()

{

**double** len = length();

x /= len;

y /= len;

}

*// 長さを返す*

**double** length() { return sqrt(x \* x + y \* y); }

*// s倍する*

**void** scale(**const** **double** s)

{

x \*= s;

y \*= s;

}

*// 加算の定義*

Vector2d operator+(Vector2d v) { return Vector2d(x + v.x, y + v.y); }

*// 減算の定義*

Vector2d operator-(Vector2d v) { return Vector2d(x - v.x, y - v.y); }

*// 内積の定義*

**double** operator\*(Vector2d v) { return x \* v.x + y \* v.y; }

*// 代入演算の定義*

Vector2d **&**operator=(**const** Vector2d **&**v)

{

x = v.x;

y = v.y;

return (\*this);

}

*// 加算代入の定義*

Vector2d **&**operator+=(**const** Vector2d **&**v)

{

x += v.x;

y += v.y;

return (\*this);

}

*// 減算代入の定義*

Vector2d **&**operator-=(**const** Vector2d **&**v)

{

x -= v.x;

y -= v.y;

return (\*this);

}

*// 値を出力する*

**void** print() { printf("Vector2d(%f %f)\n", x, y); }

};

*// マイナスの符号の付いたベクトルを扱えるようにするための定義 例：b=(-a); のように記述できる*

Vector2d operator-(**const** Vector2d **&**v) { return (Vector2d(-v.x, -v.y)); }

*// ベクトルと実数の積を扱えるようにするための定義 例： c=5\*a+2\*b; c=b\*3; のように記述できる*

Vector2d operator\*(**const** **double** **&**k, **const** Vector2d **&**v) { return (Vector2d(k \* v.x, k \* v.y)); }

Vector2d operator\*(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x \* k, v.y \* k)); }

*// ベクトルを実数で割る操作を扱えるようにするための定義 例： c=a/2.3; のように記述できる*

Vector2d operator/(**const** Vector2d **&**v, **const** **double** **&**k) { return (Vector2d(v.x / k, v.y / k)); }

*// ================================================================================================*

std::vector<Vector2d> g\_ControlPoints; *// 制御点を格納する*

*// 表示部分をこの関数で記入*

**void** display(**void**)

{

glClearColor(1.0, 1.0, 1.0, 1.0); *// 消去色指定*

glClear(GL\_COLOR\_BUFFER\_BIT); *// 画面消去*

*// 制御点の描画*

glPointSize(5);

glColor3d(0.0, 0.0, 0.0);

glBegin(GL\_POINTS);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// 制御点を結ぶ線分の描画*

glColor3d(1.0, 0.0, 0.0);

glLineWidth(1);

glBegin(GL\_LINE\_STRIP);

for (**unsigned** **int** i = 0; i < g\_ControlPoints.size(); i++)

{

glVertex2d(g\_ControlPoints[i].x, g\_ControlPoints[i].y);

}

glEnd();

*// ★ ここにベジェ曲線を描画するコードを追加する*

if (g\_ControlPoints.size() >= 4)

{

if ((g\_ControlPoints.size() - 4) % 3 == 0)

{

glColor3d(0.0, 0.0, 1.0);

glBegin(GL\_LINE\_STRIP);

Vector2d P(0, 0);

for (**unsigned** **int** i = 0; i <= g\_ControlPoints.size() - 4; i += 3)

{

for (**double** t = 0; t <= 1.0; t += 0.01)

{

P = pow(1.0 - t, 3) \* g\_ControlPoints[i] + 3 \* t \* pow(1.0 - t, 2) \* g\_ControlPoints[i + 1] + 3 \* pow(t, 2) \* (1.0 - t) \* g\_ControlPoints[i + 2] + pow(t, 3) \* g\_ControlPoints[i + 3];

glVertex2d(P.x, P.y);

*//printf("i= '%d', ", i);*

*//printf("t= '%f', ", t);*

*//printf("P=(%lf, %lf)\n", P.x, P.y);*

}

}

glEnd();

}

}

*//法線ベクトルの描画*

if (g\_ControlPoints.size() >= 4)

{

if ((g\_ControlPoints.size() - 4) % 3 == 0)

{

Vector2d P(0, 0);

Vector2d Ph(0, 0);

Vector2d Phh(0, 0);

for (**unsigned** **int** i = 0; i <= g\_ControlPoints.size() - 4; i += 3)

{

for (**double** t = 0; t <= 1.0; t += 0.01)

{

glColor3d(0.0, 1.0, 0.0);

glBegin(GL\_LINES);

P = pow(1.0 - t, 3) \* g\_ControlPoints[i] + 3 \* t \* pow(1.0 - t, 2) \* g\_ControlPoints[i + 1] + 3 \* pow(t, 2) \* (1.0 - t) \* g\_ControlPoints[i + 2] + pow(t, 3) \* g\_ControlPoints[i + 3];

Ph = -3 \* pow(1.0 - t, 2) \* g\_ControlPoints[i] + (9 \* pow(t, 2) - 12 \* t + 3) \* g\_ControlPoints[i + 1] + (-9 \* pow(t, 2) + 6 \* t) \* g\_ControlPoints[i + 2] + 3 \* pow(t, 2) \* g\_ControlPoints[i + 3];

Ph.normalize();

Ph.scale(100);

Phh.x = -Ph.y;

Phh.y = Ph.x;

Phh += P;

glVertex2d(P.x, P.y);

glVertex2d(Phh.x, Phh.y);

printf("i= '%d', ", i);

printf("t= '%f', \n", t);

printf("P=(%lf, %lf)\n", P.x, P.y);

printf("Ph=(%lf, %lf)\n", Ph.x, Ph.y);

printf("Ph-Length=%f\n", Ph.length());

printf("Phh=(%lf, %lf)\n\n", Phh.x, Phh.y);

glEnd();

}

}

}

}

glutSwapBuffers();

}

**void** resizeWindow(**int** w, **int** h)

{

h = (h == 0) ? 1 : h;

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

*// ウィンドウ内の座標系設定*

*// マウスクリックの座標と描画座標が一致するような正投影*

glOrtho(0, w, h, 0, -10, 10);

glMatrixMode(GL\_MODELVIEW);

}

*// キーボードイベント処理*

**void** keyboard(**unsigned** **char** key, **int** x, **int** y)

{

switch (key)

{

case 'q':

case 'Q':

case '\033':

exit(0); */\* '\033' は ESC の ASCII コード \*/*

default:

break;

}

glutPostRedisplay();

}

*// マウスイベント処理*

**void** mouse(**int** button, **int** state, **int** x, **int** y)

{

if (state == GLUT\_DOWN)

{

switch (button)

{

case GLUT\_LEFT\_BUTTON:

*// クリックした位置に制御点を追加*

g\_ControlPoints.push\_back(Vector2d(x, y));

break;

case GLUT\_MIDDLE\_BUTTON:

break;

case GLUT\_RIGHT\_BUTTON:

*// 末尾の制御点の削除*

if (!g\_ControlPoints.empty())

{

g\_ControlPoints.pop\_back();

}

break;

default:

break;

}

glutPostRedisplay(); *// 再描画*

}

}

*// メインプログラム*

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

{

glutInit(&argc, argv); *// ライブラリの初期化*

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE); *// 描画モードの指定*

glutInitWindowSize(800, 800); *// ウィンドウサイズを指定*

glutCreateWindow(argv[0]); *// ウィンドウを作成*

glutDisplayFunc(display); *// 表示関数を指定*

glutReshapeFunc(resizeWindow); *// ウィンドウサイズが変更されたときの関数を指定*

glutKeyboardFunc(keyboard); *// キーボードイベント処理関数を指定*

glutMouseFunc(mouse); *// マウスイベント処理関数を指定*

glutMainLoop(); *// イベント待ち*

return 0;

}

## 実行結果

